A MODEL-BASED SYSTEMS ENGINEERING METHODOLOGY TO MAKE ENGINEERING ANALYSIS OF DISCRETE-EVENT LOGISTICS SYSTEMS MORE COST-ACCESSIBLE.

A Thesis
Presented to
The Academic Faculty

by

George G. Thiers

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial & Systems Engineering

Georgia Institute of Technology
August 2014

Copyright © 2014 by George G. Thiers

A MODEL-BASED SYSTEMS ENGINEERING METHODOLOGY TO MAKE ENGINEERING ANALYSIS OF DISCRETE-EVENT LOGISTICS SYSTEMS MORE COST-ACCESSIBLE.

Approved by:

Dr. Leon McGinnis, Advisor
School of Industrial & Systems Engineering
Georgia Institute of Technology

Dr. Chris Paredis
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Rahul Basole
School of Interactive Computing
Georgia Institute of Technology

Dr. Edward Huang
Systems Engineering & Operations Research
George Mason University

Dr. Christos Alexopoulos
School of Industrial & Systems Engineering
Georgia Institute of Technology

Date Approved: May 2014

TABLE OF CONTENTS

LIST OF TABLES . . . . **vi**

LIST OF FIGURES . . . . **vii**

SUMMARY . . . . **xi**

I INTRODUCTION **1**

1.1 Definitions 1

1.2 The Methodology 6

1.2.1 Process 6

1.2.2 An Automation Method 8

1.2.3 An Abstraction Method 11

1.2.4 A Formalization Method 13

1.2.5 Contributions 14

1.3 Arguing a Hypothesis 15

1.4 Prerequisite Knowledge and Boundary 17

II PRIOR WORK **19**

2.1 Prior Work with the Same Goal 19

2.1.1 Decision-Support Systems 19

2.1.2 Reuse of Concrete Analysis Models 20

2.2 Prior Work for the Methodology's Process 21

2.3 Prior Work for the Methodology's Methods 24

2.3.1 Model-Driven Architecture of Software 24

2.3.2 Automation 26

2.3.3 Abstraction 27

2.3.4 Formalization 29

2.4 Prior Work for the Methodology's Tool of a Token-Flow Network Definition 31

2.4.1 Petri Nets 31

2.4.2 UML 2.0 Activities 32

2.5 Prior Work for the Methodology's Tool of a Question Definition 34

2.5.1 Literature Search for Published Documents 35

2.5.2 Relational Database Search 38

2.6 Summary 39

III TOKEN-FLOW NETWORK: STRUCTURE 41

3.1 Basic Structure 41

3.2 Tokens 44

3.3 Flow 45

3.4 Interfaces 48

3.5 Levels of Abstraction 50

3.6 Analysis Semantics: Variables, Constraints, Objectives, and Observations . 55

3.7 Summary 57

IV TOKEN-FLOW NETWORK: BEHAVIOR 58

4.1 Time, Behavior, State Change, and Events 58

4.2 Behavioral Model of a Process 61

4.3 Hosting Behavior at a FlowNode 66

4.4 Resources 69

4.5 Plant/Control Interface in a Token-Flow Network 73

4.5.1 Admission of Tokens into FlowNodes 74

4.5.2 Dispatching Tokens from FlowNodes 76

4.5.3 Routing a Dispatched Token 81

4.5.4 How a Token Crosses a FlowEdge 83

4.6 Summary 85

V SEMANTICS OF A WELL-FORMED QUESTION 87

5.1 A Taxonomy of Questions 89

5.2 A Taxonomy of Questions about a Token-Flow Network 92

5.3 A Question's Subject and Predicate Modifiers 95

5.3.1 Questions about Describing Structure at a Single Point in Time . . 96

5.3.2 Questions about Describing Behavior Spanning Multiple Points in Time 97

5.3.3 Questions about Predicting Behavior Spanning Multiple Points in Time 99

5.3.4 Questions about Controlling Behavior Spanning Multiple Points in Time 102

5.4 Analysis Models and Answers 104

5.5 Summary 109

VI ABSTRACTION AND AUTOMATION METHODS 111

6.1 First-Stage Abstraction by Stereotype Application 112

6.2 Modeling Levels of Abstraction 113

6.3 Second-Stage Analysis Model-Building and Automation 117

6.3.1 Example: Optimization Analysis 120

6.3.2 Example: Statistical Regression Analysis 123

6.3.3 Example: Discrete-Event Simulation Analysis 125

6.4 Model-to-Model Transformation Tools 130

6.5 Summary 131

VII EXAMPLES 133

7.1 Example using Basic Structural Semantics 134

7.2 Example using Behavioral Process Semantics 139

VIII CONCLUSIONS AND FUTURE WORK 145

8.1 Contributions 145

8.2 Boundary 146

8.2.1 Boundary Induced by the Formalization Method 146

8.2.2 Boundary Induced by the Abstraction Method 147

8.2.3 Boundary Induced by the Automation Method 150

8.3 Future Work 151

8.4 What If the Hypothesis Is Untrue? 153

APPENDIX A — PETRI NETS 156

APPENDIX B — UML 2.0 ACTIVITIES 159

APPENDIX C — INTERPRETING SEQUENCING DEPENDENCIES IN A PROCESS NETWORK 162

REFERENCES 168

LIST OF TABLES

1 A Taxonomy of Questions Specialized to a Token-Flow Network. 93

LIST OF FIGURES

1 [Pegden et al., 1995]'s Example of a Discrete-Event Simulation Analysis Model in the SIMAN Language. 3

2 [Estefan, 2008]'s Elements of a Methodology and their Relationships. 5

3 The Process in this Dissertation's Methodology. 6

4 [Roedler, 2002]'s Illustration of ISO/IEC 15288 Process Categories. 7

5 Where an Automation Method Might Be Applied to the Process in this Dissertation's Methodology. 8

6 Example: Discrete-Event Simulation Analysis Model in Simio Language v5.81. 9

7 Adding Intermediate Steps to the Process in this Dissertation's Methodology for the Abstraction Method. 12

8 Process, Methods, and Tools in this Dissertation's Methodology. 14

9 [Law and Kelton, 2000]'s Ten Steps of a Typical Sound Simulation Study. . 16

10 [Friedenthal et al., 2011]'s Illustration of the OOSEM System Development Process. 22

11 [Friedenthal et al., 2011]'s Refinement of the *Specify and Design System* Process in Figure 10. 23

12 [Hazelrigg, 1998]'s Framework for Decision-Based Engineering Design. . . . 24

13 [Friedenthal et al., 2011]'s Partial Systems Engineering Standards Taxonomy. 30

14 Prior Work on Modeling Questions: A Definition of a Published Document per *Proquest ABI/INFORM Complete*. 36

15 Prior Work on Modeling Questions: User Interface for Constructing a *Proquest ABI/INFORM Complete* Document Query. 37

16 Prior Work on Modeling Questions: Schema for a Proquest ABI/INFORM Complete Document Query. 37

17 Prior Work on Modeling Questions: A Definition of a Relational Database. 38

18 Excerpt from a Defining BNF Grammar for SQL:1999. 39

19 Token-Flow Network, Structural Definition: Basic Structure. 41

20 An Example of a Network Instance Conforming to Basic Structural Semantics. 42

21 Token-Flow Network, Structural Definition: Tokens. 44

22 Token-Flow Network, Structural Definition: Flow. 46

23 An Example of a FlowNetwork Instance Conforming to Flow Semantics. . . 47

24 Token-Flow Network, Structural Definition: Interfaces. 49

25 An Example of Integrated Network and FlowNetwork Instances using Interface Semantics. 49

26 Token-Flow Network, Structural Definition: Levels of Abstraction. 50

27 An Example of a Nested Network Instance Conforming to Level Of Abstraction Semantics. 51

28 An Example of Integrated Nested Network and FlowNetwork Instances Conforming to Level Of Abstraction Semantics. 51

29 An Example of Consumption at a Node's FlowNode Interfaces. 52

30 An Example of Consumption at a Node's FlowNode Interfaces, Allocated to FlowNodes in a Nested Network. 53

31 Incorrect Example of a FlowNode with FlowNode Interfaces. 54

32 Corrected Example of a FlowNode with FlowNode Interfaces. 54

33 Token-Flow Network, Structural Definition: Metrics 56

34 Token-Flow Network, Structural Definition: Additional Flow Constraints. . 56

35 Token-Flow Network, Behavioral Definition: Time and Events. 59

36 Token-Flow Network, Behavioral Definition: Process Behavioral Model (in isolation). 61

37 Example: A Process Network Instance Sequenced Explicitly using SequencingDependencies. 62

38 Token-Flow Network, Behavioral Definition: Process Behavioral Model (integrated). 63

39 Example: A Process Network Instance Sequenced Implicitly using Token Flows. 64

40 Token-Flow Network, Behavioral Definition: Process Subclasses. 65

41 Token-Flow Network, Behavioral Definition: Hosting Behavior. 67

42 Example: ConversionNode and ConversionEdge Hosting Behavior. 68

43 Token-Flow Network, Behavioral Definition: Resources and Corresponding Tokens. 70

36 Token-Flow Network, Behavioral Definition: Process Behavioral Model (in isolation). 71

44 A Modeling Pattern of Plant/Control Separation. 73

45 Token-Flow Network, Behavioral Definition: Blocked FlowNodes and FlowEdges. 75

46 Token-Flow Network, Behavioral Definition: Process Subclasses with Output Production Rules. 77

47 Example: A Process Network Instance with Three Levels of Parent/Child Hierarchy. 79

48 Example: A Process Network Instance with Four Levels of Parent/Child Hierarchy. 79

49 Token-Flow Network, Behavioral Definition: Routing Control Policies for Dispatching Tokens from a FlowNode onto an Outgoing FlowEdge. 81

50 Token-Flow Network, Behavioral Definition: Flow Control Policies for a Token crossing a FlowEdge. 83

51 Example: A Process Network Instance with Hosted Storage Processes and Hosted Move Processes. 84

52 A Generalized View of the Process in this Dissertation's Methodology. . . . 87

53 A Taxonomy of Questions. 90

54 Example: State Values Available at Two Points in Time. 90

55 Example: State Values Available at Many Points in Time. 91

56 Example of Reed-Kellogg Sentence Diagramming. 95

57 Pattern for a Question about a Describing Structure at a Single Point in Time. 96

58 Reed-Kellogg Diagram for an Example of a Question about Describing Structure at a Single Point in Time. 96

59 Pattern for a Question about Describing Behavior Spanning Multiple Points in Time. 98

60 Reed-Kellogg Diagram for a Question about Describing Behavior Spanning Multiple Points in Time. 99

61 Pattern for a Question about Predicting Behavior Spanning Multiple Points in Time. 100

62 Reed-Kellogg Diagram for a Question about Predicting Behavior Spanning Multiple Points in Time. 101

63 Reed-Kellogg Diagram for a Question about Predicting Behavior Spanning Multiple Points in Time, with an Embedded Behavioral Model. 102

64 Pattern for a Question about Controlling Behavior Spanning Multiple Points in Time. 103

65 Reed-Kellogg Diagram for a Question about Controlling Behavior Spanning Multiple Points in Time. 104

66 An Objects-and-Relationships View of The Process in this Dissertation's Methodology. 104

67 The Process in this Dissertation's Methodology, with Multiple Answering Analysis Models and Analyst Choice. 106

68 The Process in this Dissertation's Methodology, with Question and Answer Qualifications. 108

69 A Complete Illustration of the Process in this Dissertation's Methodology. . 109

8 Process, Methods, and Tools in this Dissertation's Methodology. 111

70 [Berner et al., 1999]'s Classification of UML Stereotypes. 112

71 OMG's Layered Abstraction Levels for Object-Oriented Modeling. 114

72 Example: A Simple SysML User Model. 114

73 Instance Models Conforming to the SysML User Model in Figure 72. 115

74 A Model-to-Model Transformation Paradigm. 118

75 Model-to-Model Transformation for MDA Object-Oriented Code Generation. 118

76 Model-to-Model Transformations in MDA Object-Oriented Code Generation and MBSE Analysis Model Generation Use Cases. 119

77 M_2-Level Definition of an AMPL Optimization Analysis Model. 120

78 M_2-Level Definition of a Statistical Analysis Model. 123

79 An M_1-Level Partial Definition of Multiple Linear Regression Analysis. . . . 124

80 [Miller et al., 2004]'s Four Defining Paradigms for Discrete-Event Modeling. 126

81 An M_1-Level Partial Definition of the SIMAN Language. 128

82 An M_0-Level Instance of a SIMAN Model and a SIMAN Experiment. . . . 129

69 A Complete Illustration of the Process in this Dissertation's Methodology. . 133

83 Example 1: SysML User Model of a Supply Chain. 134

84 Reed-Kellogg Diagram for the Question in Example 1. 135

85 Example 1: Applying Basic Network Stereotypes to a Supply Chain User Model. 136

86 Example 2: SysML Activity Model of a Manufacturing Process Instance. . . 139

87 Reed-Kellogg Diagram for the Question in Example 2. 140

88 Example 2: Apply Process Network Stereotypes to a SysML Activity Model of a Manufacturing Process. 141

89 Example: A Recursive Design Pattern for System Structure. 155

90 Higher-level Coordination Concepts for Firing Transitions in a Petri Net. . 157

91 A Portion of the UML 2.0 Activity Metamodel. 159

92 Example: A Simple Process Network to Demonstrate an Algorithm for Interpreting Sequencing Dependencies. 164

93 Example: A More Complex Process Network to Demonstrate an Algorithm for Interpreting Sequencing Dependencies. 165

SUMMARY

This dissertation supports human decision-making with a Model-Based Systems Engineering methodology enabling engineering analysis, and in particular Operations Research analysis of discrete-event logistics systems, to be more widely used in a cost-effective and correct manner. A methodology is a collection of related processes, methods, and tools, and the process of interest is posing a question about a system model and then identifying and building answering analysis models. Methods and tools are the novelty of this dissertation, which when applied to the process will enable the dissertation's goal.

One method which directly enables the goal is adding automation to analysis model-building. Another method is abstraction, to make explicit a frequently-used bridge to analysis and also expose analysis model-building repetition to justify automation. A third method is formalization, to capture knowledge for reuse and also enable automation without human interpreters. The methodology, which is itself a contribution, also includes two supporting tool contributions.

A tool to support the abstraction method is a definition of a token-flow network, an abstract concept which generalizes many aspects of discrete-event logistics systems and underlies many analyses of them. Another tool to support the formalization method is a definition of a well-formed question, the result of an initial study of semantics, categories, and patterns in questions about models which induce engineering analysis. This is more general than queries about models in any specific modeling language, and also more general than queries answerable by navigating through a model and retrieving recorded information.

A final contribution follows from investigating tools for the automation method. Analysis model-building is a model-to-model transformation, and languages and tools for model-to-model transformation already exist in Model-Driven Architecture of software. The contribution considers if and how these tools can be re-purposed by contrasting software object-oriented code generation and engineering analysis model-building. It is argued that

both use cases share a common transformation paradigm but executed at different relative levels of abstraction, and the argument is supported by showing how several Operations Research analyses can be defined in an object-oriented way across multiple layered *instance-of* abstraction levels.

Enabling Operations Research analysis of discrete-event logistics systems to be more widely used in a cost-effective and correct manner requires considering fundamental questions about what knowledge is required to answer a question about a system, how to formally capture that knowledge, and what that capture enables. Developments here are promising, but provide only limited answers and leave much room for future work.

CHAPTER I

INTRODUCTION

The objective of this research is to support human decision-making. That can be hard for many reasons, including complexity of the system which is the subject of decisions. One resolution is to only make simple systems, which seems to disagree with the human spirit. Another way is to augment the human brain with a computer and make use of modeling and analysis. Modeling and analysis can help with the description, prediction, control, and design of a complex system's structure and behavior, but can also be expensive [1]. The cost can be so high that modeling and analysis are often forgone, leading to missed opportunities at best [2] and catastrophic failures at worst. This dissertation supports human decision-making with a Model-Based Systems Engineering methodology enabling engineering analysis, and in particular Operations Research analysis of discrete-event logistics systems, to be more widely used in a cost-effective and correct manner.

1.1 Definitions

Analysis is often defined as the opposite of synthesis - synthesis is combining simple elements to derive something more complex, and analysis is breaking what is complex into simpler elements [3]. The ultimate purpose of analysis is understanding, at any of the hierarchical levels identified by [Rouse, 2009]:

- *Describe* past observations
- *Classify* past observations
- *Predict* future observations

[1] Indirect evidence for this claim is the size of teams and scope of work of INFORMS Wagner Prize finalists, documented in the *Interfaces* September/October special issue each year. The introduction to the most recent is [Butler and Robinson, 2013]. Direct evidence for the claim is elusive in published academic literature but readily available in non-peer-reviewed sources such as [LaValle et al., 2010].

[2] A representative non-peer-reviewed source is [Kiron et al., 2011].

[3] Many definitions of *analysis* are collected by the Stanford Encyclopedia of Philosophy: *http://plato.stanford.edu/entries/analysis/s1.html*, viewed 09jan2014.

- *Control* future observations
- *Design* future observations

This taxonomy is fundamental and will be relied on heavily in this dissertation. For simplicity, *describe* and *classify* are combined and *design* is left out-of-scope, so levels of understanding as used here are describing past observations, predicting future observations, and controlling future observations. Note that these levels have an inherent sequence - only after behavior can be described can it be predicted, and only after behavior can be predicted can it be controlled (also called *prescribed*). What follows are three examples of Operations Research analysis used to support each of the describe, predict, and control levels of understanding.

. . . . . .

An analysis which can be used to describe past observations is statistical regression. An analysis model for multiple linear regression, both vectorized and indexed, is:

$$y = X\beta + \epsilon \qquad \textbf{OR} \qquad y_i = \beta_1 x_{i,1} + \ldots + \beta_p x_{i,p} + \epsilon_i \qquad i = 1, \ldots, n$$
$$\epsilon \sim N\left(0, \sigma^2 I\right) \qquad \qquad \epsilon_i \sim N(0, \sigma^2)$$

Independent observations are made of p predictors $x_{i,1}, \ldots, x_{i,p}$ and one response y_i. The analysis' purpose is to fit a reduced-order description by estimating linear coefficients $\beta_1, \ldots, \beta_p$, plus σ^2 for error. Solution algorithms include least-squares optimization, maximum likelihood estimation, and possibly Bayesian methods (which require enhancing the model with prior distributions for β and σ^2). Solvers include packaged functions in R, SAS, JMP, Minitab, Excel, and more. The fitted regression line is a reduced-order model describing past and present observations, which under certain assumptions can be used to predict future observations.

. . . . . .

An analysis which can be used to predict future observations is simulation. Since there exist many ways to model behavior which can be simulated, to the best of the author's

knowledge there is no canonical model for simulation analysis. An example of a *discrete-event* simulation analysis model from [Pegden et al., 1995, p.116] in the SIMAN language is shown in figure 1.

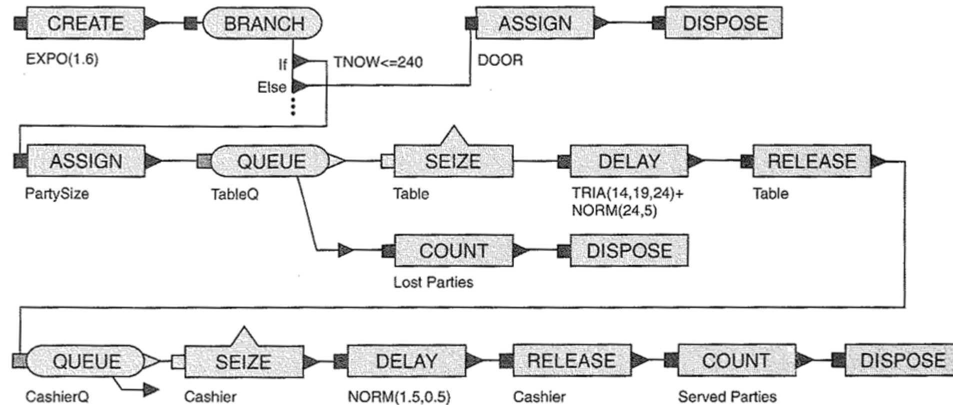


Figure 1: [Pegden et al., 1995]'s Example of a Discrete-Event Simulation Analysis Model in the SIMAN Language. The model concerns customers moving through a restaurant.

SIMAN enables expressing a discrete-event simulation analysis model either graphically or programmatically. The language includes ten basic processes (ten different block shapes) with over forty concrete realizations. Many discrete-event simulation languages enable separating a model from experiments on the model, such that one model can be reused to answer multiple questions via different experiments. An experiment on the model in figure 1 from [Pegden et al., 1995, p.118] is:

```
BEGIN;
PROJECT,            Sample Problem 3.4,SM;
ATTRIBUTES:         PartySize;
VARIABLES:          Door,10000;
QUEUES:             TableQ:
                    CashierQ;
RESOURCES:          Table,50:
                    Cashier;
COUNTERS:           LostParties:
                    ServedParties;
DSTATS:             NR(Table),Number of Busy Tables:
                    NQ(TableQ),# of Waiting Parties:
                    NR(Cashier)*100,Cashier Utilization:
                    NQ(CashierQ),# Waiting for Cashier;
REPLICATE,          1;
END;
```

Solution algorithms are generally time-stepping algorithms utilizing a simulation clock, event queues, state transitions, and statistical estimation algorithms for outputs. Solvers

include AnyLogic, Arena, SimEvents, Simio, FlexSim, Tecnomatix, and more; users can even write their own solver using the Visual Basic code in [Allen, 2011, ch.9].

. . . . . .

An analysis which can be used to control future observations is optimization. An analysis model for linear optimization, both vectorized and indexed, is:

$$
\begin{aligned}
&\min \quad c^T x && \textbf{OR} && \min \quad \sum_{i=1}^{n} c_i x_i && \\
&s.t. \quad Ax = b && && s.t. \quad \sum_{i=1}^{n} a_{ij} x_i \le b_j && j = 1, \ldots, m \\
&\qquad\;\; x \ge 0 && && \qquad x_i \ge 0 && i = 1, \ldots, n
\end{aligned}
$$

Controllable variables are $x_1, \ldots, x_n$, their values may be linearly constrained, and the objective is a cost-weighted sum of the variables. An optimization analysis' purpose is to find values for the variables which satisfy all constraints and optimize the objective. Solution algorithms include the simplex method, interior point methods, branch-and-bound for integer variables, and decomposition methods for large-scale cases. Solvers include CPLEX, Gurobi, Xpress, Excel, and more.

. . . . . .

Discrete-event logistics system is an abstract term for industrial engineering systems including manufacturing, supply chains, transportation, warehouses, healthcare delivery, and more. These systems' behavior can be described using the concepts of discrete events (any state change, such as the beginning or ending of a process' execution) and logistics (movement of goods, services, information, payments, and more). A low-level behavioral model for discrete-event logistics systems is an event-triggered state machine, and higher-level behavioral models such as a process can also be used, which may also have a lower-level state machine representation.

A **model** can be defined as "*a representation of a selected part of the world*" [4] or as a description or characterization. **Model-Based Systems Engineering (MBSE)**

[4] *http://plato.stanford.edu/entries/models-science*, viewed 17april2014.

is defined by [INCOSE, 2007, p.15] as "*the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases.*"
A **methodology** is defined by [Estefan, 2008, p.1-3]:

- "A **Methodology** is a collection of related processes, methods, and tools."
- "A **Process** is a logical sequence of tasks performed to achieve a particular objective. A process defines WHAT is to be done, without specifying HOW each task is performed."
- "A **Method** consists of techniques for performing a task, in other words, it defines the HOW of each task."
- "A **Tool** is an instrument that, when applied to a particular method, can enhance the efficiency of the task; provided it is applied properly and by somebody with proper skills and training. The purpose of a tool should be to facilitate the accomplishment of the HOW."
- "A **MBSE Methodology** can be characterized as the collection of related processes, methods, and tools used to support the discipline of systems engineering in a 'model-based' or 'model-driven' context.

[Estefan, 2008, p.3] illustrates methodology elements and their relationships in figure 2.

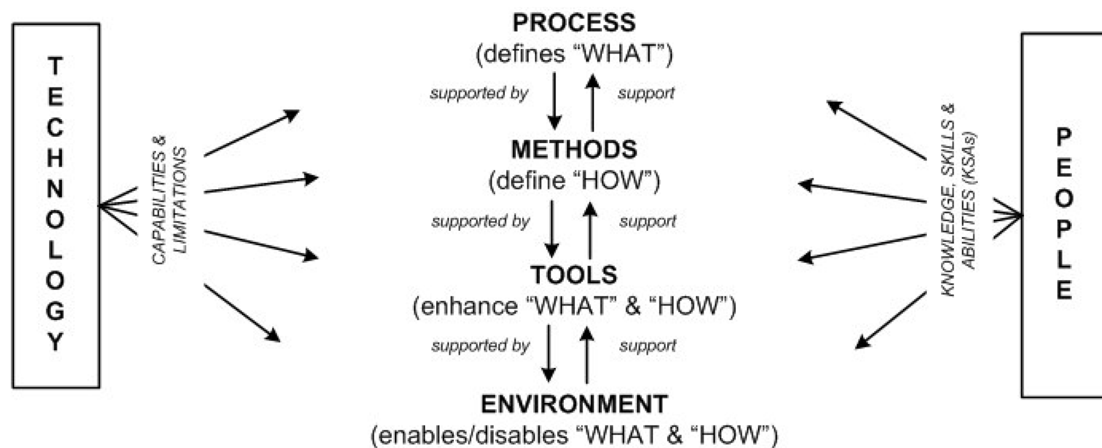


Figure 2: [Estefan, 2008]'s Elements of a Methodology and their Relationships.

1.2 *The Methodology*

Given the definitions in the previous section, next consider the methodology which is the core contribution of this dissertation. A methodology is a collection of related processes, methods, and tools, and each category is explained in the sections below.

1.2.1 Process

The process of interest is posing a question about a system model and then identifying and building answering analysis models, illustrated in figure 3.

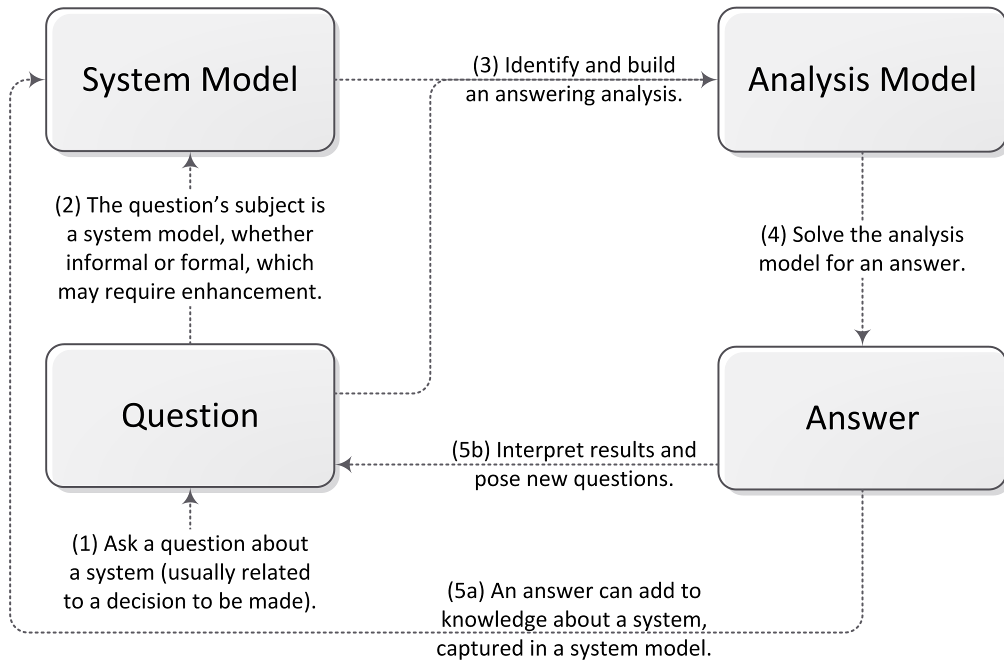


Figure 3: The Process in this Dissertation's Methodology.

Step 1 shows that the process is question-based - analysis is performed for a purpose, and a question contains that purpose. In step 2, a question may concern parts of a system which are not well-understood, whether due to missing low-level data or higher-level knowledge about structure and behavior. In this case, types of answering analysis may be identified but analysis models cannot be built until missing system knowledge is collected or computed. Step 2 may also be explained as making a question and its context precise and unambiguous enough that answering analysis models can be built.

In step 3, identifying answering analysis and building an analysis model are traditionally

lumped as ‘formulation’ and are considered an art, even in routine cases. An important observation is that *identifying* analysis to answer a question requires creativity, whether recognition in the sense of “I’ve seen something like this before” or devising something original. Once analysis is identified, however, then *building* an analysis model can be mechanical, and anything mechanical can be automated to reduce time and cost. In step 4, contemporary Operations Research literature extensively studies solving analysis models for answers, and nothing is added here. Step 5 includes two feedback loops; step 5a captures that executing the process offers the opportunity to learn something and capture knowledge, and step 5b captures that improved understanding of a system may induce additional questions for further understanding. Steps 5a and 5b are developed no further; attention in this dissertation is restricted to methods and tools for steps 1, 2, and 3.

To put the process into perspective, ISO/IEC 15288 is a comprehensive standard for systems engineering processes and includes categories shown in figure 4.

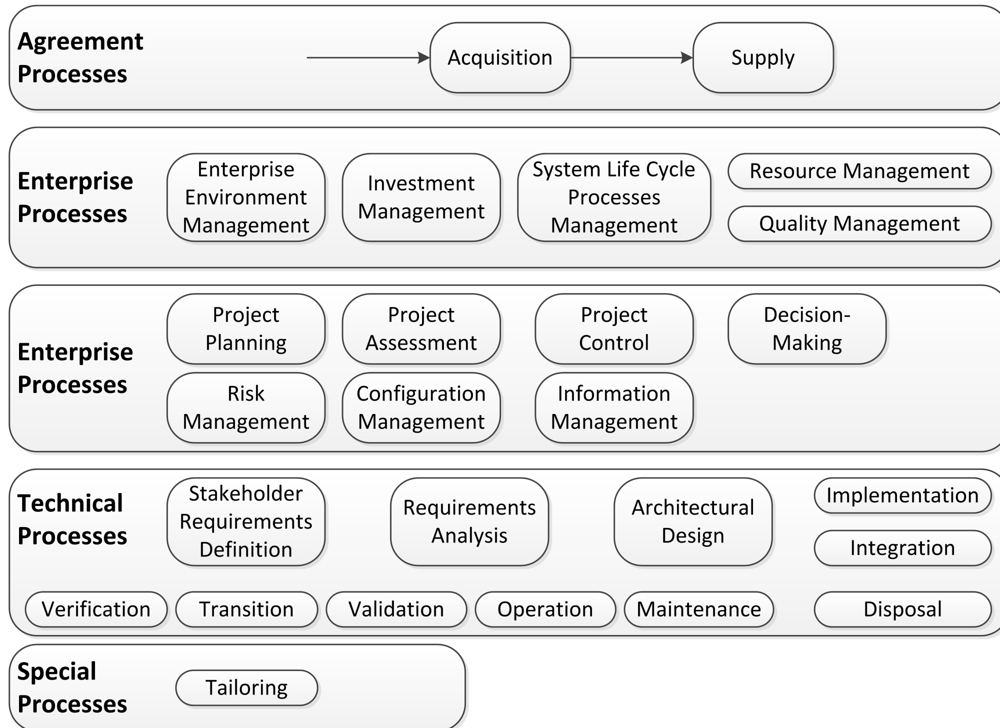


Figure 4: [Roedler, 2002]’s Illustration of ISO/IEC 15288 Process Categories.

The process in this dissertation’s methodology, illustrated in figure 3, might classify as a supporting process in the *Decision-Making* category. Note that posing a question about a

system model and then identifying and building answering analysis models is not designed to be novel, but rather to capture a process regularly executed in the status quo by any person or organization seeking to use engineering analysis to improve understanding and support decision-making. The novelty of this dissertation are methods and tools, which when applied to the process will enable the dissertation's goal.

1.2.2 An Automation Method

A method applied to the process illustrated in figure 3 is automation. Figure 5 considers what exactly in the process can be automated.

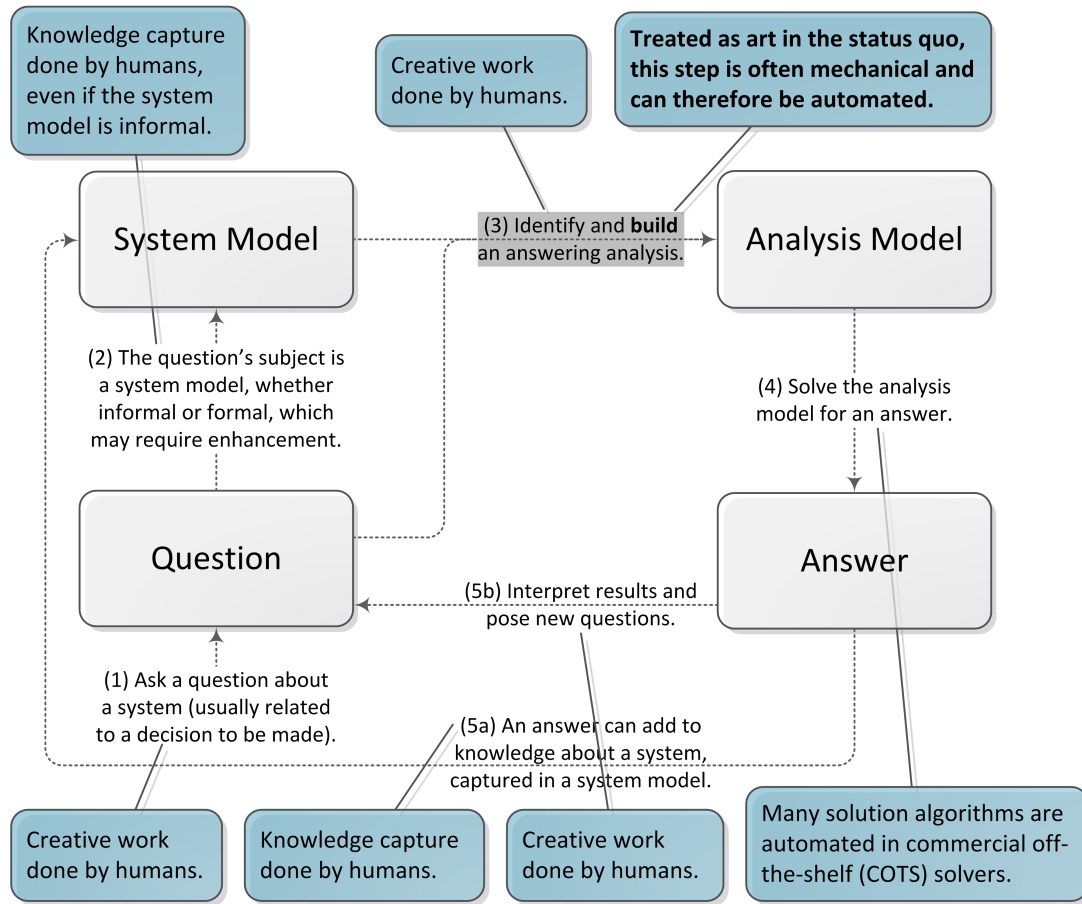


Figure 5: Where an Automation Method Might Be Applied to the Process in this Dissertation's Methodology.

The claim that analysis model-building is often mechanical and amenable to automation is explained in section 6.3; of interest here is how the automation method can help realize the dissertation's goal. Automation has good return-on-investment whenever similar tasks

are repeated over and over again. Where is repetition in building analysis models? For one answer, consider a simple discrete-event simulation analysis model in the Simio language illustrated in figure 6.

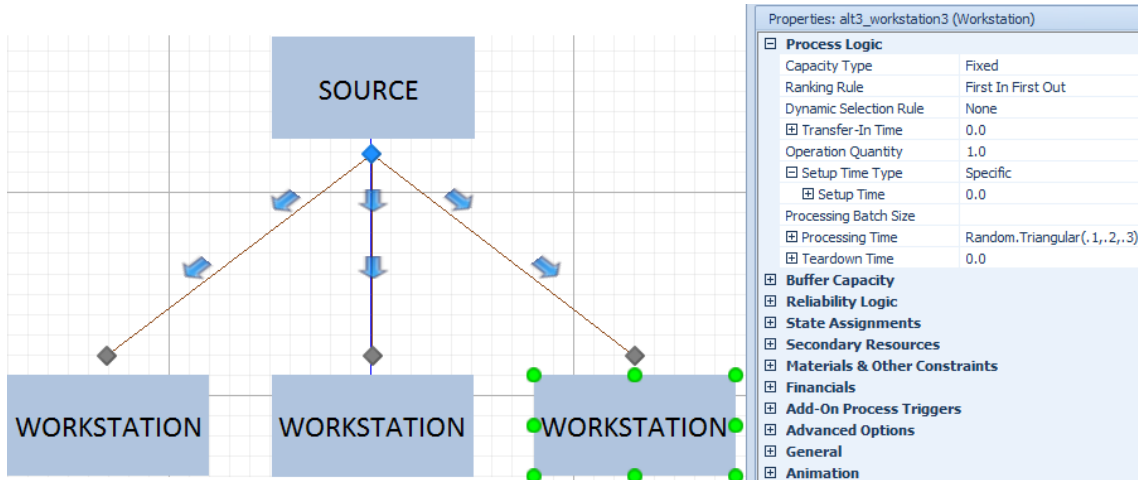


Figure 6: A Discrete-Event Simulation Analysis Model in Simio Language v5.81.

Many simulation analysis languages, Simio included, allow separating a model from any hard-coded data values, which in figure 6 include parameters for Simio's definition of a Workstation. However, process-oriented simulation models almost always hard-code a process structure, which in figure 6 includes one Source instance, three Workstation instances, and their interconnections. If the number of sources, the number of workstations, or their interconnections change, the result is a new model. Therefore, evaluating differently-structured alternatives using process-oriented simulation is a potentially high-repetition scenario in which humans may build a large number of similar analysis models, and automation may have good return-on-investment.

Beyond simulation analysis, however, many analysis modeling languages can separate a model from both hard-coded data values and also structural instances. This includes statistical regression languages, mathematical optimization languages, and in general languages allowing users to define variable-length sets or arrays of model elements. Analysis models in these languages need not be rebuilt even as the values *and also the dimensions* of data tables change. This is good for robustness but bad for model-building repetition, because a model separated from both hard-coded data values and also structural instances

need not be frequently rebuilt. Also, a bigger obstacle to model-building repetition is that any analysis model answers a specific question about a specific system, and changes to a system model's schema may make obsolete any dependent analysis model-building program. The case for automated analysis model-building needs more motivation:

- An automated builder program makes analysis models accessible to those who don't speak a particular analysis modeling language. This has value considering the number of such languages - R, SAS, JMP, and more for statistical regression, AMPL, CPLEX, Gurobi, and more for optimization, AnyLogic, Arena, SimEvents, Simio, FlexSim, Tecnomatix, and more for discrete-event simulation, and many more languages for many more types of analysis.

- An automated builder program captures the information that a certain analysis can answer a certain question about a certain system model. Creativity and innovation are always welcome for *(System Model, Question)* $\rightarrow$ *Analysis Model* relations, but there also exist routine and well-understood solutions which become more accessible with formal capture and sharing.

- In many cases, repetition is found not in the concrete but in the abstract. For example, logistics questions about moving and storing various goods in various industries may be answered with very similar network flow optimization analysis. If an automated analysis model builder can relax its dependence on a particular system model to dependence on an *abstraction* model, then its utility may multiply.

The last point is an important part of this dissertation's methodology - rather than automate analysis model-building to answer questions about concrete systems, do it to answer questions about abstractions. Automation makes sense in the presence of repetition, and while an isolated example of simulation model-building repetition based on concrete systems was identified above, much more repetition can be found by abstracting away domain-specific details and focusing on shared commonalities.

1.2.3 An Abstraction Method

> "*In the development of our understanding of complex phenomena, the most powerful tool available to the human intellect is abstraction. Abstraction arises from a recognition of similarities between certain objects, situations, or processes in the real world, and the decision to concentrate on these similarities, and to ignore for the time being the differences. As soon as we have discovered which similarities are relevant to the prediction and control of future events, we will tend to regard the similarities as fundamental and the differences as trivial. We may then be said to have developed an abstract concept to cover the set of objects or situations in question.*" [Hoare, 1972, p.83]

Developing, documenting, formulating, and solving analysis based on broadly-applicable abstractions is already practiced in the status quo. What is novel here is making the method explicit, and choosing one special abstract concept to generalize a wide range of systems and analysis of them. For discrete-event logistics systems and Operations Research analysis of them, a great amount of utility can be realized from the single abstract concept of a *token-flow network*.

A token-flow network at its core is the basic mathematical structure of a graph, modeling entities and pairwise relations between them. The graph data structure is appropriate when topology and interconnectivity of model elements is as important as the elements themselves. A graph contains one or more nodes (possibly with labels) and zero or more edges (possibly with weights). While the basic definition is simple and incontrovertible, layers on top of the basic definition are where formality and synthesis are needed. The token-flow network defined in chapters 3 and 4 is an attempt to formalize shared network semantics underlying various discrete-event logistics systems and analysis models of them, captured in the syntax of a UML profile to enable formal mapping from system to token-flow network semantics.

Making an abstraction method explicit requires enhancing the process in this dissertation's methodology with intermediate steps, shown in figure 7.

The abstraction method is crucial to the value proposition of this dissertation's

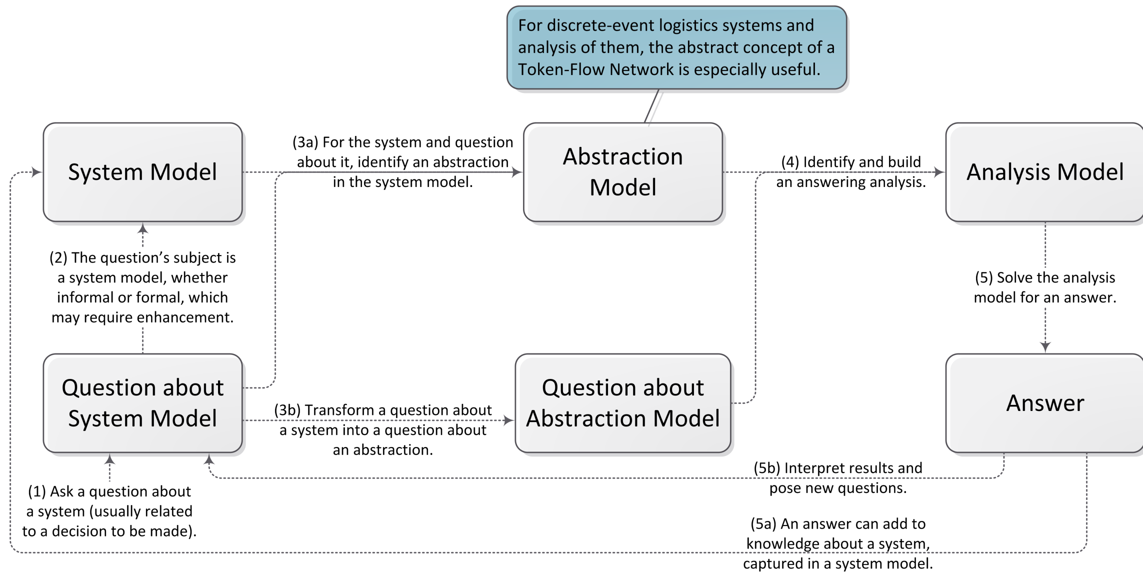


Figure 7: Adding Intermediate Steps to the Process in this Dissertation's Methodology for the Abstraction Method.

methodology - if the abstraction model is stable, then automated analysis model-building programs can be written once and broadly reused. The fragile dependence on a particular system model is not eliminated, but rather pushed upstream into an abstraction mapping from system to token-flow network semantics. Therefore, redefining mappings from system to token-flow network semantics as a system model changes must be easy, and is by using the mechanism of UML stereotype application, explained in section 6.1.

The automation and abstraction methods as discussed so far are assuming some type of knowledge capture with some degree of formality for system models, abstraction models, and even questions. A formalization method is the topic of the next section.

1.2.4 A Formalization Method

> "*Natural language is notoriously ambiguous and so if science is to progress, we need other more rule-based approaches to model building. The consensual agreement in a scientific community of signs and rules is termed* formalism. *Formalisms, such as logic-based systems with axioms and rules, tend to minimize the number of components in an attempt to reduce potential semantic ambiguity.*" [Fishwick, 2007, p.4]

Formalizing knowledge means capturing it with more precision and less ambiguity than common in natural-language descriptions. For models of an arbitrary system's structure and behavior, document-based system models of pictures and text are often too informal and require a human interpreter, and mathematical-language analysis models are often too formal and disguise any embedded system model. Capturing knowledge with a formalism like the Systems Modeling Language (SysML) [OMG SysML, 2012] instead of a natural or mathematical language may not cure all, because precisely-defined language elements does not guarantee they will be used to create an unambiguous and fidelitous model, but can be a substantial improvement. A benefit of formalization is often machine-readability, which like *formal* is an adjective of degree because even natural languages can be parsed by a computer with high accuracy using probabilistic algorithms [Klein and Manning, 2003].

A formalization method's purpose is capturing knowledge for reuse. For capturing descriptions of arbitrary systems, a preferred formalism is SysML, an extension of UML. The underlying representation is XML with a standardized schema for machine-readability, and the language allows graphical diagrams as views into a model for a human interface. The UML and SysML languages are enabling technologies, without which realizing the goals of this dissertation's methodology would be significantly more difficult.

The team performing the Europa mission concept study at NASA's Jet Propulsion Lab write that a "*common misconception is that models are not really useful until they can be subjected to quantitative analysis. This is simply not the case. Capture and description are powerful and far-reaching first steps. Just describing something in a formal*

modeling language like SysML immediately improves communications and understanding. The benefits of this would be difficult to overstate.” [Bayer et al., 2012, p.13]

1.2.5 Contributions

Figure 8 illustrates elements in this dissertation’s methodology.

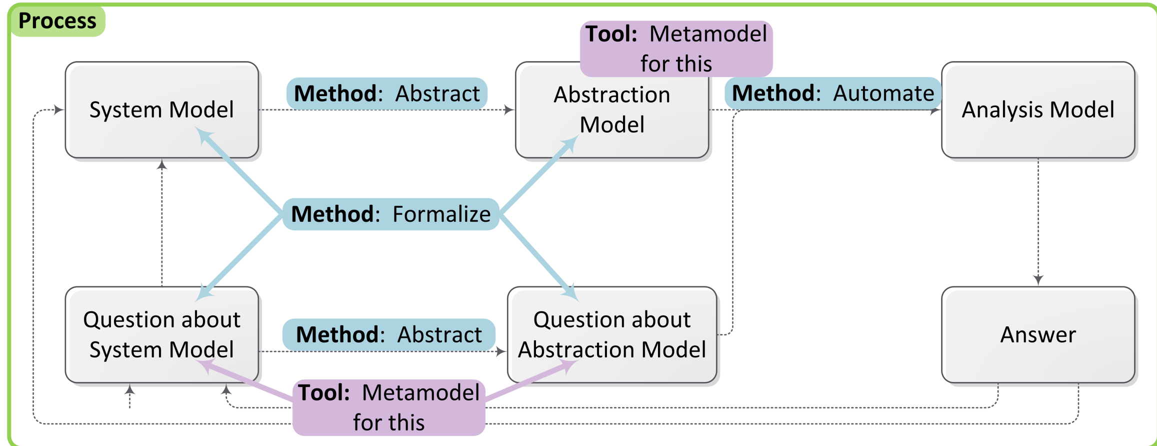


Figure 8: Process, Methods, and Tools in this Dissertation’s Methodology.

The methodology is the core contribution of this dissertation, with the goal of enabling Operations Research analysis of discrete-event logistics systems to be more widely used in a cost-effective and correct manner. The process is posing a question about a system model and then identifying and building answering analysis models. Methods include automation, abstraction, and formalization. Tools to enable the methods are where two additional contributions are found - a formal definition of the abstract concept of a token-flow network to support the abstraction method, and a formal definition of a well-formed question to support the formalization method. Investigating tools for the automation method leads to one additional contribution - addressing if and how existing model-to-model transformation tools can be re-purposed from Model-Driven Architecture of software.

1.3 Arguing a Hypothesis

A hypothesis is that this dissertation's methodology has efficacy and can enable Operations Research analysis of discrete-event logistics systems to be more widely used in a cost-effective and correct manner. One form of evidence are reports from pilot projects testing a simpler version of the methodology without the abstraction method, and these support the hypothesis. In [Batarseh et al., 2012], for an electronics assembly system and resource allocation questions about it, an experiment was conducted to automate the building of discrete-event simulation analysis models. A result estimated by the organization's analysts was a ten-fold reduction in the time required to build these analysis models. It is no accident that several such experiments concern simulation analysis models, because building same-schema but differently-structured simulation analysis models was identified in section 1.2.2 as a scenario in which this dissertation's methodology without an abstraction method can still have good return-on-investment. Similar work is documented in [Son et al., 2003], estimating a reduction from one week to less than an hour for the time required to build analysis components of a simulation-based shop floor control system. No experiments testing this dissertation's methodology with an abstraction method have been documented to the best of the author's knowledge.

A different type of argument in support of the hypothesis might derive from applying the same methods and tools to a different but similar process to illustrate their functioning and efficacy. [Law and Kelton, 2000, p.83-84] illustrate "*steps that will compose a typical, sound simulation study*" in figure 9, which is a process implicitly involving posing a question about a system model and then identifying, building, and solving an answering simulation analysis model.

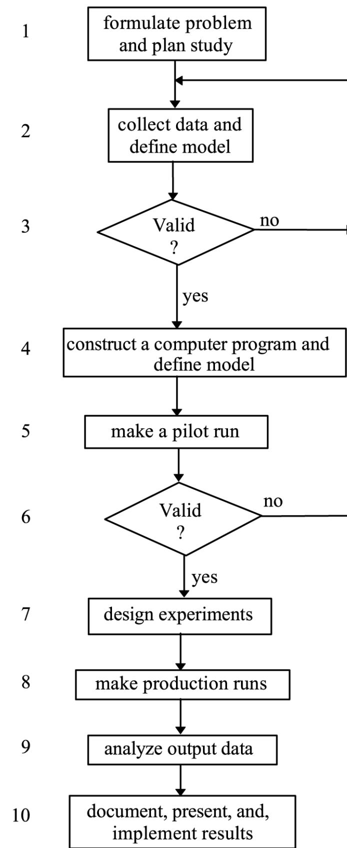


Figure 9: [Law and Kelton, 2000]'s Ten Steps of a Typical Sound Simulation Study.

In figure 9, formalization can impact the "collect data and define model" step 2. Needed information and data may already be captured in a formal system model, making this step trivial. If not - if a question targets undeveloped parts of a system model and needed information and data are missing - then step 2 requires finding, computing, or inferring needed information as in the status quo, plus augmenting the system model with that new information. Once formally captured, knowledge becomes reusable to answer any future questions, highlighting that time and cost savings from this dissertation's methodology are expected for routine questions about well-understood parts of the system.

In figure 9, formalizing questions can impact the "design experiments" step 7, and automation can impact the "construct a computer program and define model" step 4. Suppose system semantics are captured in a SysML user model, a token-flow network abstraction is defined as a UML profile, and abstracting system to network semantics is performed by stereotype application. For a token-flow network and question about it, if an

answering simulation analysis was previously identified and an automated builder program written, then step 4 will realize significant time and cost savings. If no answering analysis model was previously identified, however, then step 4 requires identifying an answering analysis' type, building the analysis model, and (if time and incentives permit) recording the model-building process into an automation program. First-case time and cost will increase in order to reduce time and cost of future repetition.

An indirect argument in support of the hypothesis is [Bertsimas and Tsitsiklis, 1997, p.266] observation that "*Network flow problems are the most frequently solved linear programming problems. They ... arise naturally in the analysis and design of communication, transportation, and logistics networks, as well as in many other contexts.*" [Collins et al., 2009, p.58] makes a similar argument by enumerating properties of a product development process and abstracting them to network statistics computable by well-known graph theoretic analyses.

Rigorously testing the hypothesis that this dissertation's methodology has efficacy would require an experiment with a statistically significant number of industrial pilot projects or implementations. Since that experiment is resource- prohibitive, it is left to happen organically and be revisited in the future to collect and analyze the results. This dissertation prepares for and tries to shape the experiment by developing required methods and tools. It is also fair to consider if and why the hypothesis may be untrue, which will be revisited in section 8.4.

1.4 Prerequisite Knowledge and Boundary

Certain knowledge is prerequisite for a reader to understand this dissertation. This includes the concept of a *model* and its various types - models describing a system's structure and behavior, models of analysis, and refinement of the latter category into models of descriptive, predictive, and control/prescriptive analysis. The most challenging prerequisite may be object-oriented thinking - understanding the relationship between a *Class* and *Object* or more generally a *Classifier* and *Instance*. Object-oriented SysML user models are used

in chapter 7's examples, and an object-oriented UML profile is used to express token-flow network semantics in chapters 3 and 4. Object-oriented thinking is implicit anytime an analysis modeling language is set-based, allowing users to use sets as model elements without knowledge of their member elements, and understanding this dissertation requires making object-oriented thinking explicit.

The scope and boundary of this dissertation's methodology are considered at length in the concluding chapter, but an abbreviated introduction is helpful here. While the methodology is desired to be fully general, the author's thinking is influenced by many years in an industrial and systems engineering department, and all use cases are derived from discrete-event logistics systems and Operations Research analysis of them. Boundary conditions follow from various components in the methodology - using the process illustrated in figures 3 and 7 or something isomorphic, using the formalization tools of SysML user models, profiles, and question semantics developed in chapter 5, using the abstraction tool of a token-flow network definition developed in chapters 3 and 4, and only building analysis models with a well-defined object-oriented metamodel.

The remainder of this dissertation is organized as follows: Chapter 2 describes prior work. Chapters 3 and 4 define structure and behavior of a token-flow network in the syntax of a UML profile. Chapter 5 documents an initial study of categories, patterns, and semantics in questions, and chapter 6 develops both the abstraction and automation methods and qualifies the methodology's boundary. Chapter 7 offers examples, and chapter 8 concludes. Appendices A and B extend chapter 2 with technical definitions of the existing token-flow network definitions of Petri Nets and UML 2.0 Activities, and appendix C extends an analysis example in chapter 7.

CHAPTER II

PRIOR WORK

Posing a question about a system model and then identifying and building answering analysis models is a process regularly executed by any person or organization seeking to improve understanding and support decision-making. This dissertation proposes applying certain methods and tools to the process to enable more widespread usage in a cost-effective and correct manner. Prior work described in this chapter shows that subsets of these methods and tools have been tried before in different contexts. Prior work with the same overall goal as this dissertation's methodology is described first. Then, prior work is outlined for the process and each method and tool.

2.1 Prior Work with the Same Goal

2.1.1 Decision-Support Systems

While history has known countless efforts to use modeling and analysis to support decision-making, attention is restricted here to contemporary efforts involving computer-based information storage systems. *Decision Support System* refers to a 1970s and onward information technology trend succeeding Management Information Systems and Electronic Data Processing. [Sprague, 1980, p.2] describes characteristics of Decision Support Systems:

- "*They tend to be aimed at the less well-structured, underspecified problems that upper level managers typically face;*
- *They attempt to combine the use of models or analytic techniques with traditional data access and retrieval functions;*
- *They specifically focus on features which make them easy to use by noncomputer people in an interactive mode; and*
- *They emphasize flexibility and adaptability to accommodate changes in the environment and the decision-making approach of the user.*"

This dissertation's methodology shares the last three characteristics and might classify as a decision-support system. It classifies even better as what Sprague calls a *DSS Generator*, which is "*a 'package' of related hardware and software which provides a set of capabilities to quickly and easily build a specific Decision Support System.*" [Sprague, 1980, p.6] An example of a DSS Generator with the same goal but narrower scope than this dissertation is [Biswas and Narahari, 2004].

2.1.2 Reuse of Concrete Analysis Models

Other methodologies with the same goal as this dissertation's involve not building analysis models from scratch but rather reusing existing ones. For statistical regression analysis, partial model reuse may be possible by copying, pasting, and modifying R programming code, keeping the model-fitting and diagnostic procedure but changing the predictors and response. For optimization analysis, partial model reuse may be possible by copying, pasting, and modifying AMPL programming code with new variables, constraints, and objective. [Robinson et al., 2004] discusses a spectrum of analysis model reuse methods from *Code scavenging* to *Function reuse* to *Component reuse* to *Full model reuse.*

Analysis models are often implemented as software. For reusing a software asset, [Larsen, 2006, p.542-543] observes:

> "*Two factors that make an asset reusable, impacting its time-to-value and thereby affecting the organization's time-to-market, are its complexity and its comprehensibility ... If an asset is truly comprehensible, offers minimal complexity, and solves a recurring problem, and if the consumer of the asset can discover it quickly, then that asset has its best chance at providing value in a timely manner.*"

Unfortunately, this observation effectively lists several obstacles to analysis models' reuse. Analysis models by their very nature can be complex, which inhibits comprehension and maintenance by authors and especially non-authors. Comprehensibility is also inhibited by a plethora of analysis modeling languages - R, SAS, JMP, and more for statistical regression, AMPL, CPLEX, Gurobi, and more for optimization, AnyLogic, Arena, SimEvents, Simio,

Tecnomatix, and more for discrete-event simulation, and many more languages for many more types of analysis. For these reasons, any methodology centered around reusing concrete and solver-ready analysis models may be fundamentally flawed and offer only marginal benefits.

Interestingly, this dissertation's methodology is all about reuse, not of concrete and solver-ready analysis models but rather of information needed to build them on-demand.

2.2 Prior Work for the Methodology's Process

The process in this dissertation's methodology is posing a question about a system model and then identifying and building answering analysis models. This process to support decision-making may be executed within a larger systems engineering process, and several of these larger processes are outlined here for context.

Per [Estefan, 2008, p.7], "*A systems engineering process is a process model that defines the primary activities that must be performed to implement systems engineering ... A variety of systems engineering process standards have been proposed by different international standards bodies, but most systems engineering process standards in use today have evolved from the early days of DoD-MIL-STD 499.*" ANSI/EIA 632, IEEE 1220-1998, and ISO/IEC 15288 are three full process standards available and contemporarily used, and categories of processes in ISO/IEC 15288 were illustrated in figure 4. A different systems engineering process using SysML to support the specification, analysis, design, and verification of systems is the Object-Oriented Systems Engineering Method (OOSEM) [Friedenthal et al., 2011]. OOSEM's system development process is illustrated in figure 10, expressed as a SysML activity.

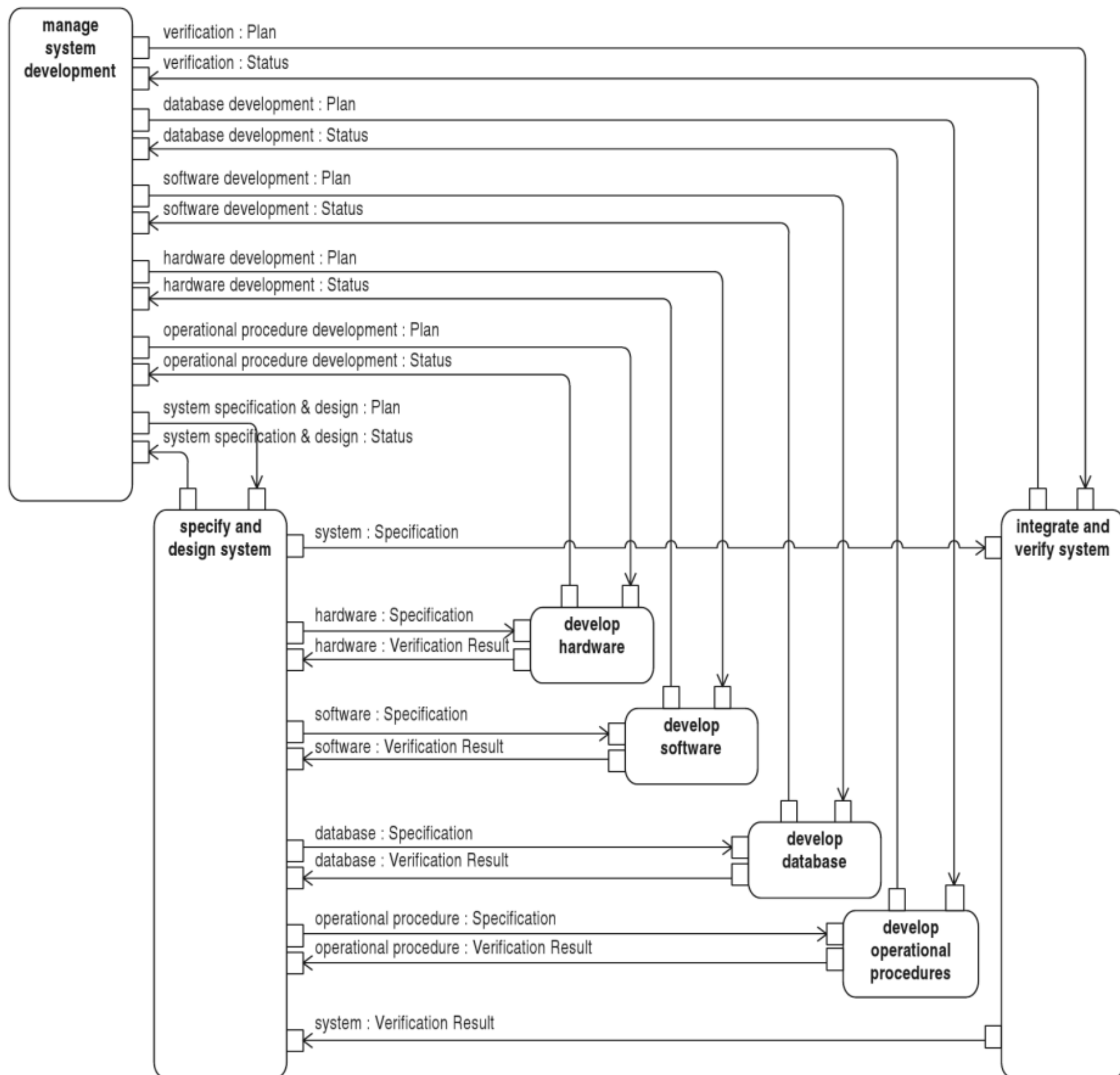


Figure 10: [Friedenthal et al., 2011]'s Illustration of the OOSEM System Development Process.

[Friedenthal et al., 2008, p.398] explain "*This process can be applied recursively to multiple levels of a system's hierarchy that is similar to a Vee development process, where the specification and design process is applied to successively lower levels of the system hierarchy down the left side of the Vee, and the integration and test process is applied to successively higher levels of the system hierarchy up the right side of the Vee. This development process is different from a typical Vee process in that it includes both management processes and technical processes at each level of the hierarchy.*" The *Specify and Design System* process in figure 10 is refined in figure 11.

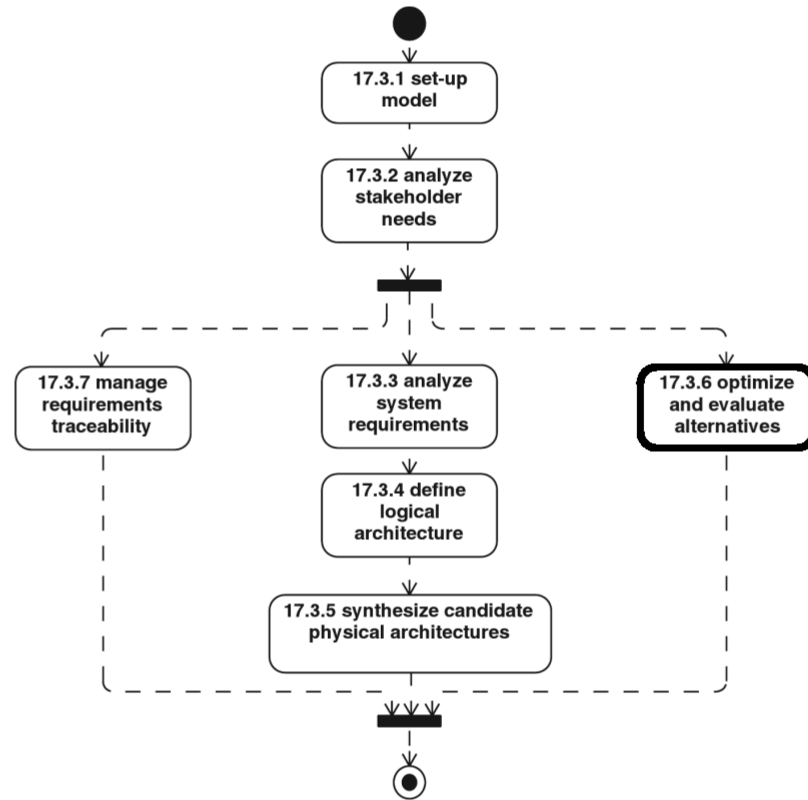


Figure 11: [Friedenthal et al., 2011]'s Refinement of the *Specify and Design System* Process in Figure 10.

The process in this dissertation's methodology can be used to support *Optimize and Evaluate Alternatives*, highlighted in figure 11.

A different class of larger process models of smaller scope than a comprehensive systems engineering process concern decision-making. A decision-making process from [Hazelrigg, 1998, p.657] is shown in figure 12, and the process in this dissertation's methodology can support several steps.

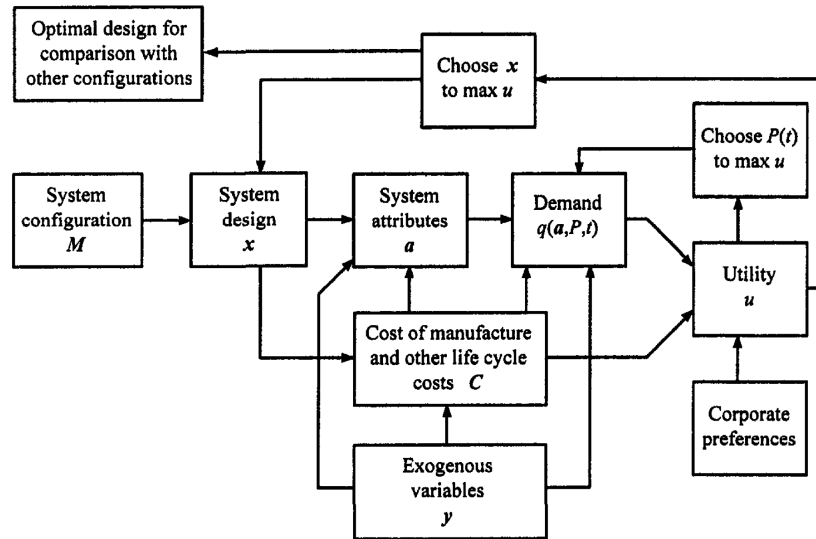


Figure 12: [Hazelrigg, 1998]'s Framework for Decision-Based Engineering Design.

2.3 Prior Work for the Methodology's Methods

In this dissertation's methodology, methods include automation, abstraction, and formalization, and prior work exists for all three. The section begins by acknowledging the heritage of software engineering; for software engineering processes, the *Model Driven Architecture* methodology applies formalization and automation methods and is described in section 2.3.1, and *Design Patterns* result from applying an abstraction method and are outlined in section 2.3.3.

2.3.1 Model-Driven Architecture of Software

The goal of *Model-Driven Architecture* (MDA) is mitigating time, cost, and complexity obstacles to make executable software more cost-accessible. It is an important predecessor to this dissertation's methodology because many processes, methods, and tools for system modeling and transformations are re-purposed from MDA. Model-Driven Architecture has origins in an OMG white paper advocating language, vendor, and middleware- independent definition of software [Soley et al., 2000]. The idea is to create a platform-independent model of software at a higher level of abstraction, which can be transformed as-needed into platform-specific and executable models. The platform-independent model aids documentation and communication, but more importantly exists as *the* definition of a

software system in a useful form. Several overviews exist, including [Kleppe et al., 2003] and [Mellor et al., 2004]. Mellor makes the motivating observation that since the earliest days of computing, designing, building, and maintaining software often takes far more time than executing it.

The main reason for creating platform-independent models at a higher level of abstraction than programming code is complexity, both of a software creation and the target platform, which make it difficult to understand structure and especially behavior. [Soley et al., 2000] advocates OMG modeling standards including the UML (Unified Modeling Language) and MOF (Meta-Object Facility); MOF is a metamodeling language which exists for the higher-level purpose of defining OMG languages including UML itself. Interesting is what the designers chose not to do - define UML using a synthesis grammar in EBNF (Extensible Backus-Naur Form). [Alanen and Porres, 2003, p.6] comment:

> "*Metamodels inherently contain more information than EBNF grammars. While an EBNF grammar is quite similar in that it can be presented as a graph of nodes and directed edges, the edges themselves do not contain as much information as properties in a metamodel. EBNF forms a tree; metamodels form graphs with special edges that can be interpreted in many ways.*"

Defining UML using MOF instead of an EBNF grammar has advantages, but impedes proofs and formal reasoning about the structure, behavior, and transformation of UML models.

Using UML's profiling mechanism, the SysML (Systems Modeling Language) was defined for general systems modeling [OMG SysML, 2012]. SysML is UML with some parts removed, some parts reused, and extensions for modeling requirements, parametric relations, and continuous-behavior activities [Friedenthal et al., 2011]. SysML represents an effort to make system models explicit, formal, and useful for the larger process of Model-Based Systems Engineering. The paradigm is the same as Model-Driven Architecture of software - create platform-independent models at a higher level of abstraction, which can be transformed as-needed into platform-specific and executable models. In this dissertation, those executable models are analysis models whose solution answers a question about a

system or its abstraction.

2.3.2 Automation

A large class of prior work for automation concerns automatically generating simulation analysis models of discrete-event logistics systems. [Oldfather et al., 1966] documents what may be the earliest simulation analysis model generator. [Yuan et al., 1993] describes a generator for SIMAN-language simulation analysis models of "discrete operational systems" modeled with an "operations network" and "operation equations", a scope similar to this dissertation's. [Son and Wysk, 2001] describe an effort to automatically build simulation analysis models in the Arena language for manufacturing systems and shop floor control questions about them. They use a network abstraction called a "message-based part state graph" to capture potential routings through a shop's resources, suggesting that the methodology may generalize to other discrete-event logistics systems and questions about resource allocation and routing. [Mueller et al., 2007] describe an effort to automatically build simulatable Petri Net models for semiconductor manufacturing systems and various questions about them, an immediate predecessor to this dissertation. Simulation analysis model generators inputting SysML-language system models are [Schonherr and Rose, 2009] and [McGinnis and Ustun, 2009]. An effort which uses a system modeling language named CONSENS instead of SysML is [Rudtsch et al., 2013].

Each of these references concern simulation model generation for a specific type of discrete-event logistics system. There also exists prior work for more arbitrary systems, such as any behavior modeled as a SysML activity [Staines, 2008], a SysML state machine [Bernardi et al., 2002], or SysML interaction in a sequence diagram [Viehl et al., 2006]. [Peak et al., 2007] and [Kim et al., 2013] describe analysis model generation based on SysML parametrics which is more general than simulation; the idea is that any parametric relation formally captured in a SysML constraint block can be the basis of an analysis model. Commercial tools to this end are ParaMagic and ModelCenter [1], enabling parametric diagrams to serve as analysis templates and SysML constraint blocks in those diagrams to

[1] *http://www.intercax.com/products/paramagic/*, viewed 27jan2014.
http://www.phoenix-int.com/software/phx-modelcenter.php, viewed 27jan2014.

be functional black boxes referencing legacy analysis models.

Important prior work also concerns automation itself - how to transform one model into another. Automated model-building programs for Model-Driven Architecture are called *model-to-model transformations* and are described in detail in section 6.3. Systems engineers have attempted to transplant software model-to-model transformation tools such as [OMG QVT, 2011] and [EMF ATL, 2014], but with difficulty. [Cleenewerck and Kurtev, 2007, p.991] suggest "*The problem of translational semantics in Model-Driven Engineering is better to be approached with a domain-specific transformation language instead of with a general purpose one*", suggesting a one-size-fits-all approach may be inappropriate. This may be the motivation of [Batarseh and McGinnis, 2012] in proposing *SysML for Arena*, a collection of Arena semantics implemented as SysML stereotypes, enabling transparently defining a model-to-model transformation while remaining in a native system modeling environment. Earlier work in the same spirit is *SysML for Modelica* [Paredis et al., 2010].

2.3.3 Abstraction

A *design pattern* captures a reusable solution to a recurring design problem in a particular context. The idea is credited to [Alexander et al., 1977, p.X] who introduced it for architectural design problems, saying "*Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.*" The concept has arguably seen its greatest utility when applied to object-oriented software design processes [Gamma et al., 1995]. The authors identify twenty-three patterns by name, problem, solution, and consequences, and partition them into *creational*, *structural*, and *behavioral* categories. Additional patterns have since been added and new categories created, both within the scope of object-oriented software design (such as *concurrency*) and outside (such as higher-level software architectures).

A software design or architecture pattern results from applying an abstraction method to software engineering processes. There have been efforts to identify patterns in systems engineering processes including [Cloutier and Verma, 2007], [Haskins, 2008], and [Pfister

et al., 2011], but the concept is less mature in this domain. Abstraction helps manage complexity in object-oriented software design, and an abstraction method is explicit in this dissertation's methodology for the same function. Semantics of a token-flow network in chapters 3 and 4 and semantics of a well-formed question in chapter 5 are structural patterns.

Prior work for an abstraction method is also found directly within object-oriented modeling languages. SysML provides several built-in language mechanisms which enable users to define multiple abstraction levels within a single model [OMG SysML, 2012]. A basic mechanism is generalization, identifying one block as a special case of another, which allows abstracting multiple blocks' commonalities into a superclass [2]. Another mechanism is nesting; a block can nest block parts, a port can nest ports, an action can nest an activity, and a state can nest a state machine. A third mechanism is stereotyping, a broad-purposed tool which can serve more functions than just abstraction.

On a deeper level, the SysML metamodel builds upon several abstract concepts. Any Block Definition Diagram can be viewed as a network of blocks and associations, and an Internal-Block Diagram can be viewed as a network of ports and connectors. A SysML Activity is defined as a token-flow network, similar to the *Process Network* definition in chapter 4. A SysML State Machine builds upon the definition of a finite-state automaton. In general, all SysML structural diagrams show views of a network or flow network, and all SysML behavioral diagrams show views of a token-flow network.

Abstraction is commonly understood as a bridge to analysis, and prior work is widespread. Network abstraction examples include [Roberts, 1978] and [Lewis, 2009]. The method itself is not novel; what is novel here is formalizing a unifying, analysis-neutral definition of the abstract concept of a token-flow network and also making explicit the process of abstracting from system to network semantics.

[2] *Generalization* is different from *Instantiation*, although both can perform the function of abstraction. A difference between *is-a* and *instance-of* relationships is that subclasses can extend a superclass' definition, whereas all instances of a class conform to the same schematic definition.

2.3.4 Formalization

Formalizing knowledge means capturing it with more precision and less ambiguity than common in natural-language descriptions. Much prior work exists for a formalization method applied to discrete-event logistics systems.

Standardization efforts exist which formalize schemas for exchanging data. An OASIS standard for the exchange of production planning and scheduling data is [OASIS PPS, 2011]. The output of the ISO TC 184/SC4 committee includes the ISO 10303 standard for machine-readable capture and sharing of product data [Pratt, 2001]. A SISO standard for the exchange of manufacturing simulation data is Core Manufacturing Simulation Data (CMSD) [SISO CMSD, 2010]. AutomationML is a standard-of-standards for exchanging data between engineering tools in a production facility [Drath et al., 2008; Faltinski et al., 2012]. These efforts standardize data schemas, but there are also efforts to standardize semantics and ontologies. An OMG standard formalizing business process semantics is BPMN (Business Process Model and Notation) [OMG BPMN, 2011]. A Supply Chain Council standard formalizing supply chain management semantics is the SCOR (Supply Chain Operations Reference) [SCC SCOR, 2012]. Proposals for formalizing manufacturing semantics are MASON (Manufacturing Semantics Ontology) [Lemaignan et al., 2006] and also [McCarthy, 1995; Molina and Bell, 1999; Cutting-Decelle et al., 2007; Guerra-Zubiaga and Young, 2008].

[Friedenthal et al., 2011, p.12] illustrates a partial classification of standardized formalisms used in systems engineering in figure 13.

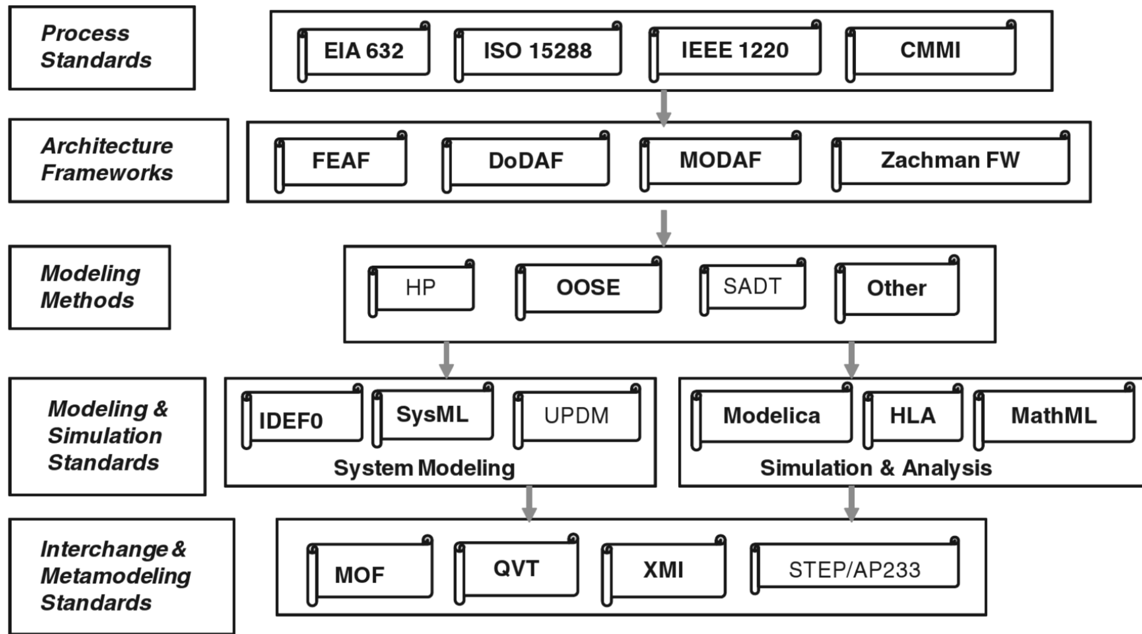


Figure 13: [Friedenthal et al., 2011]'s Partial Systems Engineering Standards Taxonomy.

A formalization method is also present in semantic web methodologies [Berners-Lee et al., 2001]. Formalization is used in this context to capture not just links to internet content but also knowledge and metadata about the content. Standards for this knowledge capture include the Resource Description Framework (RDF) and the Web Ontology Language (OWL) [3]. A connection between modeling and simulation and semantic web efforts is also made by [Miller et al., 2007, p.3-10]; the authors address and refute a claim that "*semantics is not crucial for modeling and simulation, because they are general purpose techniques that achieve their usefulness through abstraction*" in part by criticizing "*The mapping from the real world to the abstract model is largely in the mind of the simulation analyst*" and lacking any formality. [Miller et al., 2007] also collects references to numerous ontologies of scientific domains, and includes overviews of discrete-event modeling and discrete-event simulation ontologies.

[3] *http://www.w3.org/RDF/*, viewed 28jan2014.
http://www.w3.org/TR/owl2-overview/, viewed 28jan2014.

2.4 Prior Work for the Methodology's Tool of a Token-Flow Network Definition

A network is an underlying conceptual model for discrete-event logistics systems including supply chains [Bellamy and Basole, 2012], transportation systems [Magnanti and Wong, 1984], communications systems [Rogers and Kincaid, 1981], and more. It can also be argued as ubiquitous in other domains including dynamics and control of physical systems (energy flow in bond graphs [Paynter, 1961] and causal signal flow in signal flow graphs [Mason, 1953]), social relationships [Wasserman, 1994], and more.

Most relevant to this dissertation's definition are what Schruben calls *Event Graphs* [Schruben, 1983] and the SysML metamodel itself [OMG SysML, 2012], although Event Graph models lack an easy bridge backward to system semantics and SysML models lack an easy bridge forward to analysis semantics. As the network definition in chapters 3 and 4 took shape, its close similarities to the SysML metamodel were initially surprising. After so much academic training in modeling analysis, it was a surprise to discover that a network can be much more than just a bridge to analysis, it can also be a fundamental mechanism for machine-readable knowledge capture. A directed labeled graph is an underlying conceptual model for the semantic web's Resource Description Framework (RDF), and a graph is a fundamental abstract concept underlying the definition of the SysML language.

There exists prior work specific to *token-flow* networks. Two popular token-flow network definitions are Petri Nets and UML Activities, described in the next two sections.

2.4.1 Petri Nets

A Petri Net is a token-flow network with two types of nodes (places and transitions), one type of directed edge, and a single token type, although multiple token types are possible with an extension called Colored Petri Nets. A Petri Net is an elementary process behavioral model which also has straightforward representation as a (possibly infinite) state machine, where state captures the number and positions of tokens. A technical definition is in appendix A, drawing from the concise summary in [Wang, 2007].

Petri Nets' simple and precisely-defined semantics make them amenable to mathematical

analysis. If the token-flow network definition in chapters 3 and 4 can be mapped to a Colored Petri Net or another extension, this would enable using the extensive literature on Petri Net analysis to prove behavioral properties about token-flow networks abstracted from system models. Such a mapping might be possible, because the definition of a Process Network in section 4.2 has similarities to the definition of a SysML Activity whose basic elements can be mapped to a Petri Net [Huang, 2011; Storrle and Hausmann, 2005], but this is not attempted here. The object of this dissertation is to formalize a token-flow network as unifying semantics underlying discrete-event logistics system models and analysis models of them, and the low-level semantics of Petri Nets seem inadequate for this purpose. For example, an analog to a Petri Net *transition* seems unlikely to be found in the structural portion of a discrete-event logistics system model, and Petri Nets' control paradigm only allows token movements to be determined locally. Petri Nets' theoretical simplicity, while enabling rigorous mathematical analysis, can make modeling real systems' structure and control challenging.

2.4.2 UML 2.0 Activities

UML is a software modeling language, and an original purpose of activity modeling was describing object-oriented software's behavior as methods execute and exchange object pointers and control. The UML 2.0 specification released in 2003 contained a major rewriting of activities' semantics and abstract syntax; the technical definition is in appendix B, and [Bock, 2003, p.45] explains the new paradigm:

> "*UML 2 activity models follow traditional control and data flow approaches by initiating subbehaviors according to when others finish and when inputs are available. It is typical for users of control and data flow to visualize runtime effect by following lines in a diagram from earlier to later end points, and to imagine control and data moving along the lines. Consequently a token flow semantics inspired by Petri nets is most intuitive for these users, where 'token' is just a general term for control and data values. . . . UML 2 activities define a virtual machine based on routing of control and data through a graph of nodes*

connected by edges. Each node and edge defines when control and data values move through it. These token movement rules can be combined to predict the behavior of the entire graph."

The last sentence has an interesting footnote: "*It is hoped that the rules are precise enough to be translated to a formal semantics, especially to support proving properties about modeled processes. This is left for future work.*" The UML 2.4.1 specification released eight years later still claims that the intermediate level of activities "*supports modeling similar to traditional Petri nets with queuing*" [OMG UML, 2011, p.303]. [Storrle and Hausmann, 2005, p.117] attempt to map UML 2.0 Activities to Petri Nets' formal semantics, and conclude "*for basic activites, the analogy works pretty well, but for higher-level constructs, no such intuitive connection exists.*" [Schattkowsky and Forster, 2007, p.8] observe "*Although the new UML 2.0 semantics for Activities seem to be close to high-level variants of Petri Nets, there are essential differences. These differences lie mainly in the fact that unlike in a Petri Net the activation of computational steps in a UML 2.0 activity is not completely local, and that some of the model elements have quite complex semantics.*"

If the definition of a *Process Network* in section 4.2 can be mapped to a UML 2.0 Activity, then analysis model-building programs can rely on a definition in the standardized UML metamodel instead of the non-standardized definition here. This is an attractive proposition, but is not attempted here for the same reason as with Petri Nets - the object of this dissertation is formalizing an original definition of a token-flow network as unifying semantics underlying discrete-event logistics systems and analysis of them. UML 2.0 Activity semantics are process-specific and in the wrong abstract syntax. The semantics defined in chapter 4 may turn out to be isomorphic to UML 2.0 Activity semantics; the exercise is to find out.

2.5 Prior Work for the Methodology's Tool of a Question Definition

A synonym for "question" is "query", and a *query language* defines syntax and semantics for queries over database and information system content. Of all known query languages [4], only the Object Constraint Language (OCL) might function as a query language for SysML models [5].

The OCL specification introduces it as "*a formal language used to describe expressions on UML models. These expressions typically specify invariant conditions that must hold for the system being modeled or queries over objects described in a model.*" [OMG OCL, 2010, p.5] While its primary purpose is to augment a UML model with content that cannot be easily or at all expressed using the UML language, expressed in the form of constraints, OCL can also be used as a query language. Concrete syntax is defined using a context-free grammar, but regarding semantics [Kleppe et al., 1999, p.148] state "*It could be argued that OCL is a formal language, although at the time of writing no complete formal semantics exist for it.*" Formalizing OCL semantics is addressed by [Richters and Gogolla, 1998] and eventually with a chapter in the specification.

Given that query languages already exist defining syntax and semantics of a well-formed question, what does this dissertation contribute? Most contemporary query languages are paired with a particular modeling language, and can only express questions about models captured in that language. Further, a tacit restriction is that queries are only answerable by navigating through a model and retrieving recorded information. Chapter 5 does not restrict attention to questions about system models captured in any particular modeling language, nor to questions answerable by navigating through a model and retrieving recorded information. Questions may have subjects which are not an explicit system model element, and be answered by formulating and solving an analysis model. The goal here is an original development of semantics, categories, and patterns in questions which induce engineering analysis, and the results of an initial study are documented in chapter 5. Syntax

[4] Over thirty examples are collected at *http://en.wikipedia.org/wiki/Query_language*, viewed 30jan2014.

[5] The Object Query Language (OQL) might also function as a query language for SysML models because of their object-oriented nature, but the Object Data Management Group (ODMG) disbanded in 2001 and the standard has not been developed since.

is left for future work, which might involve expressing the semantics in OCL's syntax.

The following sections describe two illustrative examples of query languages - the *ProQuest ABI/INFORM Complete* query language for library databases [6], and the Structured Query Language (SQL) for relational databases [7]. Beyond query languages, there also exist other lines of prior work helpful to define semantics of a well-formed question. To identify useful categories of questions about discrete-event logistics systems, [Tako and Robinson, 2012, p.805] is a prior-work survey which diagrams much-studied topics about logistics systems and supply chains. To identify useful categories of questions about token-flow networks, taxonomy is suggested in the table of contents of a graph theory textbook such as [Diestel, 2010].

2.5.1 Literature Search for Published Documents

Searching library databases for published documents requires a query language. Such query languages are often tied to a modeling language for published documents. From the *ProQuest ABI/INFORM Complete* "Advanced Search" interface can be inferred the vendor's semantics for a published document, shown in figure 14.

[6] *http://search.proquest.com/abicomplete/advanced*, viewed 30jan2014.

[7] A seminal reference for SQL is [Chamberlin and Boyce, 1974]. The contemporary SQL standard is divided into nine parts ISO/IEC 9075-¡part¿:¡year¿

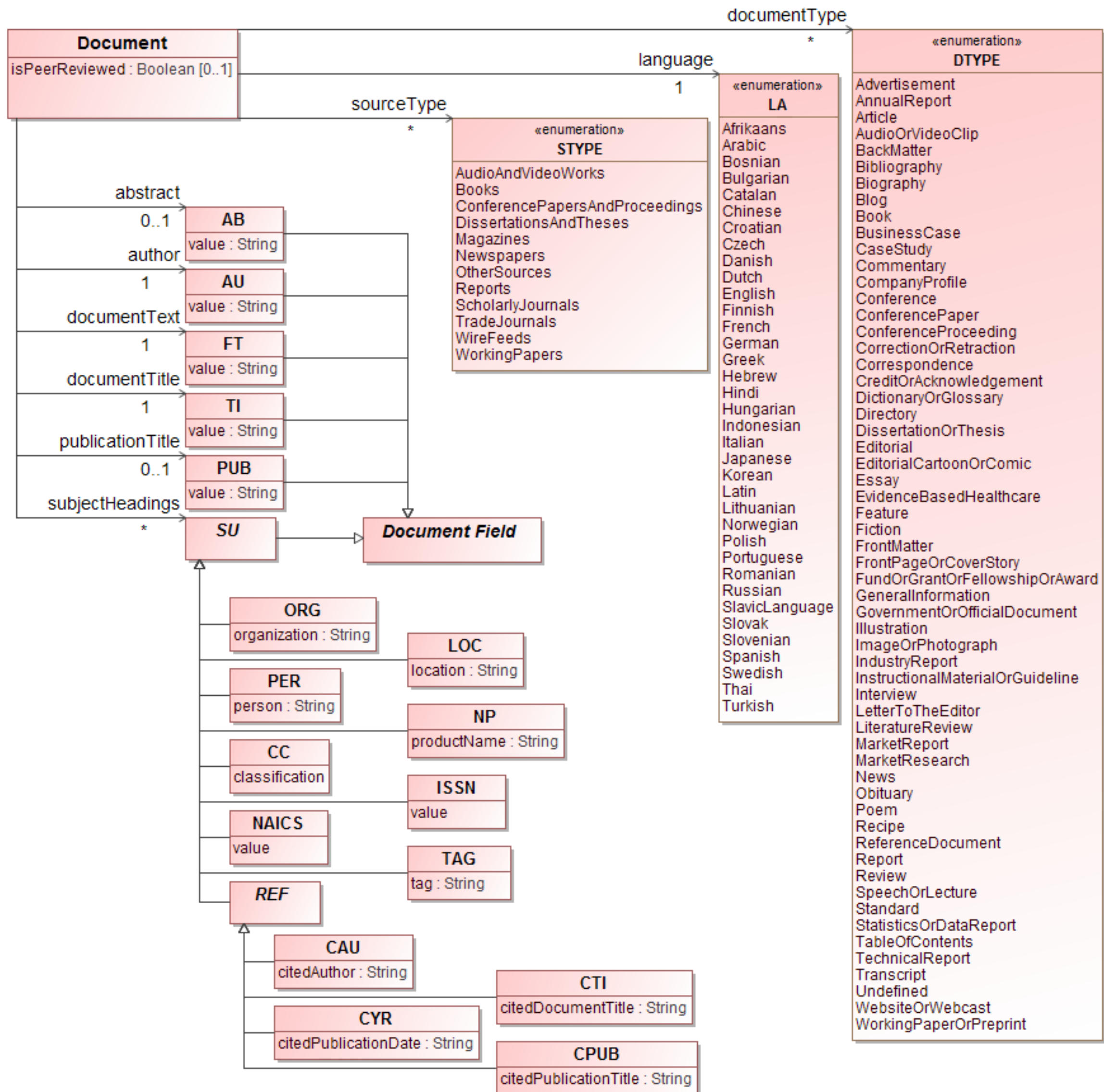


Figure 14: A Definition of a Published **Document** per *Proquest ABI/INFORM Complete*. Note that this definition is not standardized across the industry.

The vendor's user interface for a document retrieval query is shown in figure 15.

Figure 15: User Interface for Constructing a *Proquest ABI/INFORM Complete* Document Query, viewed 07feb2014. Additional search options including *Source Type*, *Document Type*, and *Language* are truncated.

From the user interface in figure 15, semantics for a document retrieval query can be inferred and are illustrated in figure 16.

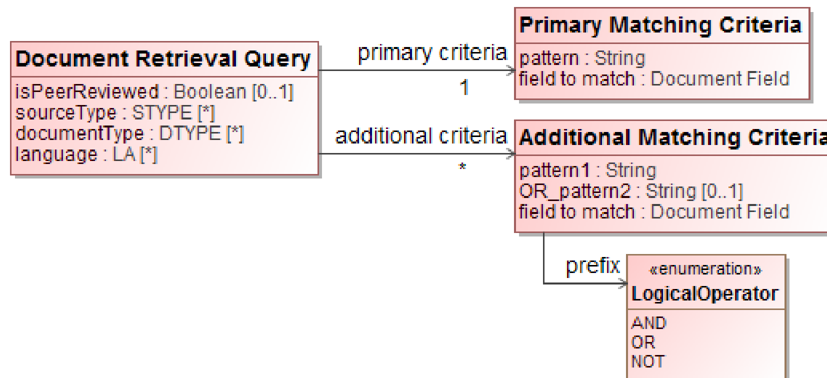


Figure 16: Schema for a Proquest ABI/INFORM Complete Document Query. Not all criteria are shown.

Questions constructed with this query language have the form "What are all published documents matching search criteria x?", where x lists at least one pattern to match in one document field and may also list source types, document types, and languages. While the *Primary Matching Criteria* is straightforward - match a certain string in a certain document field - the *Additional Matching Criteria* allows the syntax of a nested OR of two strings for user convenience.

2.5.2 Relational Database Search

Searching relational databases for relational data requires a query language. SQL (Structured Query Language) is industry-standard, although with dialects, and its syntax is formally defined using a Backus-Naur Form (BNF) grammar. Beyond queries, SQL also includes subsets of elements for user authorization (DCL, the Data Control Language), managing the schema (DDL, the Data Definition Language), and adding, updating, or deleting data (DML, the Data Manipulation Language). SQL is tied to a modeling language for relational data, illustrated in figure 17 and a metamodel for the schema of a relational database.

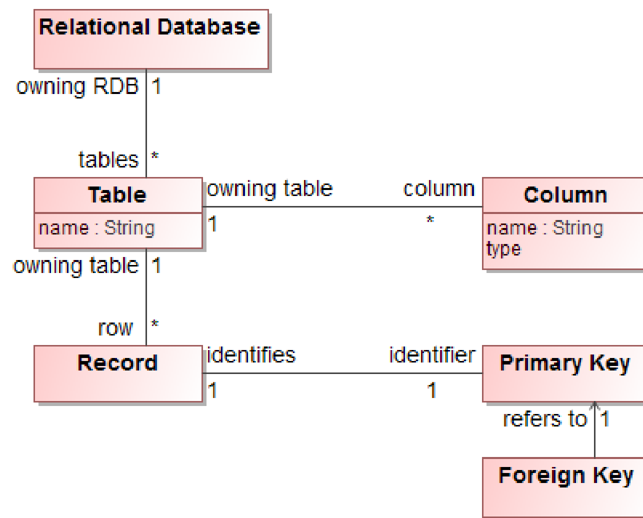


Figure 17: A Definition of a Relational Database.

The semantics in figure 17 are not complete - at the intersection of every row and column is a data value, which may have many types including character(s), boolean, integer, real, date, time, or even a foreign key referencing a primary key in another table. Tables of data values are the output of any SQL query.

The semantics of an SQL query can be inferred from the SELECT . . . FROM . . . WHERE . . . statement. The syntax of this statement, defined using a BNF grammar, is shown in figure 18 [8].

[8]The source is *http://savage.net.au/SQL/sql-99.bnf.html*, viewed 30jan2014.

```
...
<query specification>  ::=   SELECT [ <set quantifier> ] <select list> <table expression>
<select list>          ::=   <asterisk> | <select sublist> [ { <comma> <select sublist> }... ]
<select sublist>       ::=   <derived column> | <qualified asterisk>
<derived column>       ::=   <value expression> [ <as clause> ]
<as clause>            ::=   [ AS ] <column name>
...
<table expression>     ::=   <from clause> [ <where clause> ] [ <group by clause> ] [ <having clause> ]
<from clause>          ::=   FROM <table reference list>
<table reference list> ::=   <table reference> [ { <comma> <table reference> }... ]
<table reference>      ::=   <table primary> | <joined table>
<table primary>        ::=
            <table or query name> [ [ AS ] <correlation name> [ <left paren> <derived column list> <right paren> ] ]
      |       <derived table> [ AS ] <correlation name> [ <left paren> <derived column list> <right paren> ]
      |       <lateral derived table> [ AS ] <correlation name> [ <left paren> <derived column list> <right paren> ]
      |       <collection derived table> [ AS ] <correlation name> [ <left paren> <derived column list> <right paren>
      |       <only spec> [ [ AS ] <correlation name> [ <left paren> <derived column list> <right paren> ] ]
      |       <left paren> <joined table> <right paren>
...
```

Figure 18: Excerpt from a Defining BNF Grammar for SQL:1999.

The semantics of an SQL query are to choose columns from a table and then filter rows using constraints on the columns' data values. Questions constructed with this query language have the form "What is a subtable of columns from table t where the rows of that subtable satisfy criteria x?"; t can be an existing table or one created within the query. Unlike searching library databases for published documents, search criteria may be richer than string-matching because numerical data is at least ordinal and maybe even cardinal.

2.6 Summary

Posing a question about a system model and then identifying and building answering analysis models is a process regularly executed by any person or organization seeking to improve understanding and support decision-making. This dissertation proposes applying certain methods and tools to the process to enable more widespread usage in a cost-effective and correct manner. Prior work described in this chapter shows that subsets of these methods and tools have been tried before in different contexts. Both Model-Driven Architecture of software and semantic web applications are inspiring examples of what formalization, abstraction, and automation can enable, and of interest here is trying something similar for Operations Research analysis of discrete-event logistics systems.

Numerous domain-specific data schemas already exist to facilitate exchanging data, and needed next are domain-specific languages and ontologies. Examples of ontologies also exist which have enabled analysis model generation schemes with limited scope, and needed next

is a domain-agnostic and general scheme to avoid rewriting functionally equivalent analysis model-building programs for each domain. The network concept is commonly understood as a bridge to analysis, and needed next is a canonical and analysis-neutral definition in a syntax which enables formally abstracting from system to network semantics. An automation method might inherit paradigm and tools from Model-Driven Architecture, and needed next is understanding the differences between engineering analysis model-building and software object-oriented code generation to understand if this inheritance is both possible and practical.

An analysis model-building scheme which can output multiple analysis types requires indexing analyses, and a primary indexing component is the questions an analysis can answer about a token-flow network. This is distinct from prior work which avoids the issue by outputting only one type of analysis for one type of system. This motivates understanding semantics, categories, and patterns in questions which induce engineering analysis, to support questions' formalization. Query languages exist even for SysML models, but to the best of the author's knowledge have too narrow a scope by pairing with particular modeling languages and expressing only queries answerable by navigating through a model and retrieving recorded information.

For the process of posing a question about a system model and then identifying and building answering analysis models, three methods and two tools are proposed which should enable a methodology more comprehensive and with broader scope than any related prior work.

CHAPTER III

TOKEN-FLOW NETWORK: STRUCTURE

A tool to support the abstraction method is a formal definition of a token-flow network. The objective in this chapter and the next is creating an original and unifying definition of this abstract concept which generalizes many aspects of discrete-event logistics systems and underlies many analyses of them. Guidance for constructing the definition is including semantics needed to support common questions about describing, predicting, and controlling discrete-event logistics systems. The abstract syntax is a UML profile, which enables formally abstracting system semantics in SysML user models to token-flow network semantics using the mechanism of stereotype application.

3.1 Basic Structure

Basic structure shared by all network definitions should be incontrovertible - nodes connected by edges, here undirected edges. Adding node labels and edge weights can support richer questions. Node labels are sometimes called "colors", and an edge's weight can represent length, importance, viscosity, electrical resistance, ambient temperature, layers of management, density of obstacles, and more. Basic network structural semantics are shown in figure 19.

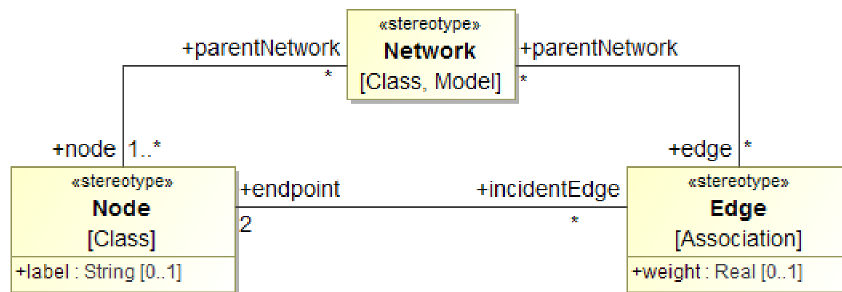


Figure 19: Semantics for Basic Structure.

A *Network* contains one or more *Nodes* and zero or more (undirected) *Edges*. Any node or edge can belong to multiple networks, and if one does then those networks overlap.

Explicitly identifying a network is optional, because one can be inferred from a collection of nodes and edges. An example of a token-flow network instance which conforms to basic structural semantics is shown in figure 20.

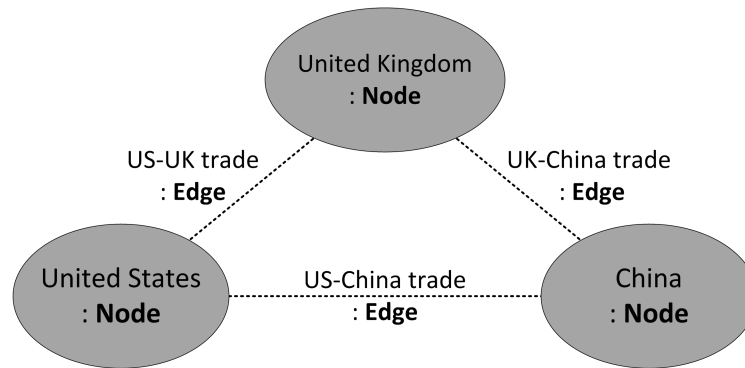


Figure 20: An Example of a Network Instance Conforming to Basic Structural Semantics.

Basic structural semantics can support definitions of all the following properties, any of which may be the subject of a question about a token-flow network:

Network Statistics:

- **Order**: The number of nodes.
- **Size**: The number of edges.
- **Density**: The ratio of the number of actual edges to the number of possible edges.
- **Clique Number**: Order of the largest complete subgraph.
- **Diameter** (requires edge lengths): The longest shortest path between all pairs of nodes.

Network Boolean Statistics:

- **Connected**: True if there exists a path between every pair of nodes.
- **Acyclic**: True if there exist no cycles, meaning for any node there does not exist a sequence of edges which revisits that node.
- **Bipartite**: True if nodes can be partitioned into two sets, with no two nodes in either set linked by an edge.

Node Statistics:

- **Degree**: The number of incident edges.
- **Local Clustering Coefficient**: Quantifies how close a node's neighbors are to being a complete graph.
- Centrality Measures: Quantifies the relative importance of a node. **Degree Centrality** counts incoming and outgoing edges. **Katz Centrality** also considers extended neighbors with an attenuating multiplication factor for each step removed. **Closeness Centrality** (requiring edge lengths) is the inverse of the sum of distances to all other nodes, such that less aggregate distance means more closeness. **Betweenness Centrality** (requiring edge lengths) counts how many times a node appears in the shortest path between all other pairs of nodes. **Eigenvector Centrality** (of which Google's PageRank is a variant) computes a measure relative to other nodes' measures, where connections to higher-measure nodes contribute more.

Edge Statistics:

- Centrality Measures: Quantifies the relative importance of an edge. **Betweenness Centrality** (requiring edge lengths) counts how many times an edge appears in shortest paths between all pairs of nodes.

Subsets of Network Elements:

- **Walk**: A sequence of edges which connect a sequence of nodes.
- **Path**: A walk in which no node is revisited (including the endpoints).
- **Coloring**: A labeling of nodes or edges (partitioning them into subsets) subject to constraints. A node coloring assigns labels such that no adjacent nodes share the same label. An edge coloring assigns labels such that no adjacent edges share the same label.
- **Covering**: A node covering is a subset of nodes collectively incident to every edge in the network. An edge covering is a subset of edges collectively incident to every node in the network.

All of these properties can optionally be recorded as Network, Node, and Edge properties in figure 19. Statistics might be recorded as real-, integer-, or boolean-valued properties, and subsets of network elements might be recorded as properties typed by network elements. A complication is when a statistic is defined in terms of input parameters and the statistic's

value is only meaningful given the parameters' values, for example the node statistic "Katz Centrality" defined in terms of an attenuation factor. This scenario can be accommodated with a custom data type recording both the parameter's value and the statistic's dependent value.

3.2 Tokens

Discrete-event logistics systems involve the movement of goods, services, information, payments, and other commodities or entities. Therefore, any useful abstraction must incorporate these semantics. *Flow* will be introduced in the next section, and in preparation, this section defines *Token* as an abstract semantic for any entity which can move in the network. The abstraction of a token is quite general, appearing in the engineering contexts of Dataflow Graphs, Petri Nets, and UML Activities, in the philosophical context of type-token distinction [1], and in software compilers as any atomic parse element. Semantics for tokens are shown in figure 21.

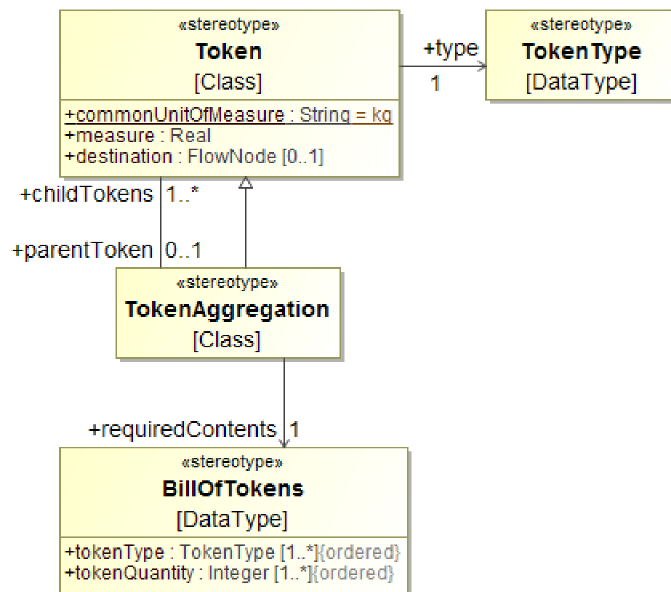


Figure 21: Semantics for Tokens. An underlined property is static, meaning the same value is shared by all Token instances.

[1]Explained at *http://plato.stanford.edu/entries/types-tokens/*, viewed 15march2014.

Any token has a static property *commonUnitOfMeasure*, which is a unit of measurement shared by all tokens of any type, and can be useful to express capacity constraints on resources. A token may also be earmarked for a particular destination, a property which may be dynamically updated, although how a token's *destination* is updated and by whom is the subject of *control* and will be revisited in section 4.5. There exists an explicit model element for *TokenType* in addition to tokens themselves; in manufacturing the acronym *SKU* (Stock Keeping Unit) can be as useful as the word *part*. A *TokenAggregation* captures a recurring group of tokens and is also a token; a manufacturing example is an assembly of parts which is itself a part. A TokenAggregation's contents are recorded in a *Bill Of Tokens*, which in other contexts is called a "Bill of Materials" or a "Packing List".

3.3 Flow

Flow is an abstract semantic to describe movement of goods, services, information, payments, and more in a discrete-event logistics system; the word enables describing token movements in the abstract and aggregate. Flow can be the subject of many questions, but since semantics for time and flow dynamics are not introduced until chapter 4, this section's semantics can only support questions whose subjects are bulk flow amounts or rates. Flow semantics are shown in figure 22.

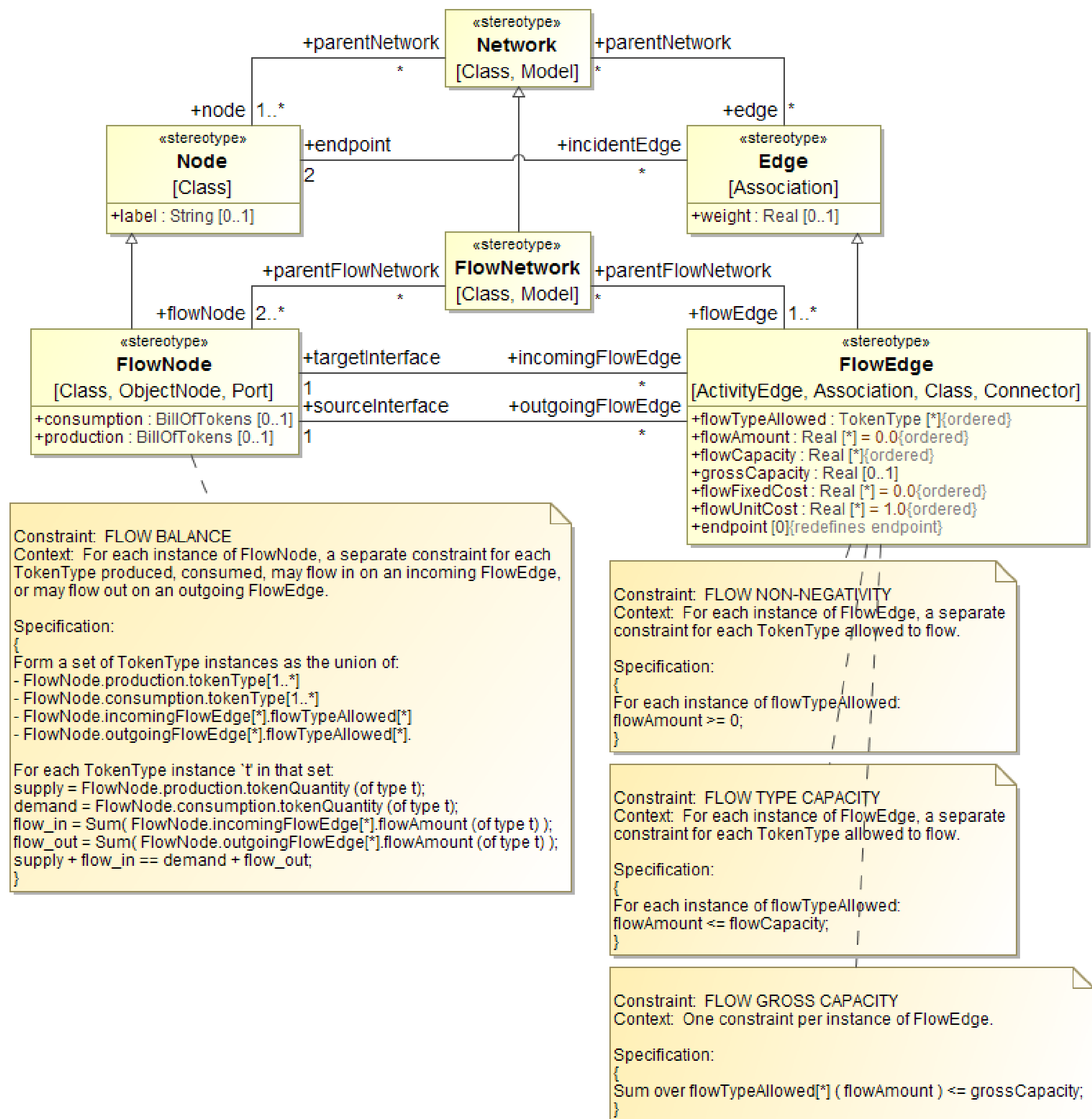


Figure 22: Semantics for Flow. Standard network flow constraints are modeled informally in textual notes; an implementation version of the profile should make them more formal, perhaps using the Object Constraint Language [OMG OCL, 2010].

A *FlowNode* is a special type of Node which supports incoming and outgoing flow. It has properties for *production* and *consumption* of tokens (also called *supply* and *demand*) to make explicit entering and exiting token flows. FlowNodes are connected by a special type of Edge called *FlowEdge*, with properties for allowable token types, capacities, and costs. Semantics in figure 22 enable creating two distinct types of networks - a Network of Nodes and Edges modeling objects and relationships, and a FlowNetwork of FlowNodes

and FlowEdges modeling flow movement, consumption, and production. An example of a FlowNetwork instance is shown in figure 23.

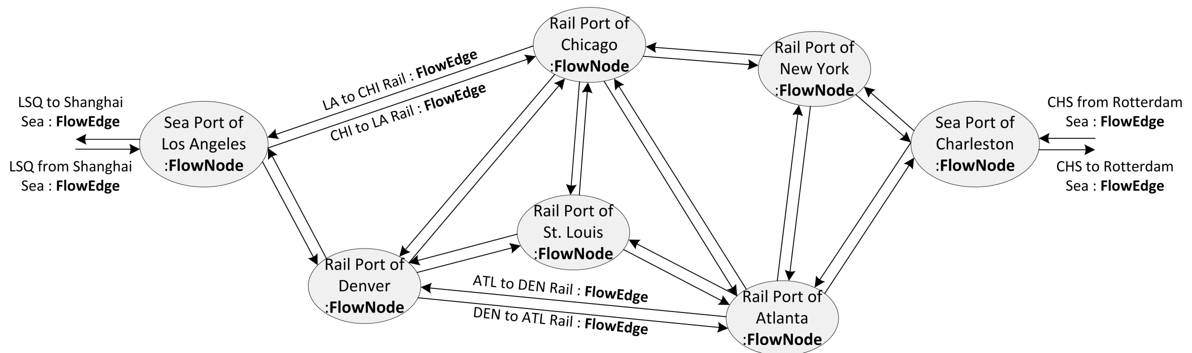


Figure 23: An Example of a FlowNetwork Instance Conforming to Flow Semantics.

More clarification of flow semantics:

- A FlowEdge is directed from a source to a target FlowNode, allowing non-negative one-way flow. FlowEdge overrides its *endpoint* property inherited from Edge because an Edge is undirected but a FlowEdge is directed. As with Node and Edge, any FlowNode or FlowEdge may belong to multiple FlowNetworks, in which case those networks overlap.

- A FlowEdge can carry multiple types of flow; an analogy is multiple small tubes inside a single large tube. For this reason *flowAmount*, *flowCapacity*, *flowFixedCost*, and *flowUnitCost* properties each have [*] multiplicity and are ordered - each is a vector over all allowed TokenTypes. FlowEdges must respect capacity constraints for each flow type, plus a gross capacity over all types. No semantics exist for incurring flow losses on FlowEdges, although the definition can be extended if needed to support questions whose answers are judged valuable.

- Self-supply at any FlowNode may occur, but only externally through a self-FlowEdge (with the same source and target). It may also seem superfluous to include both consumption and production when consumption can be inferred as negative production, but this is done because a stereotype's properties should align with properties of the block to which it is applied, and it is unreasonable to expect a business

enterprise to define its outputs as negative inputs. Production and consumption quantities have no sign restrictions, however, so only one of the two properties is strictly necessary.

- Any FlowNode has a flow balance constraint for each TokenType, with the form:

$$flow_{in} + production = flow_{out} + consumption$$

  Tokens can be created and destroyed; this constraint only requires proper accounting. Satisfying these constraints is what causes flow to move at all in network flow optimization analysis.

- There are also SysML model constraints, for example enforcing consistency among multiplicities. An example is that FlowEdge's properties *flowAmount*, *flowCapacity*, *flowFixedCost*, and *flowUnitCost* should have the same multiplicity as *flowTypesAllowed*.

- The *FlowNode* stereotype extends three metaclasses - *Class*, *ObjectNode*, and *Port*. However, only one of the three is fully functional because the UML language definition only allows Class (a subclass of *Classifier*) but not ObjectNode nor Port to have properties. This is not an obstacle upon tunneling through a Port to its typing block for properties, and tunneling through an ObjectNode (e.g. a *Pin* in an Activity Diagram) to its typing block for properties.

3.4 Interfaces

While semantics in figure 22 support two distinct types of networks, use cases integrating the two are easily imaginable. For example, a parent organization (which abstracts to a Node) may own and operate manufacturing facilities (which abstract to FlowNodes), and contractual relationships and international treaties (which abstract to Edges) may enable supply channels (which abstract to FlowEdges). Figure 24 shows one way to integrate the two types of networks.

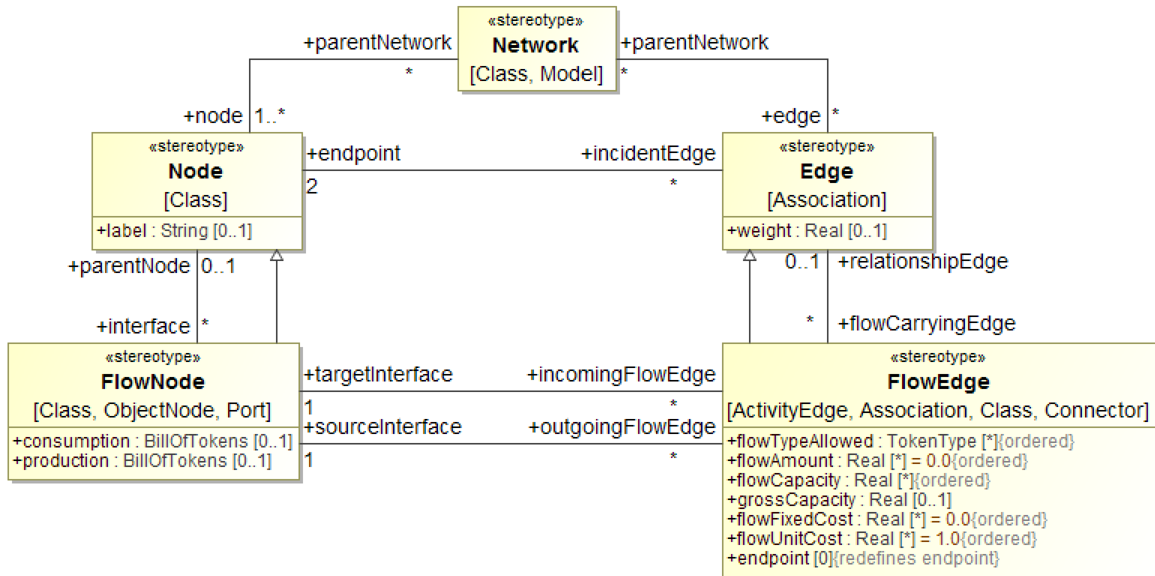


Figure 24: Semantics for Interfaces.

The only changes from figure 22 to figure 24 are two added associations. Node has a new association with FlowNode; while FlowNodes can exist independently in a FlowNetwork, the new association allows them to also be flow interfaces to Nodes. Edge has a new association with FlowEdge; while FlowEdges can exist independently in a FlowNetwork, they can now also associate with Edges modeling other types of relationships. An example of integrated Network and FlowNetwork instances are in figure 25.

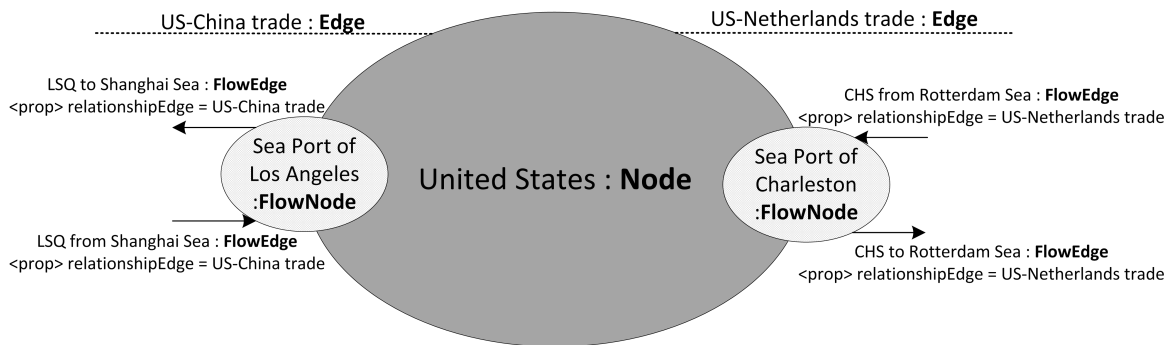


Figure 25: An Example of Integrated Network and FlowNetwork Instances using Interface Semantics.

Figure 25 shows an excerpt from figure 20 in which specific coastal ports of the United States node are visible. Container ships can only load and unload at a port-of-entry, which is a compelling use case for FlowNodes interfaces to Nodes. FlowEdges which correspond

to sea shipping lanes are associated with international trade relationships.

A complication introduced by interface semantics is that the association between Node and FlowNode, in combination with the existing generalization relationship, induces recursion - a FlowNode may have FlowNode interfaces of its own. This can be useful because further refining figure 25 may reveal that the Port of Charleston has five terminals, ports within the port, and a container ship can dock at only one. Zooming in and out to reveal different amounts of detail is enabled by levels of abstraction, the subject of the next section.

3.5 Levels of Abstraction

Levels of abstraction is a useful concept for organizing models. Levels of abstraction in a token-flow network are enabled with one simple addition to figure 24 shown in figure 26, an association allowing a Node to nest a Network.

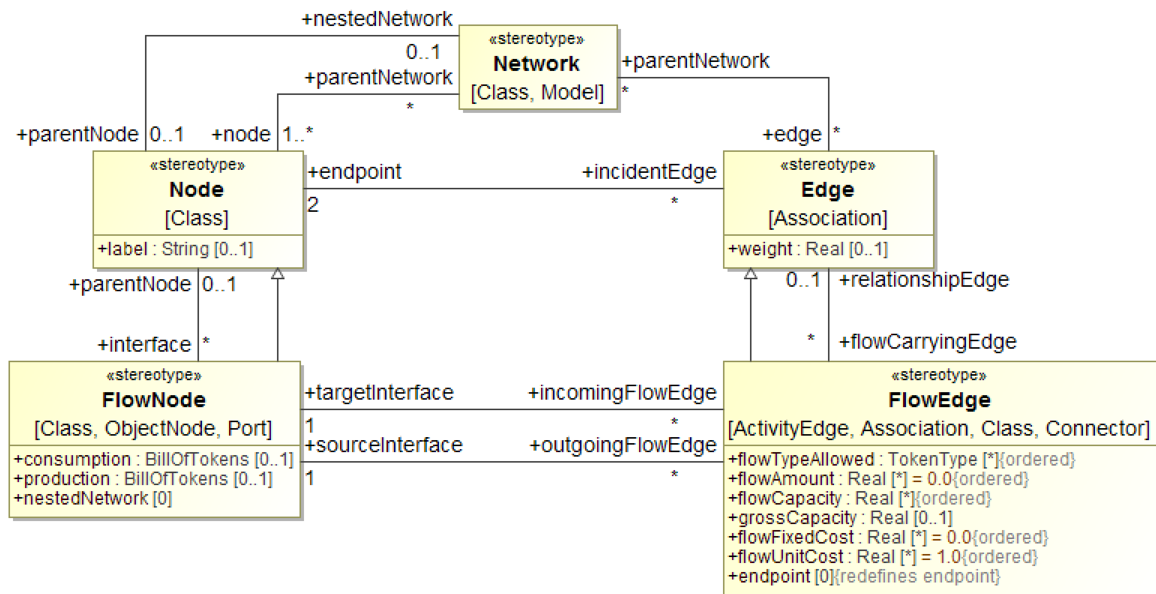


Figure 26: Semantics for Levels of Abstraction.

Figures 27 and 28 show examples of networks nested within a node.

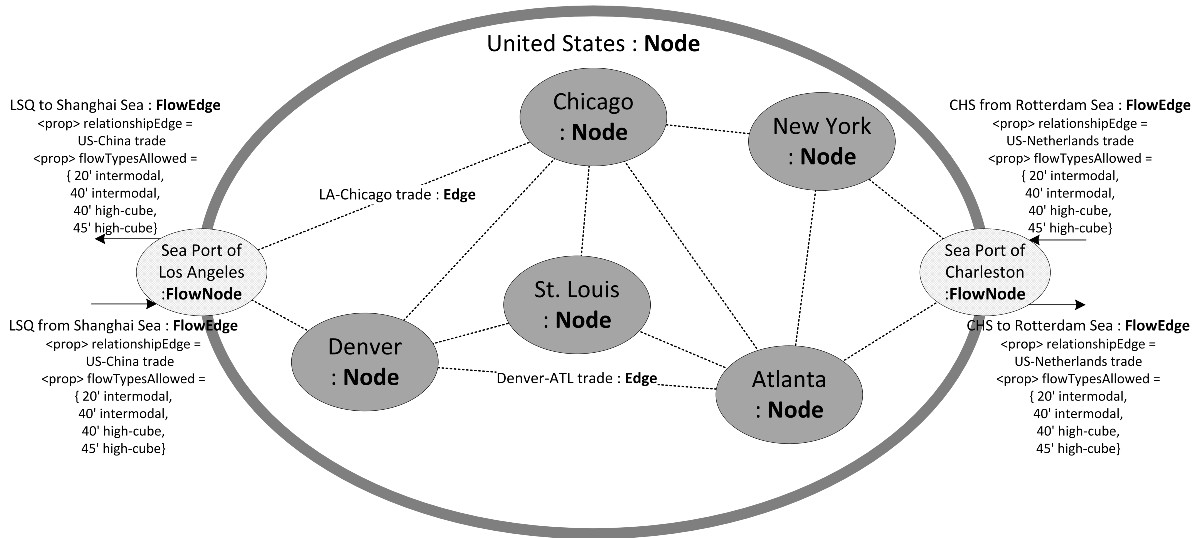


Figure 27: An Example of a Nested Network Conforming to Level Of Abstraction Semantics.

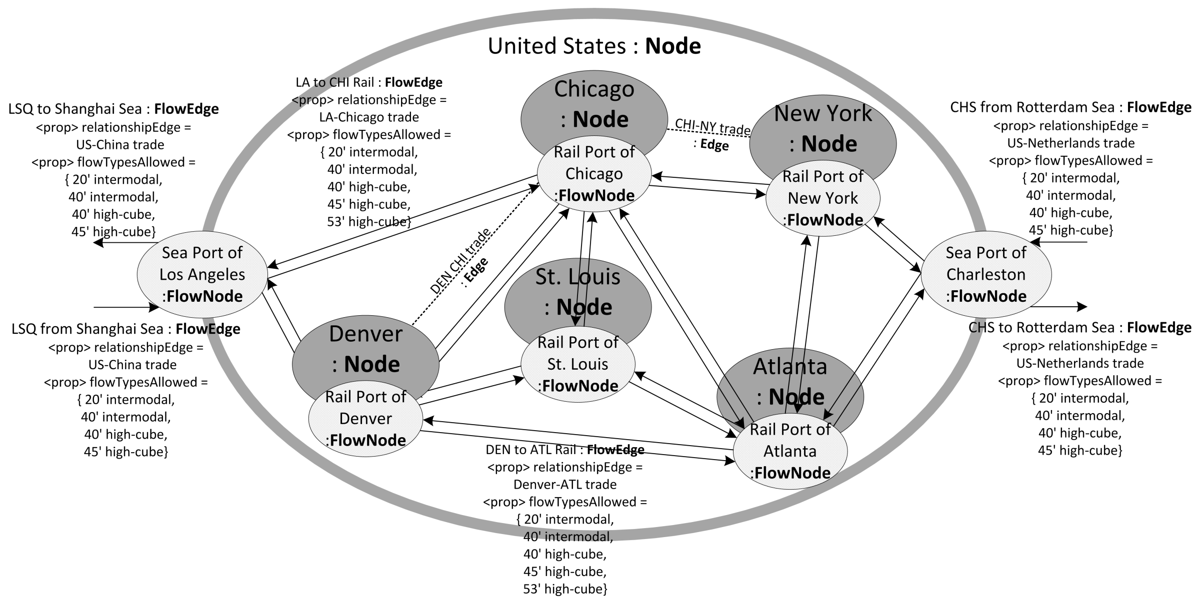


Figure 28: An Example of a Nested Network and Integrated FlowNetwork Instance Conforming to Level Of Abstraction Semantics.

The association enabling nested networks is recursive - for example, New York city in figures 27 and 28 may be refined into a nested network of five boroughs. Semantics are needed allowing flow to move between different abstraction levels, for example trade flows not just to a United States' coastal port but to a specific city, and the key enabler

is a FlowNode interface of a Node. A Node's FlowNode interface spans the boundary between a node and its nested network, and through these interfaces is the only defined way that flow can move among multiple abstraction levels. Because a FlowNode interface may span different abstraction levels, its *parentFlowNetwork* property may reflect membership in multiple FlowNetworks.

Semantics are also needed to ensure that *production* and *consumption* values are consistent between a node's FlowNode interfaces and the node's nested network. Figure 29 shows a simplified version of figure 28.

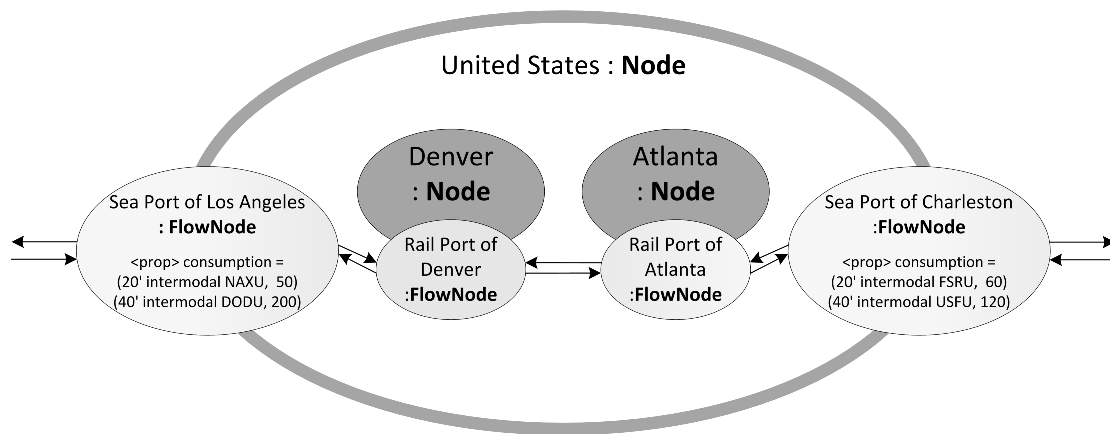


Figure 29: A Simplified Version of Figure 28, Showing Consumption of Tokens at a Node's FlowNode Interfaces.

In figure 29, the United States node can be refined into a nested network. At a higher abstraction level in which the nested network is hidden, all production and consumption of tokens happens at the FlowNode interfaces Sea Port of Los Angeles and Sea Port of Charleston. However, at a lower abstraction level the United States node is effectively replaced by its nested network. The FlowNode interfaces Sea Port of Los Angeles and Sea Port of Charleston persist and are the link between the nested network and the outside world, but now production and consumption of tokens happens in Denver and Atlanta rather than at coastal ports. Therefore, production and consumption at the Sea Port of Los Angeles and Sea Port of Charleston should become zero and *allocated* among the nested network's FlowNodes, accurately reflecting the coastal ports' function as effective flow-through ports of entry. This is shown in figure 30.

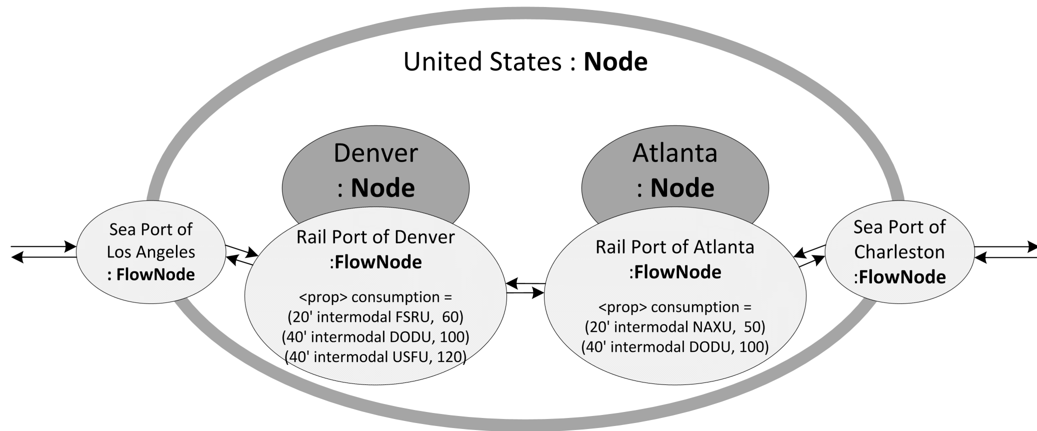


Figure 30: A Simplified Version of Figure 28, Showing Consumption of Tokens at a Node's FlowNode Interfaces*Allocated* to FlowNodes in a Nested Network.

These semantics apply recursively, for example if the city of Denver contains a nested network. Then production and consumption is allocated among that nested network's FlowNodes, and Rail Port of Denver becomes a flow-through port of entry. Ensuring production and consumption values remain consistent can be formally modeled with constraints. For a nested network, the sum of *consumption* over all nested nodes' FlowNode interfaces must equal the aggregate consumption of the parent node's FlowNode interfaces, for each TokenType, and the same for *production*. Constraining them to be equal allows placing all production and consumption of tokens at the interfaces when the nested network is hidden, and distributing it among nested nodes' FlowNode interfaces when the nested network is visible.

. . . . . .

While a Node can nest a Network, no use cases are considered in which it is valuable for a FlowNode to inherit the same ability. What a FlowNode does inherit is the ability to nest FlowNode interfaces , with an example shown in figure 31 [2].

[2]Nesting interfaces was a key enhancement in SysML version 1.3 [OMG SysML, 2012].

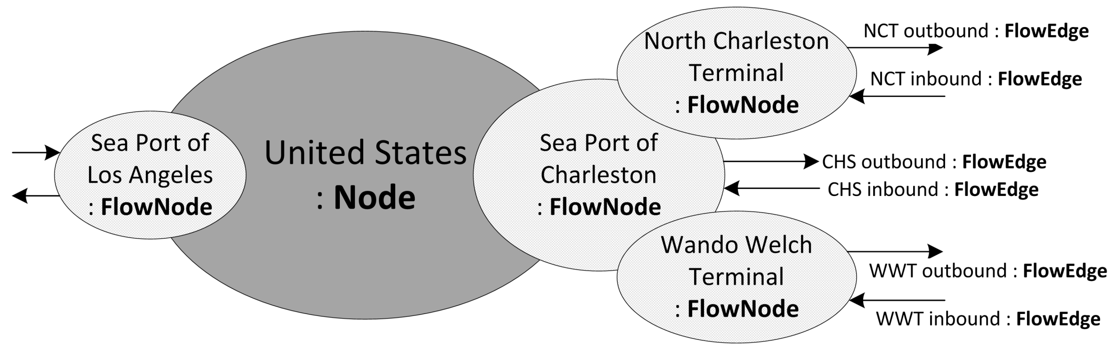


Figure 31: Incorrect Example of a FlowNode with FlowNode Interfaces.

Nested interfaces can be useful because the Port of Charleston has five terminals, ports within the port, and a container ship can dock at only one. If a FlowNode interface nests FlowNode interfaces, the parent (Sea Port of Charleston) should assume the *parentNode* role in the association between Node and FlowNode, FlowEdges incident to the parent should assume the *relationshipEdge* role in the association between Edge and FlowEdge (despite their directionality), and nested FlowNode interfaces (North Charleston Terminal, Wando Welch Terminal) should define a new set of incident FlowEdges. What should be avoided is exactly what is shown in figure 31 - a parent FlowNode interface and its nested FlowNode interfaces both having incident FlowEdges. Figure 32 resolves the issue by changing stereotypes.

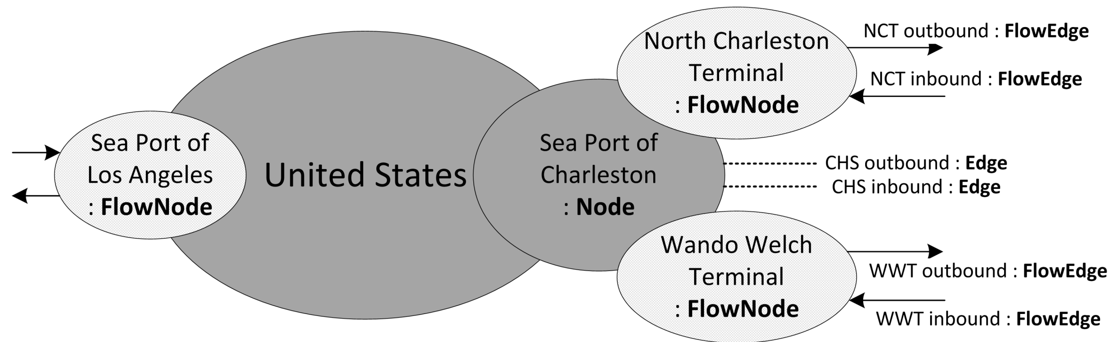


Figure 32: Corrected Example of a FlowNode with FlowNode Interfaces.

If both the United States and Sea Port of Charleston are typed by Node, the result is an ill-formed token-flow network because a Node cannot have Node interfaces, only FlowNode interfaces. This is resolved by selective focus with stereotype application - identify the Sea Port of Charleston as the parent Node and North Charleston and Wando Welch Terminals

as FlowNode interfaces. When a FlowNode nests FlowNode interfaces, *production* and *consumption* at the parent FlowNode (Sea Port of Charleston) should be allocated to its own interfaces (North Charleston Terminal and Wando Welch Terminal), such that the sums over the nested FlowNodes' *production* and *consumption* values equal the parent FlowNode's values. This example also illustrates the mechanism for formally changing abstraction levels using SysML stereotype application - clear out existing stereotypes and reapply at the desired level of abstraction. In the presence of recursive nesting, absent any validation tools, users should take care that a formally-identified token-flow network conforms to the definition.

3.6 Analysis Semantics: Variables, Constraints, Objectives, and Observations

The purpose of this section is ensuring sufficient semantics for answering questions about flow networks using optimization analysis. This is a narrow scope, but network flow optimization analysis - choosing edge flow amounts to satisfy constraints and optimize a performance measure - is widely studied and practiced within Industrial Engineering schools.

Optimization analysis involves variables, constraints, and an objective. Analysis semantics do not necessarily belong in the token-flow network definition, but do require semantics in that definition from which they can be synthesized. An optimization *variable* carries assumptions that its value can change, be controlled, and set to a desired value[3]. A variable can follow from any numerical property such as FlowEdge's *flowAmount* for network flow optimization. For identifying variables, the semantic shown in figure 33 may be helpful.

[3]If not true, optimization analysis can still be used to answer questions about describing limits on performance or optimal scenarios, even if they cannot be realized.

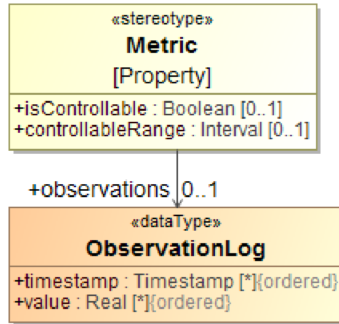


Figure 33: Semantics for Metrics.

Metric adds optional semantics to make a property's controllability explicit instead of assumed. *Metric* also adds optional semantics for recording observations of a property's value, although a time series requires the concept of *time* not introduced until section 4.1. This is not directly useful for questions answered by optimization analysis, but potentially very useful for questions answered using statistical analysis. Note that tool support for *Metric*'s properties might be lacking, but their definition is consistent with the UML metamodel - the metaclass *Property* has a property *qualifier : Property [0..*]* where these semantics might be recorded.

For the analysis semantics *Constraint* and *Objective*, their definitions often include mathematical expressions, but this is not done here because there already exist numerous languages defining mathematical expressions. Standard network flow constraints were defined in figure 22 (flow balance at FlowNodes, flow non-negativity and capacity on FlowEdges), and to that collection can be added a general flow lower bound constraint shown in figure 34.

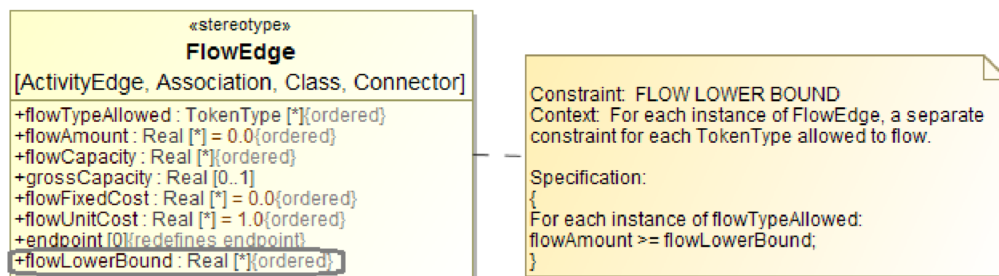


Figure 34: Token-Flow Network Structural Definition: Additional Flow Constraints.

Beyond constraints in figures 22 and 34, additional constraints must be defined in a question. Performance measures from which an objective may follow are also left to be defined in a question. It is possible to include a set of standard performance measures in the flow network definition, a common one being sum of cost-weighted edge flows, but this is left for future work. Inspiration for flow network performance measures might follow from abstracting domain-specific standards such as the SCOR model for supply chains [SCC SCOR, 2012].

3.7 Summary

Flow was introduced in section 3.3, but to this point no semantics are defined for its dynamics, meaning flow is just another structural attribute. This is enough to answer a class of questions about flow networks using network flow optimization analysis - any question answerable by choosing edge flow amounts or rates to satisfy constraints and optimize a performance measure. In this case, a flow network model and analysis of it effectively concern a single point in time.

Going forward, behavior will be added to the token-flow network definition. The difference between structure and behavior can be explained using the concept of *state*, the general condition of a system at any moment in time. Structure concerns what elements are in a token-flow network's state vector, and behavior concerns how that state vector can change with time. To add behavior, semantics are needed allowing a token-flow network to be in different states at different times, and this is where chapter 4 begins.

CHAPTER IV

TOKEN-FLOW NETWORK: BEHAVIOR

Structure concerns what elements are in a token-flow network's state vector, and behavior concerns how that state vector can change with time. For another explanation leading to the same definition of behavior, suppose a system is structurally described in a SysML user model, conforming instance models are stored in a relational database, and a table captures all instances of a Block or more generally any Classifier. In this scenario, the table and its columns are schema, rows are structure, values are state, and behavior concerns state change.

Throughout this chapter, recall the guidance for constructing a token-flow network definition - including semantics needed to support common questions about describing, predicting, and controlling discrete-event logistics systems.

4.1 Time, Behavior, State Change, and Events

Adding behavior to the token-flow network definition requires semantics allowing a token-flow network to be in different states at different times, shown in figure 35.

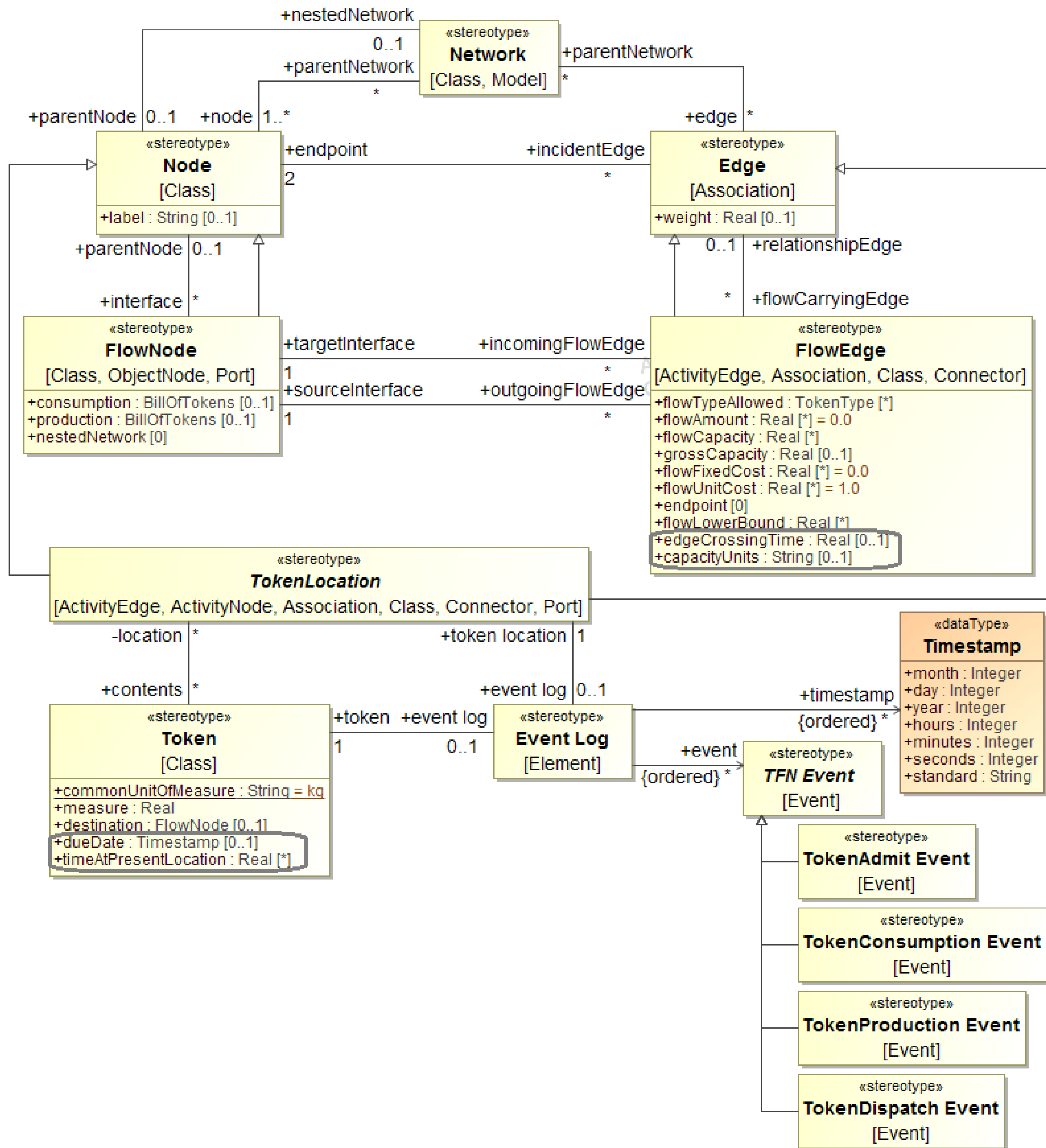


Figure 35: Semantics Introducing **Time** and **Events** into a Token-Flow Network.

Time, in a calendar date-and-time sense, is realized in the *Timestamp* semantic which is useful for discrete events. Timestamps are needed to answer questions about when state changes occur, length of time between them, and their absolute frequencies, although are not needed for questions about state change counts and relative frequencies.

Different states at different times often realizes as tokens in different locations at different times, enabled by the *TokenLocation* semantic. Number and position of tokens

is an important state component, and Node and Edge inherit a *contents* property from TokenLocation. Tokens have state too, part of which is location and possibly multiple locations in the case of nested networks. An implicit constraint is that any token's location and that location's contents should be consistent.

Any state change corresponds to an **event**, and the two are arguably synonymous. An *EventLog* records what changed and when, what recorded using a specific event subclass and when using a timestamp. TokenLocations and Tokens may each have an *EventLog*, which enables recording almost any imaginable state change in the token-flow network. Implicit constraints are that timestamps should be consistent for the same event recorded in multiple event logs, and timestamps in event logs should be consistent with dependent properties such as a token's *timeAtPresentLocation* and FlowEdge's *edgeCrossingTime*.

FlowNode and FlowEdge's definitions in chapter 3 say nothing about the dynamics of token input/output at FlowNodes, the dynamics of tokens crossing FlowEdges, nor what happens to tokens while resident at FlowNodes. All of these omissions are addressed in this chapter, but first consider FlowNode and FlowEdge's existing properties when flow has dynamics. Recall that any FlowNode has a flow conservation constraint, one per TokenType produced or consumed:

$$flow_{in} + production = flow_{out} + consumption$$

This constraint can still be enforced in a dynamic case, although only in an integrated-over-time sense, which requires additional semantics for the integration's endpoints. *Production* and *consumption* of tokens have no dynamics defined, addressed in the next section. The FlowEdge property *flowAmounts* loses its meaning absent any precision about time, and properties *flowCapacity* (per TokenType) and *grossCapacity* may constrain either integrated flow amounts or flow rates, where the new property *capacityUnits* can be helpful to explicitly identify what is constrained.

4.2 Behavioral Model of a Process

How can behavior be modeled, e.g. what are ways to describe how a token-flow network's state vector can change with time? A low-level behavioral model is a state machine, in which state transitions can be triggered by a random number generator (a Markov Chain) or by external inputs (a Finite State Automaton). A higher-level behavioral model developed here is a process. Both low-level state machine and higher-level process models of behavior have the underlying conceptual model of a token-flow network.

A **process** is defined as "*a set of interrelated tasks that, together, transform inputs into outputs.*" [1] A preferred definition is a "*sequence of interdependent and linked procedures which, at every stage, consume one or more resources (employee time, energy, machines, money) to convert inputs (data, material, parts, etc.) into outputs.*" [2] Phrases like "interrelated tasks" and "linked procedures" suggest that a process has the underlying conceptual model of a token-flow network, with semantics shown in figure 36.

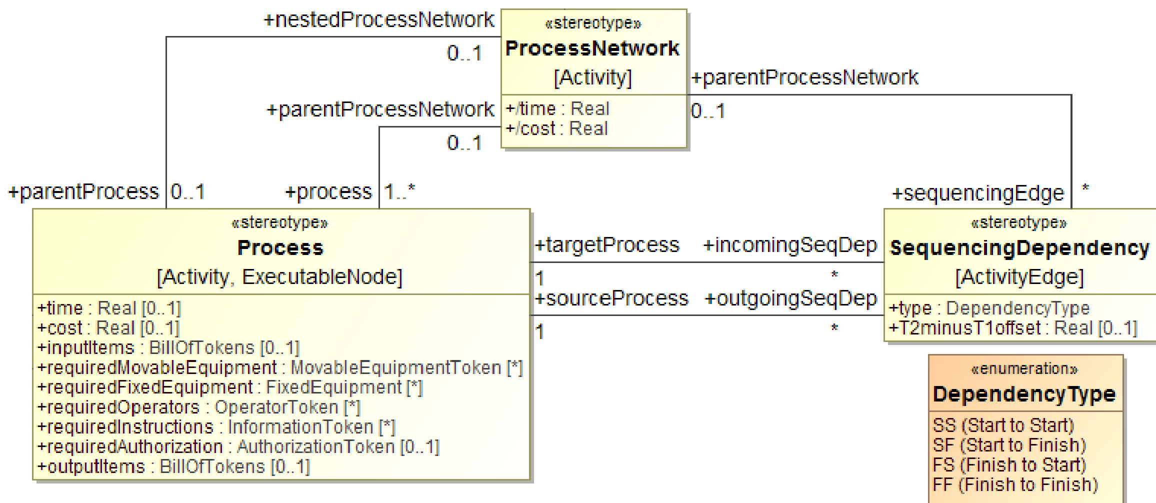


Figure 36: Semantics for a **Process** Behavioral Model in a Token-Flow Network.

Just as any Node may be elaborated as a nested network, any Process may contain a nested ProcessNetwork, allowing it to be atomic at one level of abstraction but refined into a Process Network at a lower level. The semantic *Sequencing Dependency* is borrowed

[1] http://en.wikipedia.org/wiki/Process_(engineering), viewed 19nov2013.
[2] http://www.businessdictionary.com/definition/process.html, viewed 19nov2013.

from the domain of project management and is used to model a simple form of execution semantics; the most common Sequencing Dependency is FS (Finish to Start), meaning the target process cannot start until the source process finishes. Sequencing Dependency execution semantics enable modeling higher-level strategic processes in discrete-event logistics systems, and also lower-level tactical processes if moving and waiting are explicitly included as sequenced processes. More generally, the semantics in figure 36 support modeling processes in which token flows are not explicit (note the absence of FlowNode interfaces and FlowEdges) and satisfaction of a Sequencing Dependency implies authorization for a process to begin execution. An example of a Process Network instance conforming to semantics in figure 36 is shown in figure 37. Note in the example that a Process' properties for required inputs can be useful for accounting, but cannot delay process execution.

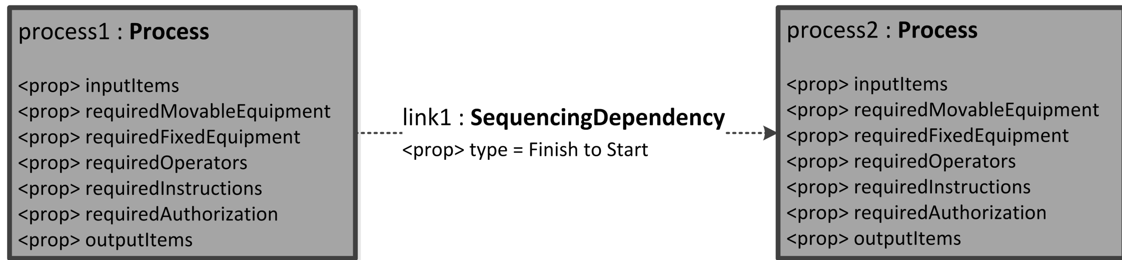


Figure 37: Example: A Process Network Instance Sequenced Explicitly using SequencingDependencies.

A different form of execution semantics are inspired by Petri Nets and involve token flows - a process begins execution as soon as all required inputs are available. This requires semantics enabling a Process to have incoming and outgoing token flows, which are added by inheriting FlowNode interfaces and FlowEdges from the existing network definition, shown in figure 38.

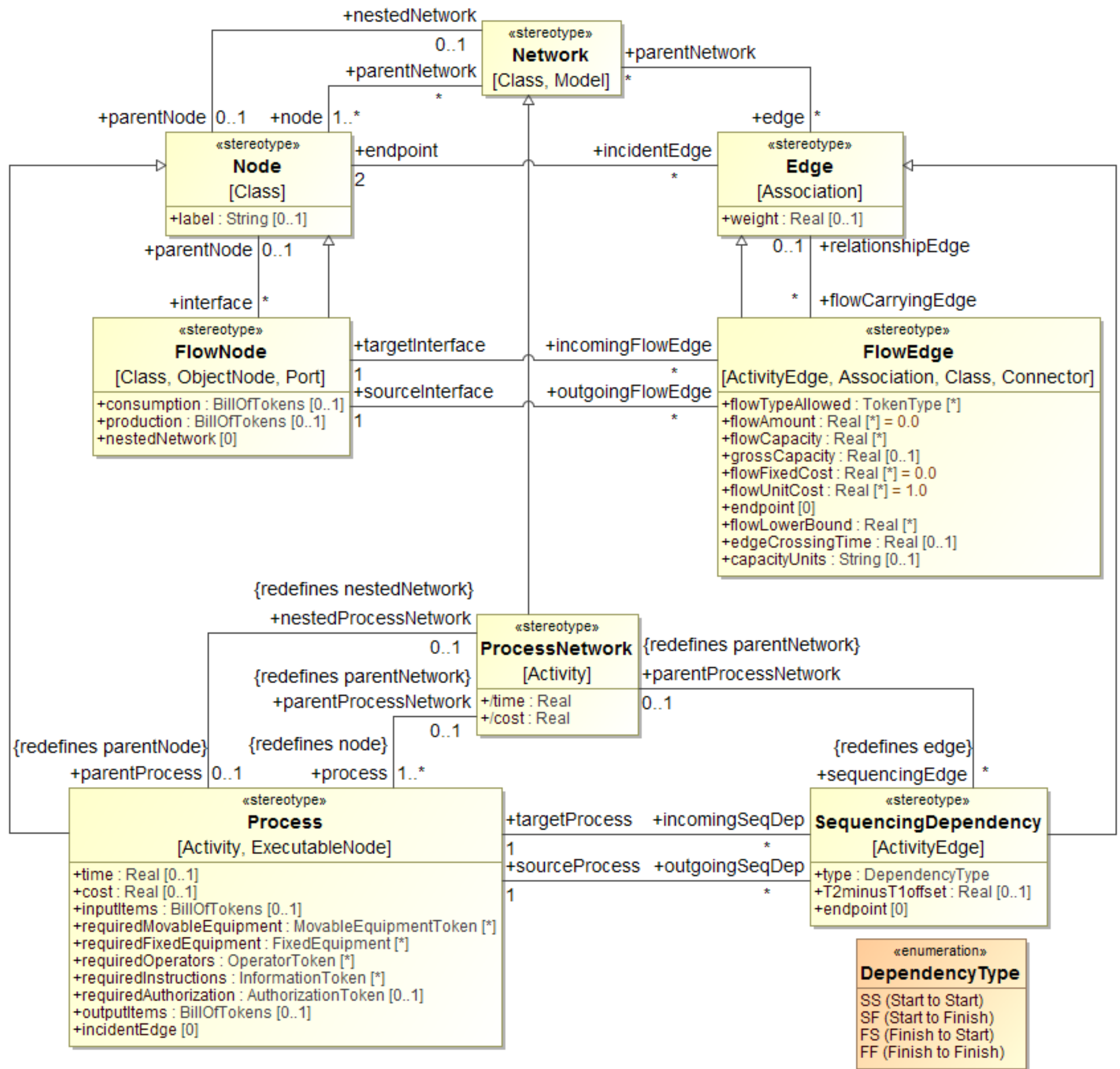


Figure 38: Semantics for a **Process** Behavioral Model in a Token-Flow Network, Integrated with Existing Semantics to Enable Token Flows.

Process subclasses Node, inheriting FlowNode interfaces for incoming and outgoing token flows. *SequencingDependency* subclasses Edge, inheriting an association with FlowEdges. Execution semantics are the same as SysML Activities and Petri Nets - when all required input tokens are available on all of a process' FlowNode interfaces, tokens in the types and quantities modeled by each interface's *consumption* property are immediately consumed and the process begins. [3] *Process* defines no semantics for what happens to a

[3]No use cases were considered in which a Process is interrupted mid-execution and it is necessary to sort out exactly what inputs have and have not been consumed at that point. If a Process has a nested Process Network, then mid-execution consumption information is partially available.

token upon consumption; this is left for subclasses to define. A fabrication process which irreversibly transforms raw materials will destroy consumed tokens and produce new ones upon completion, but other process types such as assembly, movement, and storage will produce the same tokens which were consumed. If a process has FlowNode interfaces, their aggregate *production* and *consumption* should be consistent with the Process' properties for *inputItems*, required resources, and *outputItems*. However, it is the *production* and *consumption* typing each FlowNode interface which ultimately enforces token requirements, not the Process' properties. An example of a Process Network instance conforming to semantics in figure 38 is shown in figure 39.

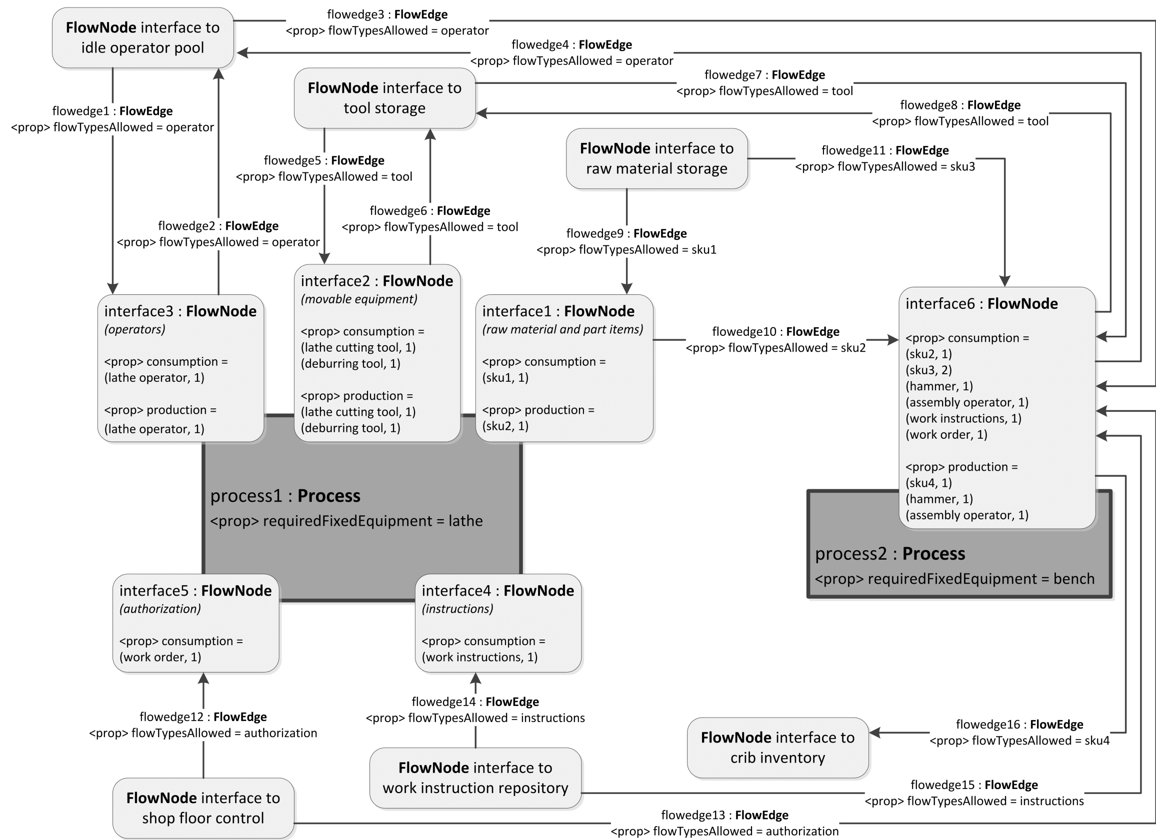


Figure 39: Example: A Process Network Instance Sequenced Implicitly using Token Flows.

In figure 39, both *process1* and *process2* begin execution when tokens are available on each FlowNode interface in the types and numbers specified by the *consumption* property. When process execution completes, tokens are produced onto each FlowNode interface in the types and numbers specified by the *production* property.

Combined use of both Sequencing Dependency and Token Input/Output execution semantics is conceptually possible in a single token-flow network, but not developed here. What implicit sequencing by token flows clearly models, but explict sequencing using Sequencing Dependencies only assumes, is memory. A token waiting on a Process' FlowNode interface serves as memory that an input item is available. Memory is required but only assumed if a Process has multiple incoming Sequencing Dependencies and somehow remembers which are satisfied and which are pending. Combined use of both types of execution semantics requires formalizing a Sequencing Dependency's notion of memory and then reconciling the two notions to work together. [4]

To support common questions about discrete-event logistics systems, the Process subclasses shown in figure 40 may be useful.

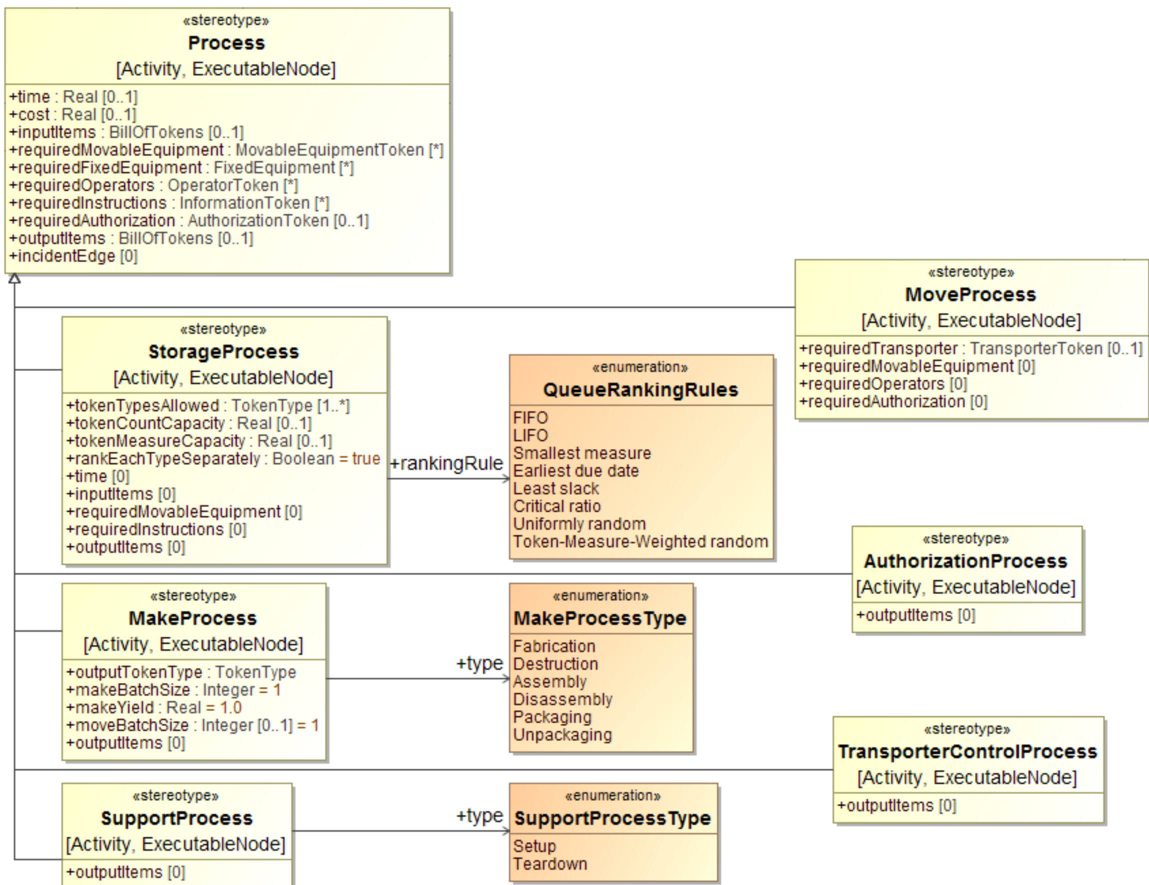


Figure 40: Process Subclasses.

[4]UML Activities accommodate both forms of execution semantics with token flow - a Sequencing Dependency is analogous to a control edge on which may flow control tokens.

A StorageProcess converts a token's age, a MoveProcess converts a token's location and possibly age, a MakeProcess may convert a token's existence and physical properties, and a SupportProcess is auxiliary, for tasks such as machine setups and teardowns. AuthorizationProcess and TransporterControlProcess serve control functions, discussed in section 4.5. A notable omission are subclasses for knowledge-collection processes including TestProcess, MeasureProcess, and ComputeProcess, which are a relatively easy addition if needed.

4.3 Hosting Behavior at a FlowNode

Structural semantics in chapter 3 enable constructing network models with physical analog, and semantics in section 4.2 enable constructing behavioral models of process networks. Important questions about discrete-event logistics systems require using structural and behavioral models in combination. For example, questions can be asked about describing, predicting, or controlling the execution of a functional manufacturing process plan, but high-fidelity answering analysis must also consider the physical manufacturing facility executing that process plan. A functional process network may allow ten processes to execute concurrently, but that may require more resources than a particular manufacturing facility has available.

New semantics *ConversionNode* and *ConversionEdge* are defined as subclasses of FlowNode and FlowEdge, with the added feature that they can host behavior. Since the only behavioral model defined in this dissertation is a Process Network, *ConversionNode* and *ConversionEdge* can host processes, as shown in figure 41.

Figure 41: ConversionNode and ConversionEdge to Host Behavior.

Hosting is a name for the concept which links a ConversionNode or ConversionEdge with the behavior executed there. In common use cases a ConversionNode will host a single process which may refine (recursively) into a nested process network. While a ConversionNode can host arbitrary processes, including a StorageProcess for input and output tokens, a ConversionEdge can only host a MoveProcess. Since a ConversionNode is a

FlowNode, a Process may have ConversionNode interfaces which themselves host processes, useful for making explicit the behavior of storing a process' input tokens pre-consumption and output tokens post-production. An important detail is that if a process is hosted at a ConversionNode, for example a hosted StorageProcess as just described, then the process cannot have its own FlowNode interfaces, a consistency requirement which will be explained in section 4.5.2. For a hosted process without FlowNode interfaces, *production* and *consumption* of the hosting ConversionNode enforce token requirements, which as before should be consistent with the process' properties for *inputItems*, *outputItems*, and required resources.

Semantics allowing FlowNodes and FlowEdges to host behavior enable modeling explicitly that when a token is resident at a ConversionNode for any non-zero amount of time, behavior must be happening. Corresponding state changes to resident tokens may concern existence, age, physical properties, packaging, location, knowledge, or more. Absent any behavior, tokens entering a ConversionNode must be immediately dispatched onto an outgoing FlowEdge. A token crossing a ConversionEdge also corresponds to behavior, which may acknowledge state changes enacted at the upstream FlowNode or state changes enacted by FlowEdge-crossing itself, for example to tokens' age and location. An example of a Process Network instance conforming to semantics in figure 41 is shown in figure 42.

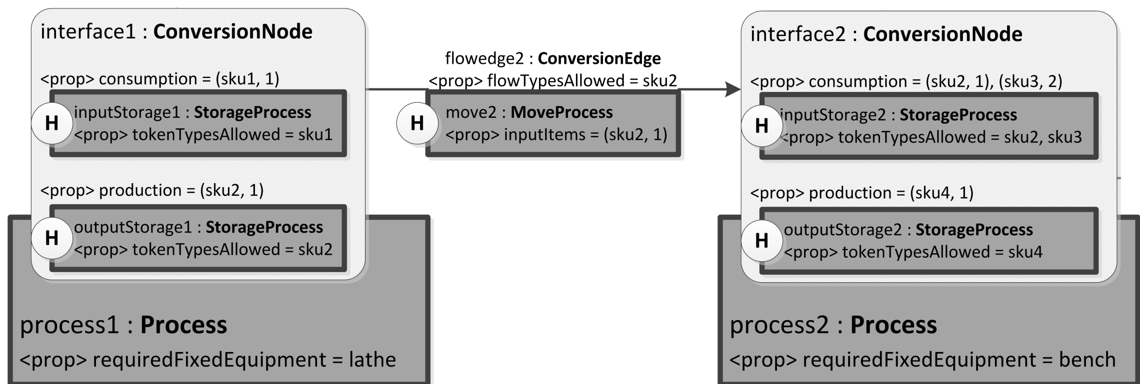


Figure 42: Example: ConversionNode and ConversionEdge Hosting Behavior. A circled H indicates hosting, in this example of StorageProcesses at ConversionNodes and MoveProcesses on ConversionEdges.

4.4 Resources

A preferred definition of a *process* was given in section 4.2: "*A sequence of interdependent and linked procedures which, at every stage, consume one or more resources (employee time, energy, machines, money) to convert inputs (data, material, parts, etc.) into outputs.*" Converting inputs to outputs requires resources in discrete-event logistics systems and many other domains, and important questions concern how to allocate limited resources. Just how important these questions are can be inferred from [Hopp and Spearman, 1996, p.270]'s observation that in manufacturing systems, "*actual process time (including setups, downtime, etc.) typically represents only a small fraction (5 to 10 percent) of the total cycle time in a plant ... The majority of the extra time is spent waiting for various resources (workstations, transport devices, machine operators, etc).*" This section defines resources and their corresponding tokens.

A **resource** is defined as anything which can enact or actively support enacting a state conversion. A resource must be more than the passive subject of a state conversion, so raw materials and parts do not qualify. Discrete-event logistics system resources may include people, real estate, facilities, capital equipment, shared public infrastructure, budgets, and information. Figure 43 defines resources for a token-flow network, and also a subclass of Token for any resource which can move.

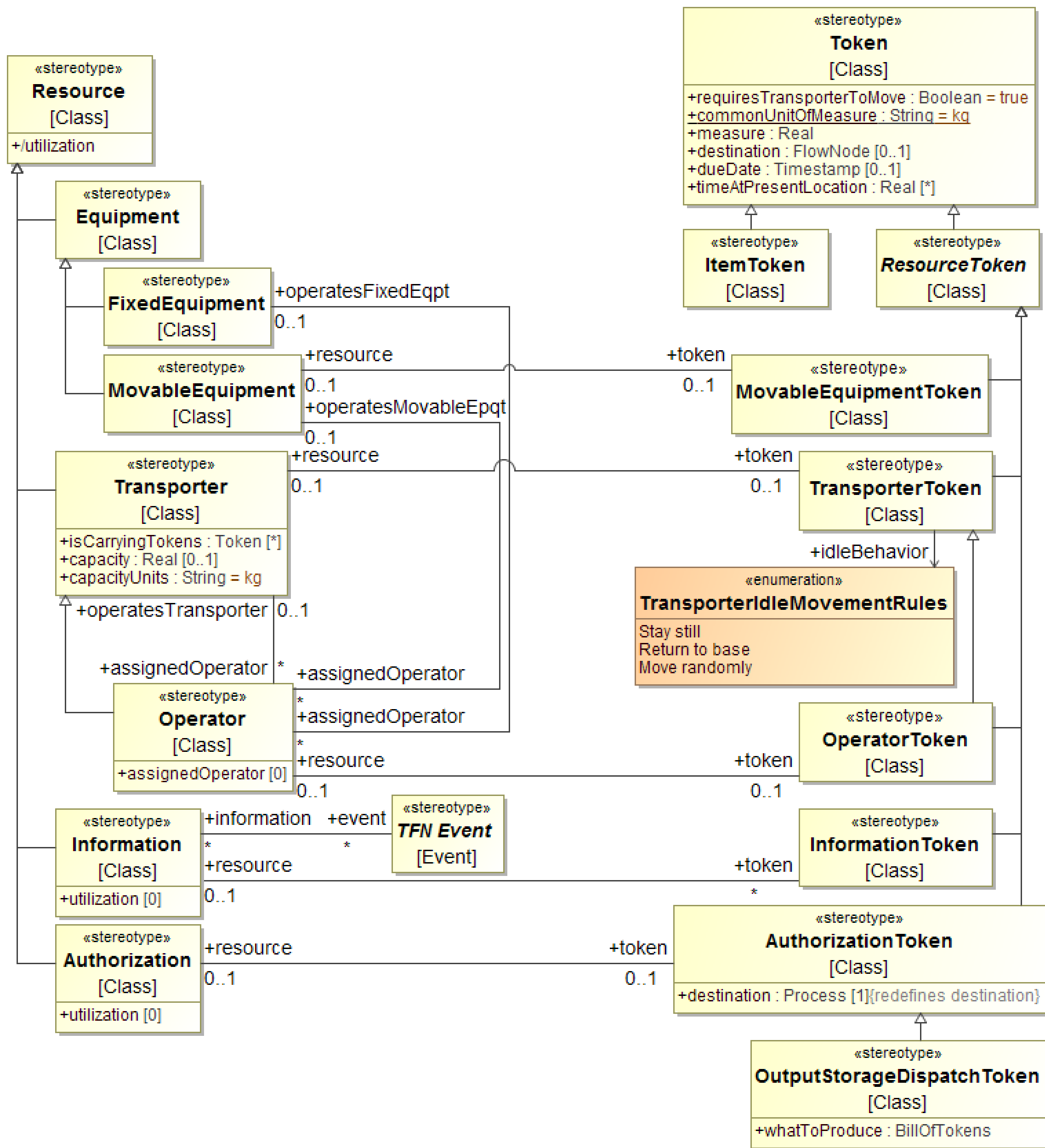


Figure 43: Semantics for **Resources** and Corresponding Tokens in a Token-Flow Network.

On the left-hand side of figure 43, *Equipment* and *Transporter* directly support token state conversions. *Information* and *Authorization* cannot enact conversions themselves, but can indirectly support conversions if they are required inputs for a process to begin execution. Regarding the right-hand side, recall that *TokenType* was defined in section 3.2 with metaclass DataType to allow arbitrary categories of tokens. The new Token subclasses in figure 43 add no new modeling capabilities, just higher-level semantics, and to

the schema rather than as instance data. With these resource semantics, now is a good time to revisit the definition of a *Process* and explain its properties for *inputItems* and resource requirements which are highlighted in figure 36 below. The highlighted properties and a Process' *outputItems* should be consistent with the *consumption* and *production* properties across all of a process' FlowNode interfaces, which are the ultimate enforcers of token requirements for processes whose sequencing is implicitly controlled by token input/output execution semantics.

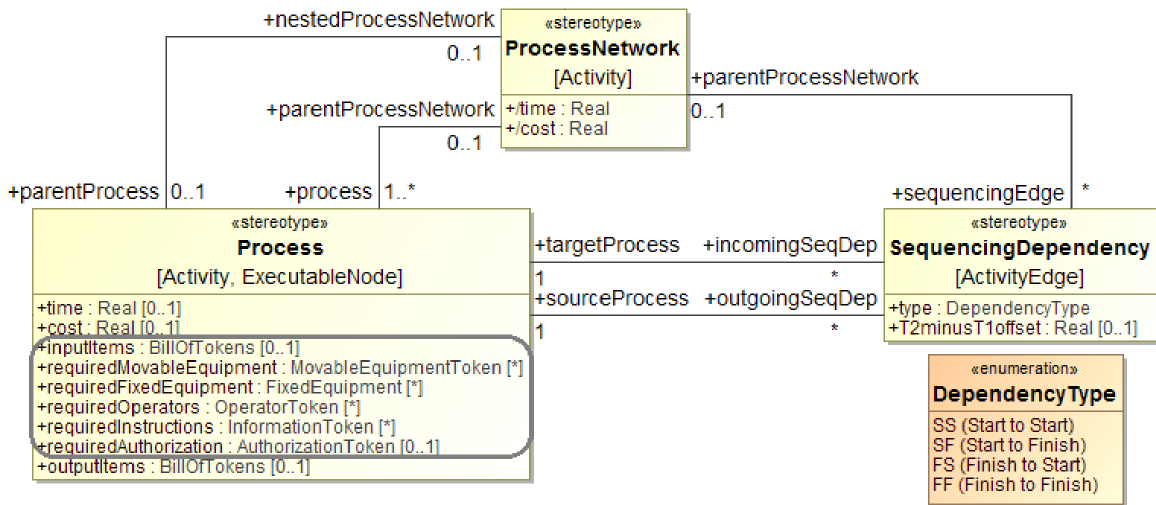


Figure 36: Semantics for a Process Behavioral Model in a Token-Flow Network, with Properties for *inputItems* and Resource Requirements Highlighted.

- *inputItems* may be raw materials and parts modeled with *ItemTokens*. ItemTokens do not correspond to resources because they are only the passive subject of a state conversion.

- *requiredMovableEquipment* may be tools and fixtures and modeled with *MovableEquipmentTokens*. Partitioning fixed and movable equipment requires precision - “movable” means during normal process execution. Any MovableEquipmentTokens consumed by a process must be produced when execution completes. [5]

- *requiredFixedEquipment* may be machines. Fixed equipment cannot move during

[5]This can be enforced using a constraint on the *consumption* and *production* properties across all of a process' FlowNode interfaces. However, an always-executing process may require enhancing these semantics because there is no current capability to input and output resource tokens mid-execution. One way to do this might be to borrow the concepts of *Seize* and *Release* from process-oriented discrete-event simulation languages.

normal process execution and has no corresponding token. If a process is hosted at a ConversionNode which lacks required fixed equipment, then the hosted process can never execute. If a ConversionNode lacks required movable equipment, however, then process execution is delayed until a corresponding token arrives.

- *requiredOperators* may be modeled with *OperatorTokens*. Humans are a difficult resource to classify because they can perform so many functions. In figure 43, note that fixed equipment, movable equipment, and transporter all have an *assignedOperator* property which can overlap with a process' *requiredOperators*, so the properties should be constrained consistent. Any OperatorTokens consumed by a process must be produced when execution completes.
- *requiredInstructions* may be work instructions and modeled with *InformationTokens*. What moves through a network is usually not an original information source such as a paper document or digital file, but rather a copy or link to information. Therefore, information is the only type of resource in figure 43 not in one-to-one correspondence with a token, because a single item of information may be associated with many copies or links.
- *requiredAuthorization* or permission may be modeled with *AuthorizationTokens*. Authorization is discussed further in section 4.5. Note that there are other functionally equivalent ways to convey authorization, for example with an InformationToken.

In addition to these properties of *Process*, the subclass *MoveProcess* may also require a vehicle resource modeled with a *TransporterToken*, which in manufacturing use cases may be forklifts, push carts, conveyors, and even operators. A TransporterToken carries tokens which cannot move themselves.

4.5 Plant/Control Interface in a Token-Flow Network

Consider a pattern for system modeling shown in figure 44.

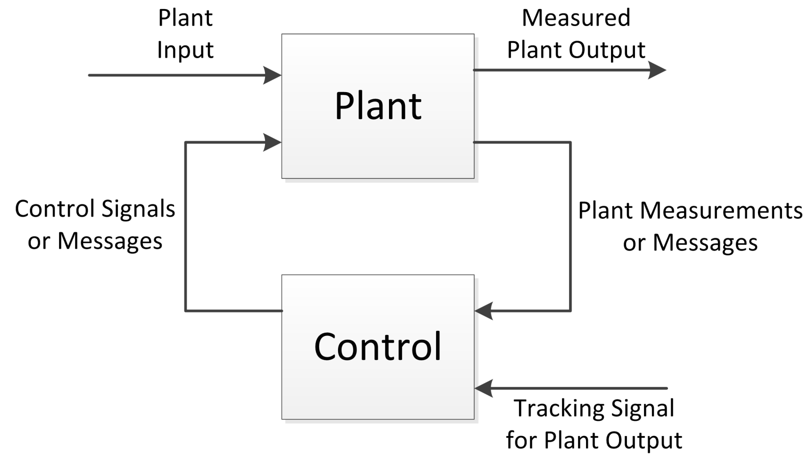


Figure 44: A Modeling Pattern of Plant/Control Separation.

What is the value of this pattern? In the describe / predict / control hierarchy, control is at the top and superceded only by design, which is out-of-scope. Controlling future observations is an essential level of understanding, and requires knowledge of sensors and actuators and more broadly what state values are controllable, the levers and mechanisms available to manipulate those values, and how those levers and mechanisms can be used to realize goals. The value of a Plant/Control separation pattern for system modeling is to help make all this knowledge explicit.

Figure 44 identifies separate concepts of *plant* and *control* and message-based or signal-based interaction between them. Plant is passive; imagine a manufacturing facility in which every machine, tool, operator, and transport vehicle remains idle waiting for instructions. Control is what makes everything in the manufacturing facility move and perform in a desired way. A subtle point is that even hardware whose exclusive purpose is control functions belongs in the *plant* category; the *control* category involves concepts like allocation, plan, planning horizon, policy, decision, re-evaluation frequency, and more. [6] In figure 44, the internal workings of the control function which convert measurements into

[6][Hopp and Spearman, 1996, p.381] offer high-level definitions in the context of production planning. They partition planning horizons into long, intermediate, and short, and define that long-range decisions address **strategy**, intermediate-range decisions address **tactics**, and short-range decisions address **control**.

decisions are not modeled here, only the plant interface points at which control decisions arrive and are enacted in a token-flow network [7].

Interface points between plant and control, and choices needing decisions at each, include:

- If, when, and in what numbers to admit tokens into FlowNodes. If a FlowNode is an interface to a Process, choices also concern when the Process consumes those tokens.
- If, when, and in what numbers to dispatch tokens from FlowNodes. If a FlowNode is an interface to a Process, choices also concern when the Process produces those tokens.
- Onto which outgoing FlowEdge to route dispatched tokens.
- The dynamics of a token crossing a FlowEdge.

To the list might be added controlling what happens to a token while resident at a Node. However, the only behavioral model defined here is a process, whose underlying conceptual model is a token-flow network, so this case's control mechanisms and choices are considered covered by the just-listed cases.

4.5.1 Admission of Tokens into FlowNodes

Only a small number of control mechanisms are modeled for controlling a token's *admission* into a FlowNode. This section describes those mechanisms, and then considers the subsequent question of when a process *consumes* tokens admitted into its FlowNode interfaces.

If a process has FlowNode interfaces, any FlowNode has a *consumption* property listing token types and quantities required for process execution to begin. In this way a FlowNode can filter admissible tokens, and (possibly dynamic) choices of admissible token types and quantities is a control mechanism. There may not exist much flexibility for a MakeProcess requiring a specific bill of materials, but there may be more flexibility

[7] In addition to individual decisions, this section also uses the word *policy* for a collection of decisions, sometimes time-indexed. [Schruben and Yücesan, 1993, p.266] define *policy* as: "The rules that govern the interaction of entities within a system are called *laws* if they are not under our control and *policies* if they are."

for StorageProcess, MoveProcess, and others. Another control mechanism is *blocking* a FlowNode or a FlowEdge, shown in figure 45.

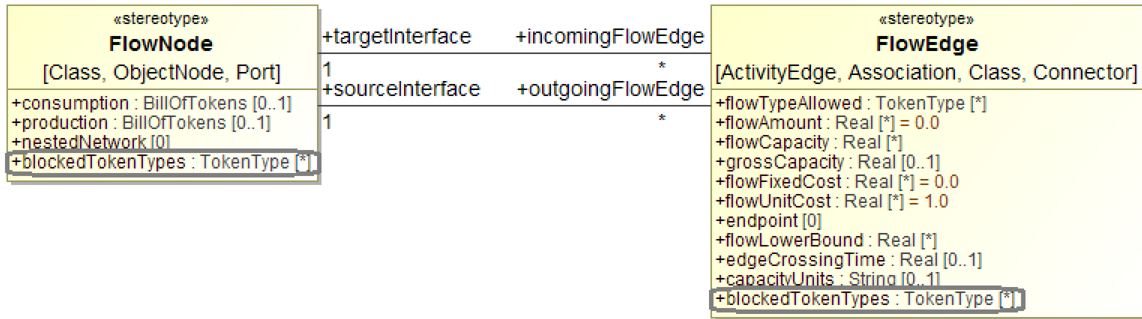


Figure 45: FlowNode and FlowEdge with Blockage properties.

Blocking is type-specific; a FlowNode or a FlowEdge may be blocked for token type A but not type B. Blocking a FlowNode means tokens cannot be admitted into the FlowNode, and may be caused by full or nonexistent storage, the concept of failure, the concept of maintenance, and more. Blocking a FlowEdge means that no tokens may be dispatched onto the FlowEdge, and may be caused by a blocked FlowNode target, the concept of failure, the concept of maintenance, and more. If a token crosses a FlowEdge but the target FlowNode is blocked for that type, then the token blocks the FlowEdge for its type until the target FlowNode's blockage clears. FlowNode and FlowEdge's *blockedTokenTypes* properties and FlowNode's *consumption* and *production* properties are all interface points between plant and control.

Filtering and blocking are the only control mechanisms defined for admitting a token into a FlowNode. After admission, a token may be consumed by the parent process, and defining how this works more concerns behavioral execution semantics than control choices. If not immediate, for example if all required inputs are not yet available on all of a process' interfaces, then storage is needed. FlowNode's subclass ConversionNode exists to host processes and may host StorageProcesses for input and output tokens, as was shown in figure 41. Even if a ConversionNode hosts a storage process, storage may not be available for an arbitrary token depending on allowed types and capacities. Types and capacities of StorageProcesses are another control mechanism. Storage processes hosted at

ConversionNodes are not the only possible mechanism to store tokens; another possibility for any process in which required input tokens may not arrive all at once is for the process to contain a nested process network with StorageProcess as the first step. In both possibilities, note something unusual about a StorageProcess - it may consume and produce tokens mid-execution.

Execution semantics were mentioned earlier in section 4.2 - when all required input tokens are available on a FlowNode interface, either in storage or blocking an incoming FlowEdge, then that interface is *active* or *enabled* in Petri Net vocabulary. When all of a process' FlowNode interfaces are active, then the process immediately consumes all required input tokens from each interface (but not any extras) and begins execution. Again, it is the interfaces' *production* and *consumption* which ultimately decide inputs and outputs, and for processes sequenced implicitly using token flow those properties should be consistent with the process' properties *inputItems*, *requiredOperators*, *requiredAuthorization*, etc. An all-tokens-up-front paradigm may be semantically limiting, especially for inputs consumed in later process steps for which just-in-time is desired. However, the notion of "later process steps" implies a nested process network which can replace the parent process. Then an all-tokens-up-front paradigm shifts from the parent process to each nested process. This is not a perfect solution, as it recursively pushes the issue to lower abstraction levels rather than eliminates it, but it suffices for many questions about discrete-event logistics systems.

4.5.2 Dispatching Tokens from FlowNodes

This section concerns semantics and control mechanisms for controlling a token's dispatch from a FlowNode onto an outgoing FlowEdge. First, however, this section concerns the upstream question of when a process produces output tokens onto its FlowNode interfaces. Semantics are that when a process' execution completes, the *production* property of each FlowNode interface specifies token types and quantities immediately produced onto the interface. The default behavior can be redefined, and that is done by several Process subclasses as shown in figure 46.

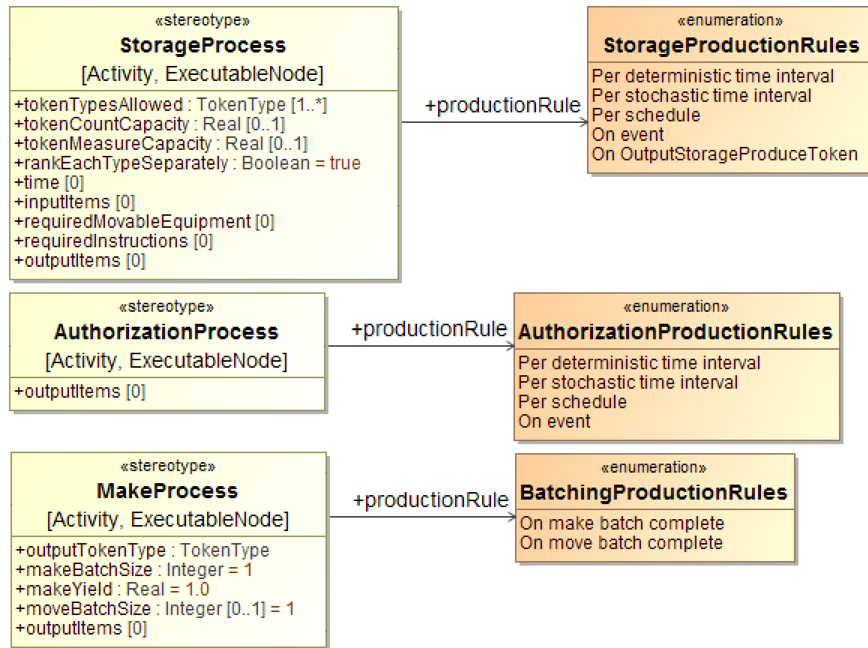


Figure 46: Process Subclasses with Output Production Rules.

- A MakeProcess introduces the concept of batching, both a make batch and a move batch, which need not be the same size - production lots might be 10,000 units to avoid costly machine setups, which might be divided into 20 move batches of 500 units each for the sake of variability reduction. A MakeProcess outputs a single token type, and choices of make and move batch sizes are control mechanisms at the plant/control interface. If the move batch size is less than the make batch size, a MakeProcess produces tokens onto FlowNode interfaces upon completion of each move batch while process execution continues. Completion of a make batch marks the finish of process execution, and all remaining output tokens are produced onto FlowNode interfaces, whether or not they comprise an entire move batch. [8] Also included is the concept of *yield* - not all output tokens may be produced correctly and some may be rejected. Imperfect yield requires the ability to dynamically update *production* quantities on a MakeProcess' FlowNode interfaces.

[8]Semantics are to produce output tokens for an entire move batch all at once upon batch completion. What this does not model is accumulation of a partial batch. Alternative semantics are to produce output tokens onto the interface one-at-a-time, and define move batch semantics for a FlowNode interface rather than a Process. If this alternative is preferred, then move batch semantics defined for a FlowNode interface must be consistent with semantics of a MoveProcess possibly hosted on an outgoing ConversionEdge.

- An AuthorizationProcess can support scheduling controllers in discrete-event logistics systems, may be always-running, and may produce AuthorizationTokens mid-execution, one-at-a-time, and earmarked for specific destinations. [9] A control mechanism at the plant/control interface is *when* authorization tokens are produced, and a few common policies are enumerated in figure 46, but nothing is modeled for how to choose an AuthorizationToken's destination and routing.

- A StorageProcess may be always-running and may consume and produce tokens mid-execution. If a StorageProcess is stand-alone (not hosted) and has its own FlowNode interfaces, process execution may be delayed for required input tokens. Once execution begins, any token consistent with the *tokenTypesAllowed* property whose consumption will not exceed capacities is immediately consumed. A control mechanism at the plant/control interface is *when* to produce tokens from a StorageProcess, and a few common policies are enumerated in figure 46. A complementary choice is *what* to produce, for which a few primitive mechanisms are defined:
  - Produce the highest-ranked token in a single queue.
  - Produce token types and quantities of the *production* property on a StorageProcess' FlowNode interfaces (this only works if stand-alone and not hosted).
  - Produce token types and quantities of the *whatToProduce* property of an OutputStorageDispatchToken, which was defined in figure 43 and will be explained shortly.

When a StorageProcess is hosted at a ConversionNode instead of being stand-alone, semantics are more involved. A hosted StorageProcess may require fixed equipment such as shelves or racks, which a hosting ConversionNode must have or else the assignment is bad. A bigger issue is consistency among several levels of parent/child hierarchy. For an example, figure 47 shows a simplified version of figure 39 in which Processes have ConversionNode interfaces for token input/output, and those interfaces each host StorageProcesses.

[9] An AuthorizationProcess is defined for explicitly modeling the generation of a specific kind of control message; this process subclass is effectively a shell for a controller without any internal logic defined. An alternative is to have authorization control messages move implicitly using Events.

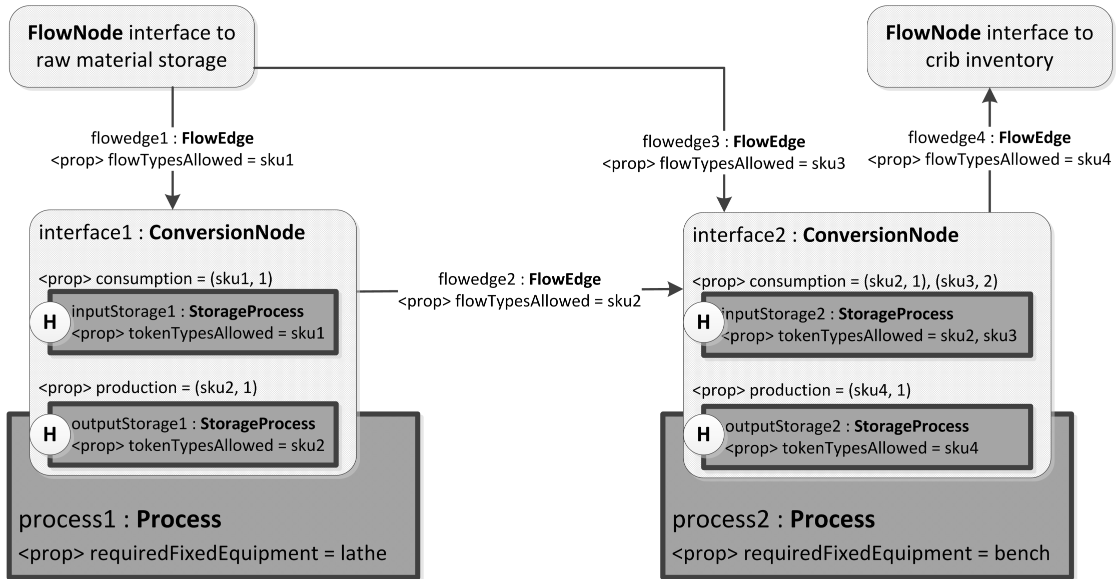


Figure 47: A Process Network Instance with Three Levels of Parent/Child Hierarchy. A circled *H* indicates hosting, in this example of a StorageProcess at a ConversionNode.

There are three levels of parent/child hierarchy in figure 47: Process → ConversionNode → StorageProcess. To explain a semantic difficulty with hosted processes, consider a forbidden scenario in figure 48 which adds an additional level of parent/child hierarchy, where the hosted StorageProcesses *inputStorage1* and *outputStorage1* themselves have FlowNode interfaces for token input/output.

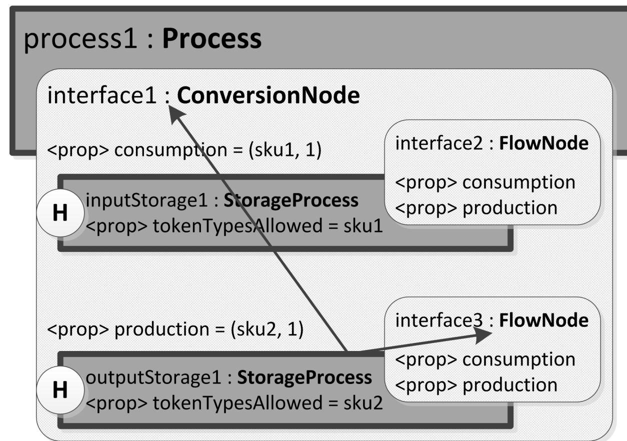


Figure 48: A Process Network Instance with Four Levels of Parent/Child Hierarchy. A circled *H* indicates hosting, in this example of a StorageProcess at a ConversionNode. **This is a forbidden scenario.**

There are four levels of parent/child hierarchy in figure 48: Process → ConversionNode → StorageProcess → FlowNode. Suppose *outputStorage1* produces a token. Onto which interface should it be placed - its own *interface3* or its parent *interface1*? This is the ambiguity resulting in the restriction that a hosted process may not have its own FlowNode interfaces (*interface2* and *interface3* in figure 48 are disallowed). If a process is hosted, then how is token input/output handled? Token input/output routes through *consumption* and *production* of the hosting ConversionNode, unlike earlier cases. If the hosting ConversionNode is itself an interface to a process, however, any *consumption* is by the parent *process1*, and tokens are only diverted to the hosted *inputStorage1* in specially-defined cases such as *process1* awaiting additional input tokens. Hosted StorageProcesses do have value for buffering a parent Process' input and output tokens, so to work around this issue special token subclasses can be defined targeting an interface's hosted processes rather than its parent process. One example is *OutputStorageDispatchToken* targeted at a ConversionNode's hosted StorageProcess instead of any parent Process. Because of difficulty with token input and output, a hosted StorageProcess is assumed to be always-running. Also, note that the workaround is unnecessary for processes hosted at a ConversionNode which is stand-alone (not an interface to a parent Process), which is a common use case for a physical network with fixed and mobile resources hosting and executing a functional process plan.

A StorageProcess has one or more queues. Default semantics are a separate queue for each token type, which accommodates producing a Bill Of Tokens - for each token type, produce the specified quantity from the front of that type's queue. If *rankEachTokenTypeSeparately* is false, however, then all token types share a single queue. As modeled, all queues within a single StorageProcess share the same ranking rule, although the definition can be changed to allow independent rules.

So far this section has concerned semantics and control mechanisms for a process producing tokens onto its FlowNode interfaces, but has said nothing about *dispatching* those tokens onto outgoing FlowEdges. Dispatch execution semantics are straightforward: Tokens produced onto a FlowNode are immediately dispatched onto an outgoing FlowEdge,

and a choice among multiple possibilities is discussed in the next section. However, there also exists the control mechanism that a ConversionNode may host a StorageProcess for output tokens as shown in figure 41. Storage between a token's production and dispatch can be useful in at least two cases:

- If an outgoing FlowEdge is blocked, and an alternative routing is not possible or permissible, then tokens of the blocked type(s) are stored until the blockage clears.
- If an outgoing ConversionEdge hosts a MoveProcess, and the MoveProcess is perhaps awaiting a transporter and not ready to begin, then tokens are stored until the MoveProcess is ready to consume them.

In many cases control mechanisms for dispatching tokens are deferred to upstream production of tokens. However, whether to store an output token post-production, how to enact that choice (for example, by blocking outgoing FlowEdges), and when to produce output tokens from storage are additional control mechanisms at the plant/control interface.

4.5.3 Routing a Dispatched Token

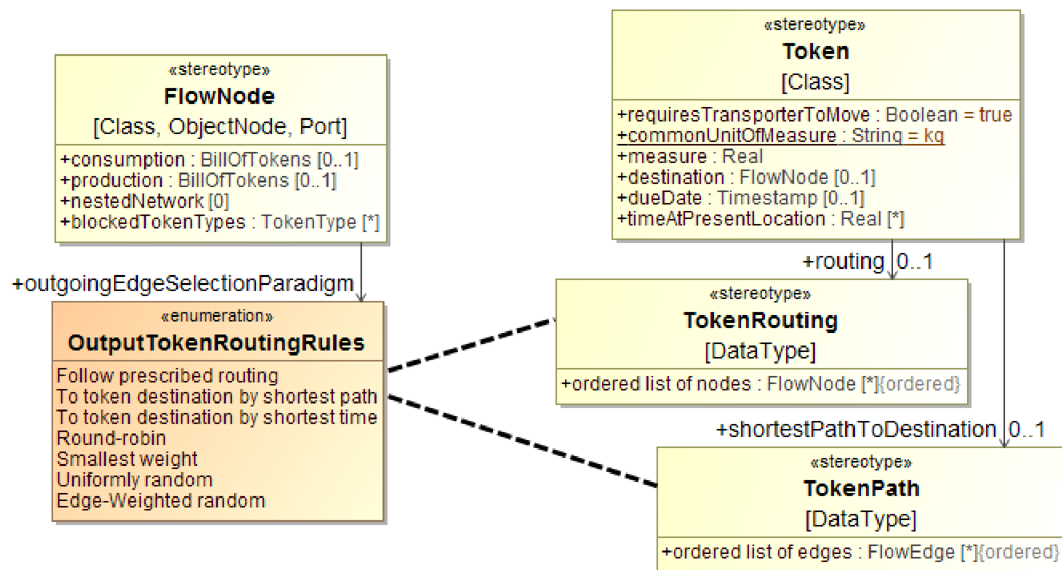


Figure 49: Several Routing Control Policies for Dispatching Tokens from a FlowNode onto an Outgoing FlowEdge.

Onto which of a FlowNode's outgoing FlowEdges to dispatch a just-produced token is a control choice at the plant/control interface. A few common policies are enumerated

in figure 49, and a (possibly dynamic) choice among them is captured with FlowNode's *outgoingEdgeSelectionParadigm* property.

- **Follow prescribed routing**: Any token may have a prescribed *routing*, a sequence of nodes which should be consecutively connected by directed edges. Choosing a token's routing is a control mechanism at the plant/control interface. A routing may contain degrees of freedom, for example if consecutive nodes are linked by multiple parallel edges, in which case a secondary rule is needed such as 'smallest weight' or 'uniform random'. A token's routing may be dynamically updated, again a control mechanism.

- **To token destination by shortest path**: A token may be earmarked for a particular destination, and 'by shortest path' may seem to reduce this to a prescribed routing. However, a path is a sequence of edges whereas a routing is a sequence of nodes. A shortest path can be dynamically re-computed as network state changes, as can a routing, but a semantic difference is that here only the destination is of interest, whereas in a routing the intermediate nodes may be of interest too. A token can store an edge sequence in its *shortestPathToDestination* property.

- **To token destination by shortest time**: FlowEdge has an *edgeCrossingTime* property, which enables recording a notion of crossing time distinct from whatever the inherited *weight* property models, for example length. *edgeCrossingTimes* may be dynamically updated as storage processes congest with tokens and FlowNodes and FlowEdges become blocked for particular token types. The semantic difference between this and the previous two policies is that shortest-time may not realize as a pre-computed routing (sequence of nodes) or path (sequence of edges) but rather as sequential decisions made immediately before a token is dispatched from successive FlowNodes. Nothing is said about how to compute a shortest-expected-time path, although ingredients such as Process' *time*, FlowEdge's *edgeCrossingTime* and inherited *weight*, and a StorageProcess' inherited *contents* may be helpful.

- **Round-robin**: A FlowNode's outgoing FlowEdges take turns accepting dispatched tokens, and the turn rotates in a fixed sequence. Semantics are that tokens dispatched together are routed together, meaning that successive turns may dispatch unequal numbers of tokens. An alternative is one token per turn, for which semantics can be redefined if preferred. Not modeled is any sequencing among a FlowNode's outgoing FlowEdges, nor any mechanism for remembering which FlowEdge is next in the sequence - an implementation version of the token-flow network profile should include these omissions.

- **Smallest weight**: Output tokens are dispatched onto the outgoing FlowEdge with smallest weight. Not modeled is a tiebreaking rule if multiple FlowEdges have the

same weight. Note that the *weight* property may dynamically update, which is a control mechanism at the plant/control interface.

- **Uniformly random**: All outgoing FlowEdges have an equal probability of selection, and one is chosen at random. If there are n outgoing FlowEdges then each has a $1/n$ probability of selection. As with round-robin, semantics are that tokens dispatched together are routed together, which can be redefined to one token per turn if preferred.

- **Edge-Weighted random**: Each of a FlowNode's outgoing FlowEdges has a selection probability proportional to its weight, and one is chosen at random. Selection probability is defined as an edge's weight divided by the sum of all outgoing FlowEdges' weights. As with round-robin and uniformly random, semantics are that tokens dispatched together are routed together.

4.5.4 How a Token Crosses a FlowEdge

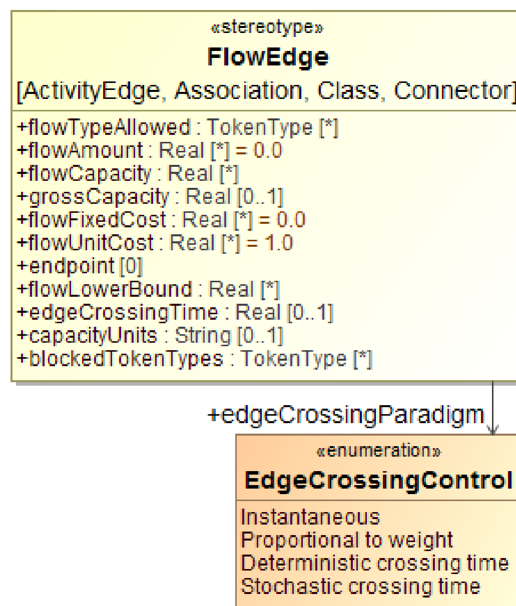


Figure 50: Several Flow Control Policies for a Token Crossing a FlowEdge

The dynamics of how exactly a token crosses a FlowEdge involve behavioral semantics and control mechanisms at the plant/control interface. A few simplifications of these dynamics are enumerated as control policies in figure 50, and a (possibly dynamic) choice among them is captured with FlowEdge's *edgeCrossingParadigm* property.

- **Instantaneous**: The simplest policy, which can support anything moving electrically or optically depending on the fidelity required. FlowEdge's *edgeCrossingTime* must be zero for consistency.

- **Proportional to weight**: This policy models *edgeCrossingTime* proportional to a FlowEdge's *weight* property. Not modeled is the constant of proportionality.

- **Deterministic crossing time**: This policy implies that *edgeCrossingTime* is a literal number or derived per a deterministic formula.

- **Stochastic crossing time**: This policy implies that *edgeCrossingTime* is either a literal random variable specification (e.g. $Normal(0, 1)$ or $Exponential(2)$), a derived random variable specification (e.g. $Normal(\mu, \sigma^2)$ with deterministic formulas for μ and σ^2), or some other stochastic specification such as the ingredients for a kernel density estimator.

In addition to FlowEdge's *edgeCrossingParadigm* property, other semantics and control mechanisms are in the definition of a MoveProcess possibly hosted on a ConversionEdge. Figure 51 shows an extension of figure 47 in which ConversionEdges host MoveProcesses.

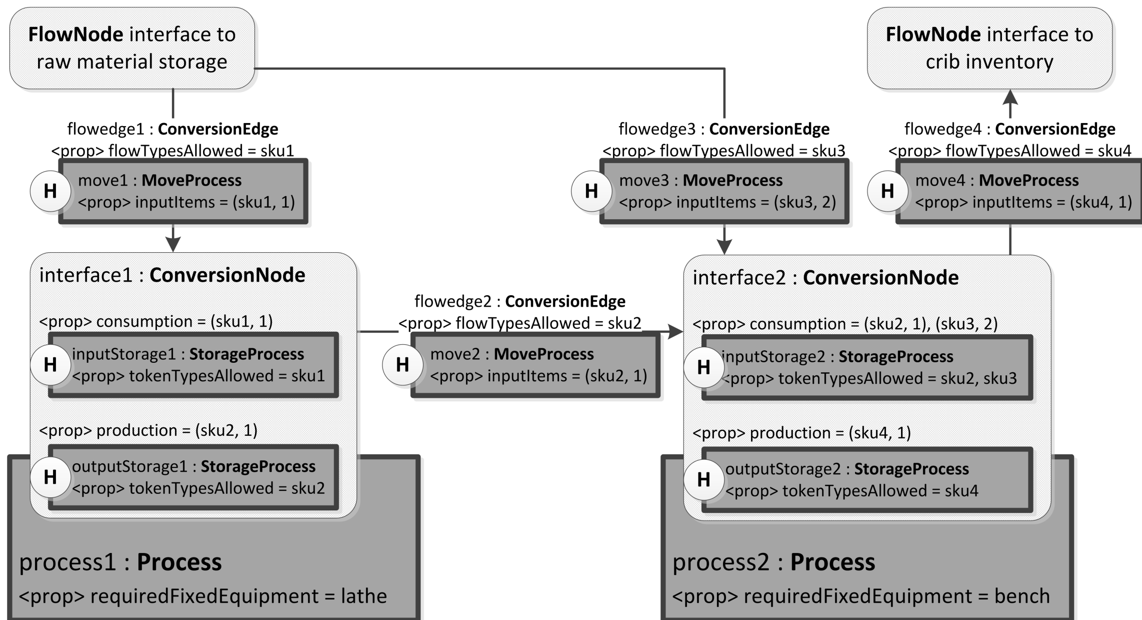


Figure 51: A Process Network Instance in which each ConversionEdge hosts a MoveProcess. A circled *H* indicates hosting, here of StorageProcesses at ConversionNodes and MoveProcesses on ConversionEdges.

MoveProcess inherits a Process' properties for input items (think Bill Of Lading), output items (usually the input items), fixed equipment, authorization, and adds a

property *requiredTransporter* for tokens which cannot move themselves. MoveProcess may also contain a nested process network, which allows refinement into steps like loading, transportation, unloading, inspection, and accepting or declining delivery. If a MoveProcess is stand-alone and not hosted, it may have its own FlowNode interfaces for input and output tokens. If hosted on a ConversionEdge as in figure 51, it cannot have FlowNode interfaces and additional semantics are needed.

A MoveProcess hosted on a ConversionEdge has a source FlowNode and target FlowNode. Semantics are that tokens dispatched onto the ConversionEdge are immediately consumed by the MoveProcess, and upon completion are immediately produced for admission into the target FlowNode. MoveProcess inherits a property *requiredFixedEquipment* (such as rails or conveyors) which a hosting ConversionEdge must have or else the assignment is bad. A difficulty is that TransporterTokens and AuthorizationTokens must be input/output to/from a MoveProcess, but if hosted there are no interfaces for token flow. A hosted MoveProcess does effectively have interfaces, however, at the ConversionEdge's endpoints. A transporter always loads tokens at a ConversionEdge's source and unloads them at a ConversionEdge's target; therefore, a hosted MoveProcess' TransporterTokens can be admitted at the source and dispatched onto the target. Authorization for a hosted MoveProcess can be accommodated upstream, because a StorageProcess hosted at the ConversionEdge's source may wait for an OutputStorageDispatchToken to produce tokens. An alternate is to extend the token-flow network defintion with a special subclass of AuthorizationToken directed at hosted MoveProcesses.

4.6 Summary

To support the abstraction method, chapters 3 and 4 define the abstract concept of a token-flow network, both structure and behavior. Semantics and modeling choices can be debated, but these chapters should provide compelling evidence that a token-flow network can generalize a wide range of discrete-event logistics systems and underlies many analyses of them. Since a network (for structure) and token-flow network (for behavior) are also underlying conceptual models for the SysML language, a token-flow network seems a

particularly good choice for the abstraction model in this dissertation's methodology.

The next chapter supports the formalization method, not for systems or abstractions but for questions about them. If arbitrary systems can be formalized with the SysML language, and a token-flow network formalized as a UML profile using semantics in chapters 3 and 4, what language should be used to formalize questions? Chapter 5 contains an initial study of semantics, categories, and patterns in questions, which when coupled with syntax may comprise a formalism for questions with broader scope than any known query language.

CHAPTER V

SEMANTICS OF A WELL-FORMED QUESTION

Questions play a central role in the description, prediction, and control of discrete-event logistics systems. Any decision answers a "What should I do?" question. Decision-making is commonly supported by evaluating alternatives, with evaluation done by posing and answering questions about a system's structure or behavior. Answering questions invariably involves the development and analysis of models, which may be explicit or implicit, qualitative or quantitative, analytical or computational. For these reasons, the process in this dissertation's methodology is question-driven. Figure 52 shows a general version of the process in which *Context Model* represents either a system model or an abstraction model.

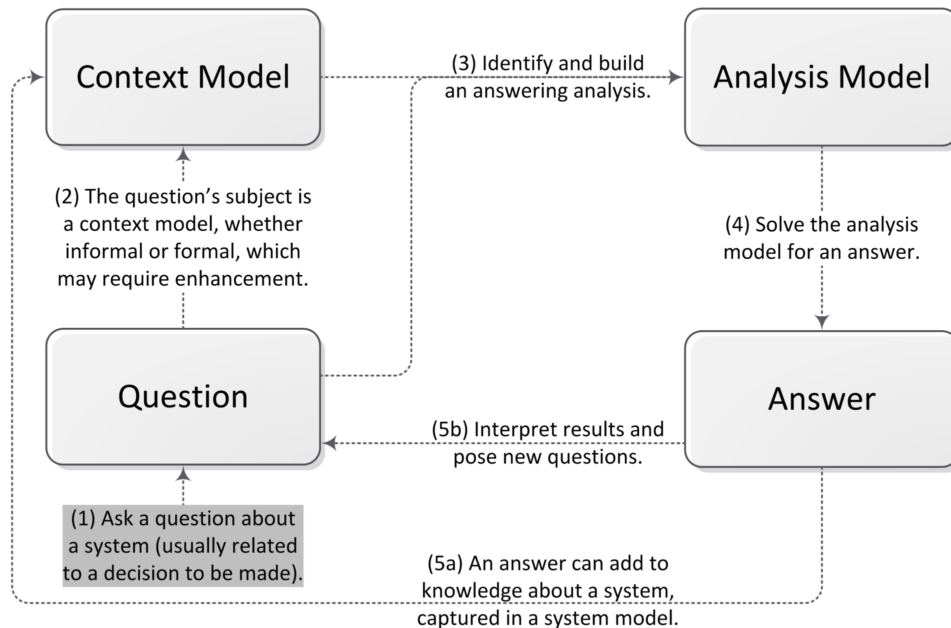


Figure 52: The Process in this Dissertation's Methodology, in which *Context Model* represents either a system model or an abstraction model.

This chapter presents an initial study of formalizing questions in the context of designing and operating discrete-event logistics systems. What distinguishes developments in this

chapter from prior-work query languages was discussed in chapter 2 and is important enough to repeat here. Most contemporary query languages are paired with a particular modeling language, and can only express questions about models captured in that language. Further, a tacit restriction is that queries are only answerable by navigating through a model and retrieving recorded information. This chapter does not restrict attention to questions about system models captured in any particular modeling language, nor to questions answerable by navigating through a model and retrieving recorded information. Questions may have subjects which are not an explicit context model element, and be answered by formulating and solving an analysis model. The goal here is an original development of semantics, categories, and patterns in questions which induce engineering analysis, and the results of an initial study are documented in chapter 5. Syntax is left for future work, which might involve expressing the semantics in the Object Query Language's (OCL's) syntax.

For this initial study, assume that the subject of a question is a system or token-flow network instance model with concrete instance data, as opposed to a user model without conforming instance models or a metamodel. This assumption ensures that answering analysis models have instance data and can be solved for an answer. General question semantics include:

- All information needed to identify and build an analysis model in figure 52's process must be captured in either the context model or the question.
- Analysis is performed for a purpose, and a question contains that purpose - what information is sought about which context model elements?
- A question is independent of any answering computational process. This is important because query languages such as SQL can make it easy to conflate a question with an answering analysis model, even though they are two distinct concepts.
- A pattern for questions is proposed as "(Question Word) is (Question Subject) (Predicate Modifiers)?" *Question Words* include who, what, which, when, where, how, and why. *Question Subject* (which may have modifiers) and *Predicate Modifiers* are the links between a question and a context model, and concentrated in these elements is the purpose of answering analysis.

Additional question semantics such as characterizing a *Question Subject*, its modifiers,

and *Predicate Modifiers* can only be developed for narrower categories of questions. "Narrower categories of questions" implies a taxonomy for questions which induce engineering analysis, which is the subject of the next section.

5.1 A Taxonomy of Questions

A question contains analysis' purpose, an axiom from section 1.1 is that analysis' purpose is understanding, and [Rouse, 2009] refines understanding into the hierarchical levels of *describing* past observations, *predicting* future observations, and *controlling* future observations. Therefore, describe / predict / control seems a natural taxonomy for questions. Another high-level taxonomy which is fundamental to system modeling is structure / behavior. [1] The cross product of these two classifications applied to questions suggests six categories - questions about describing structure, predicting structure, controlling structure, describing behavior, predicting behavior, and controlling behavior. However, an issue with these categories is that they overlap, which is undesirable because disjoint categories are conceptually simpler, and also that important variables such as time are missing.

The difference between structure and behavior was explained in section 3.7 using the concept of *state*, the general condition of a system at any moment in time. Structure concerns what elements are in a token-flow network's state vector, and behavior concerns how that state vector can change with time. Prediction and control inherently concern state change spanning multiple points in time, and therefore prediction and control inherently concern behavior, and therefore questions about predicting and controlling structure are inherently questions about predicting and controlling behavior. Six overlapping categories can be reduced to four disjoint ones, and figure 53 shows this simplified classification of questions with disjoint leaf categories and also incorporates the concept of time.

[1] Another taxonomy introduced in section 4.5 is plant/control, but since no general definition of *control* is referenced or developed here, this classification is not pursued. A missing definition of *control* will prove limiting in this chapter when developing the last category in describe / predict / control.

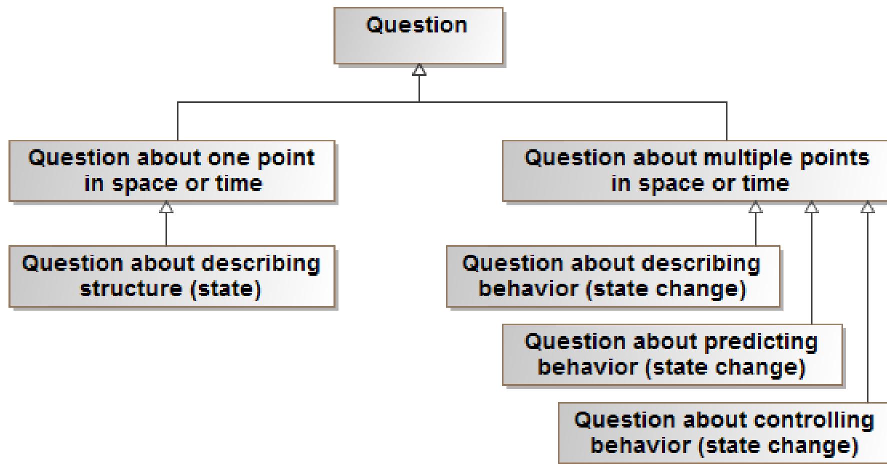


Figure 53: A Taxonomy of Questions.

For the simplest category of questions in figure 53 - questions about describing structure at a single point in time - note that answering analysis can still be quite sophisticated, for example the Operations Research analyses of optimization and statistics. For questions about multiple points in time, note that they inherit a natural sequence from [Rouse, 2009]'s levels of understanding - only after behavior can be described can it be predicted, and only after behavior can be predicted can it be controlled (also called *prescribed*). For the first category of questions about multiple points in time, questions about describing behavior, suppose that the subject of a question is a model element's property with a recorded time history containing two observations, as shown in figure 54.

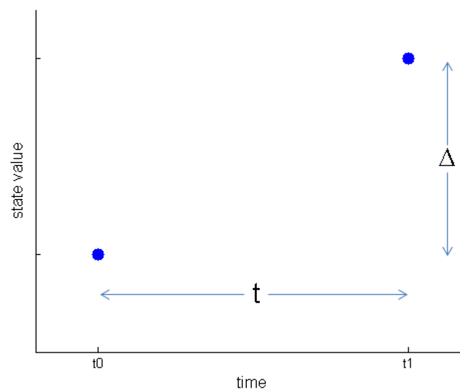


Figure 54: State Values Available at Two Points in Time.

For questions about describing behavior, analysis of the time history in figure 54 is limited to state change magnitude Δ, time interval t, Δ/t, and not much more. Δ/t

may suggest a linear-trend analysis model which can be used for prediction, but doing that requires caution given only two data points. Suppose that the subject of a question is a model element's property with a recorded time history containing many more observations, as shown in figure 55.

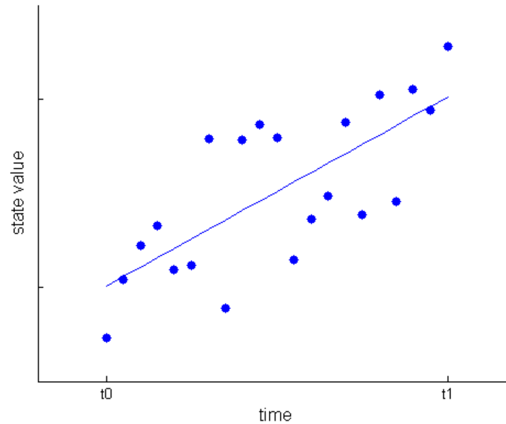


Figure 55: State Values Available at Many Points in Time, Plus a Regression Line.

For questions about describing behavior, analysis of the time history in figure 55 may again include state change magnitudes Δ, time intervals t, and Δ/t for many permutations of points which can aggregate to a descriptive analysis model for system behavior. However, enough recorded state values enable more sophisticated reduced-order analysis models. One example is a regression line (plotted in figure 55), and other options include fitting an ordinary differential equation or stochastic process model, for example a Markov Chain on a discretized state space.

Consider the second category of questions about multiple points in time in figure 53, questions about predicting behavior. Such questions generally have the form "What are consequences of making a certain change?", where change may be active intervention or passively advancing time, and consequences are subjects of questions about description. This does not mean that predictive analysis always reduces to descriptive analysis at some future time and system state; rather, predictive analysis usually concerns how to estimate the future state given present and past. An example is that analysis to answer the question "*What is the expected degree of node x after advancing time by t increments?*" may be

completely different from analysis to answer the question "*What is the degree of node x?*"

Consider the third category of questions about multiple points in time in figure 53, questions about controlling behavior. Such questions generally have the form "What changes should be made to realize certain consequences?", where consequences are subjects of questions about description, and evaluating consequences of any change can be treated as a separate question about prediction. A liability of this subclass of questions is missing semantics needed to discuss control changes and their consequences, because the context model may not include or reference any definition of *control*.

5.2 A Taxonomy of Questions about a Token-Flow Network

Figure 53's taxonomy can be extended in the context of a token-flow network. This is done for several reasons, including testing the taxonomy's usefulness by seeing how it organizes questions about well-known network properties, and also to characterize the space of well-formed questions about a token-flow network. Table 1 extends figure 53 in tabulated form.

Table 1: A Taxonomy of Questions Specialized to a Token-Flow Network. CONTROL is asterisked because the token-flow network definition does not include or reference any definition of the concept.

DESCRIBE questions about TFN Structure	Questions about Structural Statistics	- (network statistics) order, size, density, global clustering coefficient, diameter, circumference, clique number, chromatic number, arboricity - (network boolean statistics) connected, bipartite, acyclic, regular - (node statistics) degree, local clustering coefficient, centrality measures - (edge statistics) betweenness centrality
	Questions about Structural Subsets	- subgraphs including trees, strongly connected components, and graph factorizations - node subsets including cliques, independent sets, edge coverings, and edge colorings - edge subsets including node coverings (matching, assignment), node colorings (map-making), k-factorization
	↰ Questions about Routing	- walks - paths including Hamiltonian path (visit every node) and Eulerian path (visit every edge) - cycles
DESCRIBE questions about TFN Behavior	Questions about Flow Statistics	- entering and exiting nodes - across edges - from a token's perspective (e.g. time in motion, time stationary)
	↰ Questions about Flow Rates on Edges	- max flow (spans *describe* and *predict*)
	↰ Questions about Flow Rates through Network	- cycle time - throughput - capacity
	↰ Questions about Total Cost associated with Flow	- transshipment (spans *describe* and *control*) - edges off / on (including fixed-charge for use) - nodes off / on
	Questions about Waiting	- queueing - inventory, including work-in-process - from a token's perspective (e.g. time in queue)
PREDICT questions about TFN Behavior	Such questions can be written in the form "What are consequences of making a certain change?", where change may be active intervention or passively advancing time, and consequences are subjects of questions about description.	
CONTROL* questions about TFN Behavior	Such questions can be written in the form "What changes should be made to realize certain consequences?", where consequences are subjects of questions about description, and evaluating consequences of a certain change could be treated as a separate question about prediction.	

Table 1's third column is not intended to be exhaustive. The second-column categories are not disjoint; one example is that answering a question about the network structural statistic *diameter* (the longest shortest path between any pair of nodes) requires first finding those shortest paths by answering a question about routing. Many of the structural statistics and subsets in the third column are defined in section 3.1.

Predicting and controlling token-flow network behavior inherently concern change, whether given a change and predicting its consequences or given desired consequences and finding a control change to realize them. Therefore, questions about predicting and controlling token-flow network behavior can be further characterized by better understanding the ways a token-flow network can change. Prior work exists, and much is in the spirit of [Koka et al., 2006] which allows two primitive change mechanisms (creation and dissolution of ties) and then defines aggregate patterns of network evolution resulting from the two primitives (network *expansion*, *churning*, *strengthening*, and *shrinking*). This dissertation's token-flow network definition allows many more primitive change mechanisms:

- Advance time
- Add or remove a node set
- Modify a node set's state
- Add or remove an edge set
- Modify an edge set's state
- Create or destroy a token set
- Modify a token set's state

Continuing to develop the idea of network change exceeds the scope of this initial study, which focuses on semantics, categories, and patterns in questions which induce engineering analysis. However, the primitive change mechanisms just listed suggest several directions for future work, including investigating aggregate patterns of token-flow network evolution which follow from the primitive change mechanisms listed, and also expressing higher-level control concepts such allocations, plans, and policies in terms of the primitive change mechanisms, which might be done in combination with creating a definition of *control*.

5.3 A Question's Subject and Predicate Modifiers

Given the taxonomy in section 5.1, this section returns to the pattern "(Question Word) is (Question Subject) (Predicate Modifiers)?" and further characterizes a *Question Subject*, its modifiers, and *Predicate Modifiers* for each of the four leaf categories in figure 53. A syntax used to display a question's structure is Reed-Kellogg sentence diagramming [Reed and Kellogg, 1896]. A question is a sentence, and every sentence has a subject, predicate, and direct object, each possibly with multiple levels of modifiers. The question "*What is the degree of node x?*" diagrams as [2]:

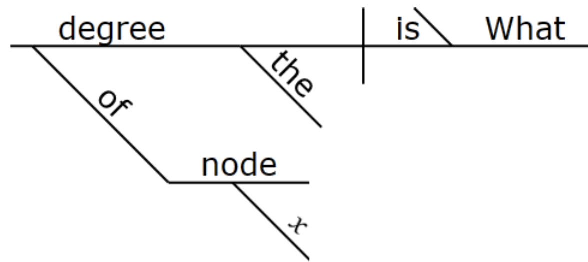


Figure 56: Example of Reed-Kellogg Sentence Diagramming.

"*What is the degree of node x?*" has subject *degree* modified by *of node x*, predicate *is*, and direct object *What*. If the question is to be machine-interpretable, note that it leaves information implicit, specifically *in network n*. Such information might be inferred from context in a conversation between humans, but in this initial study assume that all information needed to answer a question must be explicit in either the context model or the question. Therefore, given the pattern "(Question Word) is (Question Subject) (Predicate Modifiers)?", information needed to answer a question which is not in the context model must be in either the Question Subject or Predicate Modifiers, and important question semantics include types and patterns of information found in these question elements. The four subsequent sections formalize these types and patterns of information for each of the four leaf categories in figure 53's taxonomy.

[2] Created by the Reed-Kellogg Diagrammer at *http://1aiway.com/nlp4net/docs/help_reed_kellogg.aspx*, used 13march2014. Since the tool only parses complete sentences of dictionary words, identifier x is post-processed into a diagram for the sentence "What is the degree of this node?" This tool and this type of post-processing are used to create figures 56, 58, 60, 62, 63, and 65.

5.3.1 Questions about Describing Structure at a Single Point in Time

For questions about describing structure at a single point in time, the pattern "(Question Word) is (Question Subject) (Predicate Modifiers)?" expressed in the syntax of a Reed-Kellogg sentence diagram specializes to what is shown in figure 57.

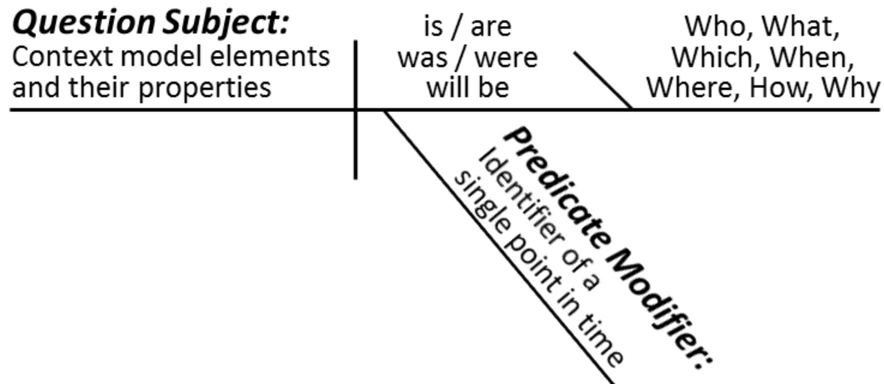


Figure 57: Pattern for a Question about a Describing Structure at a Single Point in Time.

For this category of questions, a question's subject is a subset of context model elements and their properties. Those model elements and properties need not be explicitly recorded, and if they are not then a question must define them in terms of context model elements and properties which are recorded. The predicate modifier's function concerns time. In the question "*What is the degree of node x in network n?*", the predicate modifier is omitted and understood as the present. In the question "*What was the degree of node x in network n **at time** t?*", the predicate modifier is included and specifies the single point in time t. This question is diagrammed in figure 58.

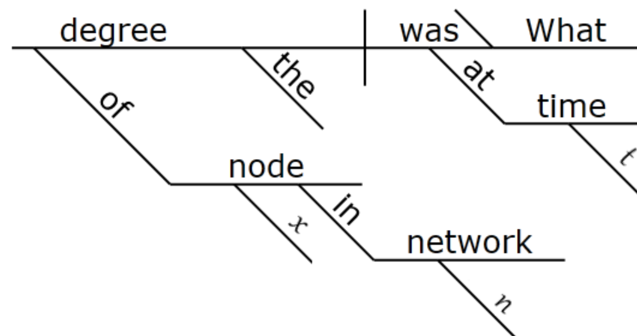


Figure 58: Reed-Kellogg Diagram for an Example of a Question about Describing Structure at a Single Point in Time.

The modifier *at time t* can be argued as either a predicate modifier as shown in figure 58 or as a nested subject modifier. The former is chosen here and for the remainder of this dissertation, that precision about time will modify *is*. A bigger complication in figure 58 is a missing definition of a node's *degree*, which is not recorded as any Node property in figure 19, and is also not defined in the question. The question should actually read "What was the **number of edges incident to** node x in network n at time t?", but for convenience a well-understood analysis semantic such as a node's degree should be allowed in place of its verbose definition. To allow this convenience yet still have the question be machine-interpretable, one solution is to have an ontology mapping certain analysis semantics back to token-flow network semantics. Network, node, and edge statistics and subsets from the first two categories in table 1 might also be recorded in this ontology, for example a network's density, diameter, global clustering coefficient, and more.

Note that for questions about describing structure at a single point in time, figure 57 is a realization of this chapter's goal - characterizing what information is needed in a question about a model such that answering analysis can be formulated.

5.3.2 Questions about Describing Behavior Spanning Multiple Points in Time

For questions about describing behavior spanning multiple points in time, the pattern "(Question Word) is (Question Subject) (Predicate Modifiers)?" expressed in the syntax of a Reed-Kellogg sentence diagram specializes to what is shown in figure 59.

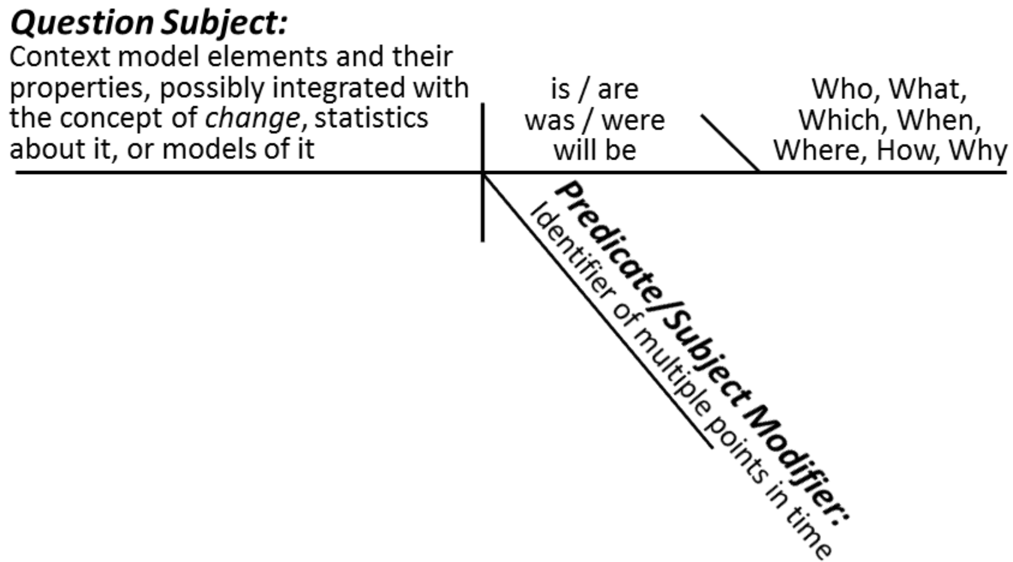


Figure 59: Pattern for a Question about Describing Behavior Spanning Multiple Points in Time.

Differences between this and the previous category are that the predicate modifier may specify multiple points in time, and that a question's subject may be a state value which can vary over those points in time. Therefore, a question's subject may be not only context model elements and properties, but also semantics for change, statistics about it, or models of it. The burden is on a questioner to ensure that a question's subject is precise with respect to time; an example of imprecision is the question "*What was the degree of node x in network n between times t_1 and t_2?*", which can only be answered with a plot of node x's degree over the continuous time interval $[t_1, t_2]$. The question might be revised to "*What was the **change in the** degree of node x in network n between times t_1 and t_2?*", diagrammed in figure 60, or in place of change may also ask about mean, variance, a random variable model , and more.

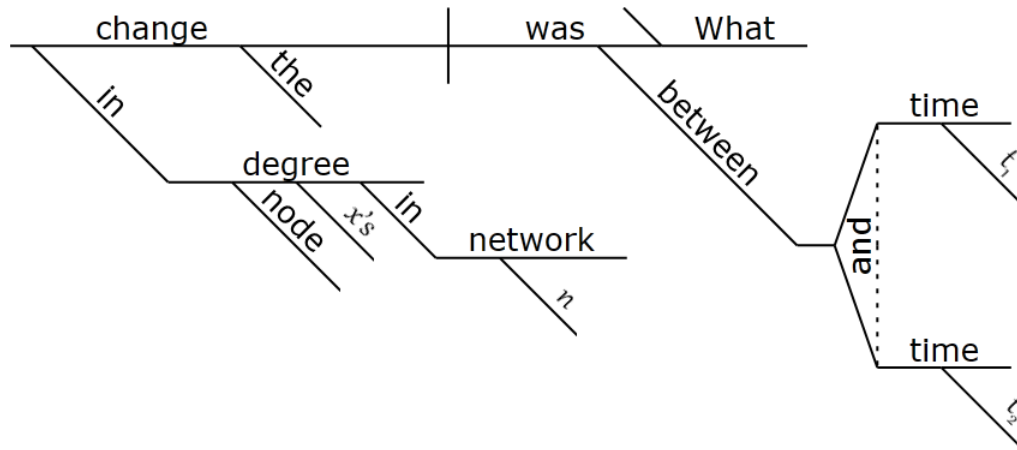


Figure 60: Reed-Kellogg Diagram for a Question about Describing Behavior Spanning Multiple Points in Time.

While the predicate modifier in this example specifies an entire interval of time points "*between times* t_1 *and* t_2", the question may be answered using only state values at the two endpoints, so just because multiple points in time are identified does not mean that they must all be used to answer the question. Also, in special cases the predicate modifier identifying multiple points in time better classifies as a subject modifier, specifically when the question's subject is time such as the length of time between two events.

. . . . . .

Questions about describing behavior are effectively questions about describing state change, and can often be answered without knowing underlying causes or dynamics of changing state, e.g. without a behavioral model. Indeed, sometimes these questions concern *finding* a behavioral model when none exists. Going forward, questions about predicting and controlling behavior will require knowledge of underlying causes and dynamics, and a new requirement in subsequent categories will be a behavioral model which must be explicit in either a context model or a question.

5.3.3 Questions about Predicting Behavior Spanning Multiple Points in Time

For questions about predicting behavior spanning multiple points in time, the pattern "(Question Word) is (Question Subject) (Predicate Modifiers)?" expressed in the syntax of a Reed-Kellogg sentence diagram specializes to what is shown in figure 61.

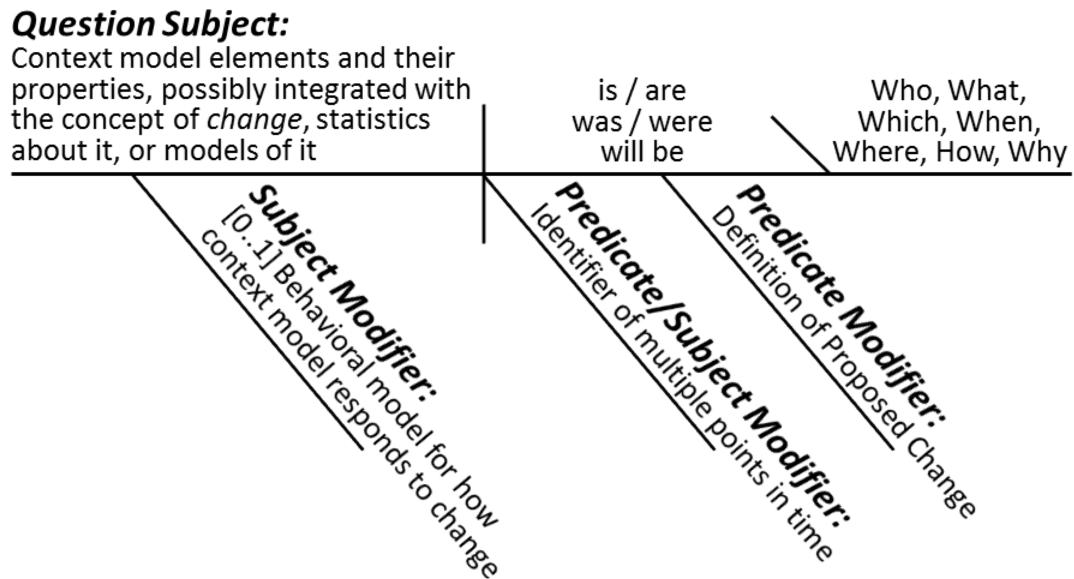


Figure 61: Pattern for a Question about Predicting Behavior Spanning Multiple Points in Time.

Section 5.1 stated that questions about predicting behavior generally have the form "What are consequences of making a certain change?", where change may be active intervention or passively advancing time, and consequences are the subjects of questions about description. Therefore, one difference between this category and the previous two is a new type of predicate modifier defining a proposed change, which is chosen as a modifier of the predicate *is* because the change is a precondition to being. Primitive change mechanisms for a token-flow network were considered briefly at the end of section 5.2 and their development was left for future work, but defining a proposed change should have similar semantics to a question's subject - it must be precise about which context model elements are to be changed in which ways at which times. If no change except advancing time, then this predicate modifier may be implicit. Another important difference between this category and the previous two is since prediction (in time) extrapolates past and present observations into an uncertain future, a behavioral model is required for how the question's subject in the context model will respond to the proposed change. An example of a question conforming to the template "What are consequences of making a certain change?" is "*What is the expected degree of node x in network n at time t_2 after making change c at time t_1?*", diagrammed in figure 62.

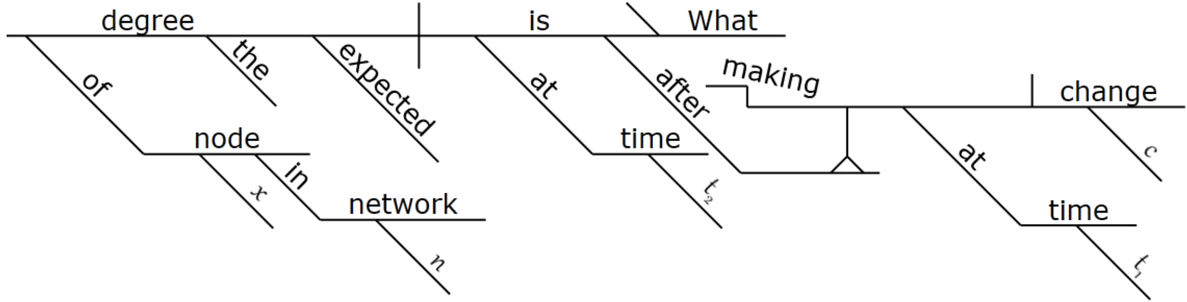


Figure 62: Reed-Kellogg Diagram for a Question about Predicting Behavior Spanning Multiple Points in Time.

The example in figure 62 is similar to the previous section's example in figure 60, with the addition of a predicate modifier defining a proposed change. What is missing is a behavioral model defining how node x's degree responds to change c as a function of time. There are several options for where such a behavioral model may reside - in the context model, in the question as an additional predicate modifier, or assumed by an analyst and returned as an answer qualification (which is defined in section 5.4). Consider an example of each:

- Suppose the needed behavioral model is in the context model. Behavior can be defined in several ways in a SysML user model, including a parametric model, an operation, an activity, and a state machine. An example of a parametric model is an ordinary differential equation describing the degree's trajectory:

  $$\dot{d} = f(d, t) + c(t)$$

  where d represents degree, $\dot{d}$ represents its first time derivative, f is an arbitrary function, and $c(t)$ is a forcing function. Note that since time is not explicit in the SysML language, t must be assigned meaning - see [Friedenthal et al., 2008, p.161] for an example.

- Suppose the needed behavioral model is in the question as a subject modifier [3]. The question diagrammed in figure 62 with an embedded behavior model is "*What*

[3]The behavioral model is chosen as a question's subject modifier because it describes how the subject responds to the proposed change. It might also be argued as a predicate modifier if it describes how being evolves in time.

is the expected degree of node x in network n at time t_2 after making change c at time t_1, ***where degree responds by exponential decay with change-dependent constant*** *c.a?*", diagrammed in figure 63.

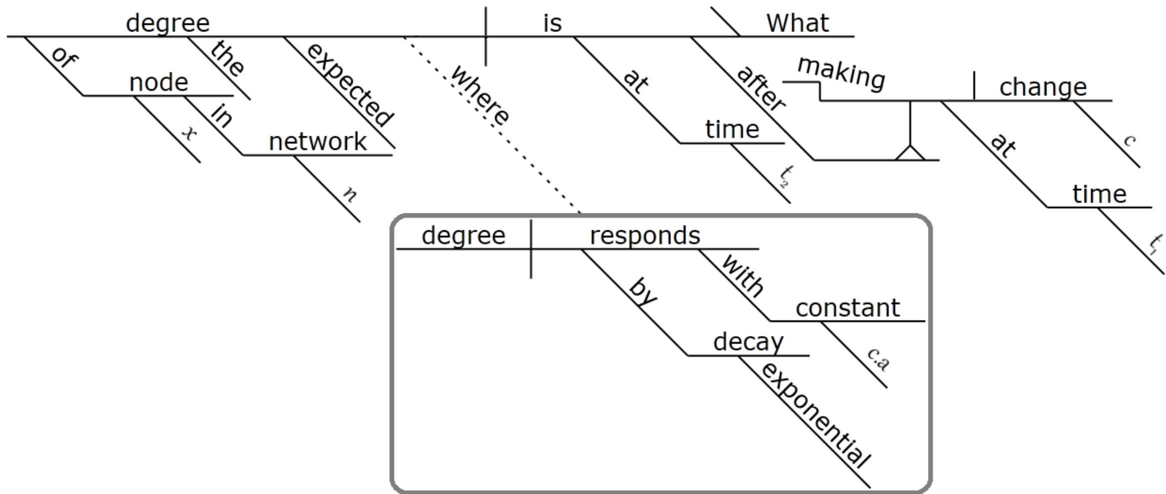


Figure 63: Reed-Kellogg Diagram for a Question about Predicting Behavior Spanning Multiple Points in Time, with an Embedded Behavioral Model.

A challenge which is not resolved here is that behavioral models stated in prose may lack the formality needed for machine-interpretability.

- Suppose the needed behavioral model is neither in a context model nor a question. Then an analyst may assume a behavioral model such as "*degree responds by exponential decay with change-dependent constant c.a*", answer the question using this assumption, and note the answer's dependence on the assumed behavioral model. Qualifications of answers are defined in section 5.4.

5.3.4 Questions about Controlling Behavior Spanning Multiple Points in Time

For questions about controlling behavior spanning multiple points in time, the pattern "(Question Word) is (Question Subject) (Predicate Modifiers)?" expressed in the syntax of a Reed-Kellogg sentence diagram specializes to is shown in figure 64.

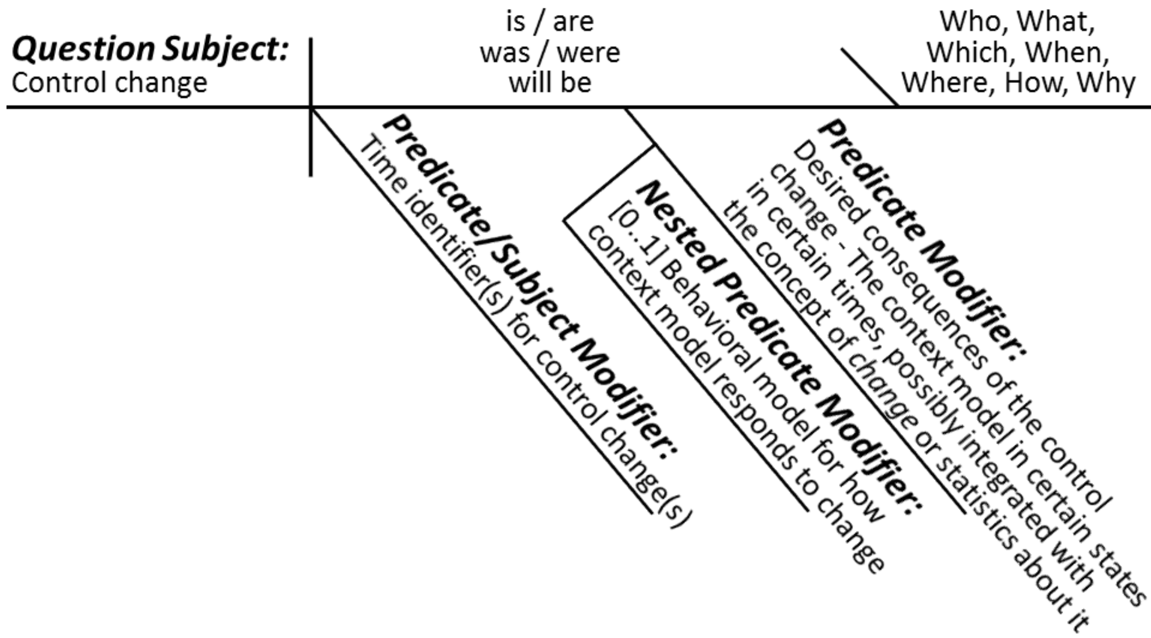


Figure 64: Pattern for a Question about Controlling Behavior Spanning Multiple Points in Time.

Section 5.1 stated that questions about controlling behavior generally have the form "What changes should be made to realize certain consequences?", where consequences are again subjects of questions about description, and evaluating consequences of a certain change could be treated as a separate question about prediction. This form inverts consequences and changes from the previous category, questions about predicting behavior, and effectively swaps the question subject and a predicate modifier between figures 61 and 64. The question's subject is a control change made to the context model, which is the desired output of an answering analysis, and as discussed in section 4.5 the concept of *control* is undefined and semantics cannot be elaborated. The desired consequences of a change - the context model in a particular state at a particular time - have swapped from a question's subject into a predicate modifier. An example of a question conforming to the template is "*What is a change at time t_1 to make d the expected degree of node x in network n at time t_2?*", diagrammed in figure 65.

Figure 65: Reed-Kellogg Diagram for a Question about Controlling Behavior Spanning Multiple Points in Time.

Similar to the previous category, predicting consequences of change requires a behavioral model. This behavioral model describes how context model elements and their properties will respond to candidate changes. As before, this behavioral model might be included in the context model, supplied in the question as a nested predicate modifier, or assumed by an analyst and returned as an answer qualification.

5.4 Analysis Models and Answers

So far this chapter has concerned the semantic content of questions and questions' links to a context model. Nothing has yet been said about analysis models and answers. Figure 66 shows an objects-and-relationships view of the process in this dissertation's methodology.

Figure 66: An Objects-and-Relationships View of The Process in this Dissertation's Methodology.

While previous sections in this chapter concern the left half of figure 66, this section concerns the right half. A question and context model are associated with zero or more answering analysis models. A question is defined as **well-formed** if, in combination with its context, at least one answering analysis can be identified. This does not imply that the resulting analysis model can be solved, only that experiments are known which can yield the answer, whether or not those experiments can actually be performed. A question which is not well-formed can become so by devising experiments or inventing analysis which can answer the question.

When a question is well-formed, the multiplicity of answering analysis models is one or more. This means that the mapping from a question and context model pair to answering analysis models is a relation, not a function. Also, an analysis model is associated with one or more answers, again a relation instead of a function - examples include an optimization analysis with multiple optima, or any analysis including randomness in the solution algorithm. These two relations are the subject of this section.

The pairing of a question and a context model may have multiple answering analysis models because there may exist multiple ways to answer the same question. For the question "*What is the degree of node x?*", even a property so simply defined can be computed in at least two ways - query node x and count the number of incident edges, or query all of a network's edges and count how many are incident to node x. For the question "*What is the expected degree of node x at time t?*", using past observations of node x's degree and choosing time series forecasting allows autoregressive models, integrated models, moving average models, combinations including ARMA or ARIMA, non-parametric methods, regression analysis, and more. In general, any question about any context might be answered by a variety of analyses, and an exogenous factor such as how much time a decision-maker can wait for the answer can influence the analysis choice. While the type of answering analysis may change, the analysis' purpose does not, which is why a question is defined independently from any answering computational process.

Given a question and context model pair, and with multiple answering analyses identified, how to choose one? Exogenous information such as "I need an answer in five

minutes" may influence the choice. In the status quo the link which converts a context model, question about it, and requirements for obtaining an answer into a choice among answering analyses is an *analyst*, included in figure 67.

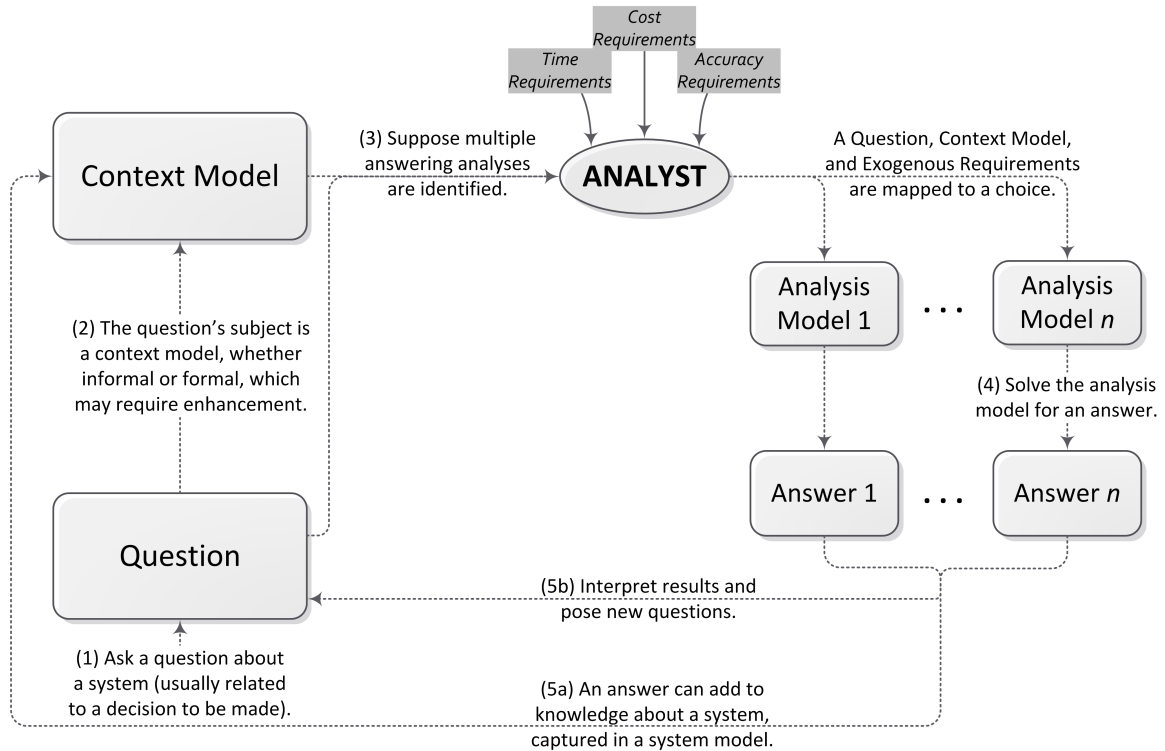


Figure 67: The Process in this Dissertation's Methodology, with Multiple Answering Analysis Models and Analyst Choice.

A tenuous assumption in figure 67 is that an analyst can appreciate the consequences of each alternative in useful terms such as time, cost, and accuracy. When the requirements and performance of an analysis model's solution algorithms are known in such terms, for example that the fastest solution algorithm has running time $\mathcal{O}(n^2)$ with n a measure of problem size, that information should be formally captured *and used as an indexing mechanism for analysis models.* This allows a choice among multiple answering analyses to be framed in terms which are useful to a decision-maker.

For an example of an accuracy requirement, suppose one way to answer a question about a context model is with integer programming analysis. If branch-and-bound is part of the solution algorithm, it maintains lower and upper bounds on the optimal solution. A possible stopping criterion is when the ratio between the bounds (the *integrality gap*) falls below a

threshold g, which may be a user-supplied accuracy requirement. However, a complication is that g may be used to select an answering analysis *and also again by the solver for a stopping criterion*, and requires careful consideration of where this information belongs.

Information absolutely essential to an analysis, meaning an analysis description is ambiguous and cannot be solved without that information, must be captured in either the question or the context model. Threshold g is not absolutely essential - without this information, branch-and-bound will continue until the entire tree is pruned and an optimal solution is exactly determined. g is integral to solution approximation, not analysis description. One way to capture such important but not essential information is to pair it (but not conflate it) with a question. Figure 68 contains this addition to figure 67, plus adding a semantic for qualified answers, plus generalizing an analyst with the function they perform [4].

[4] It is an open question if all the criteria a human analyst uses to choose among multiple answering analyses can plausibly be captured for an analysis' indexing. If they can, then a function from a context model, question about it, and requirements for obtaining the answer to a unique choice of answering analysis can be fully automated. In the present, however, it seems prudent to ask how the choice can be *narrowed* or *filtered* by indexable information, leaving a human analyst to choose among the remainder.

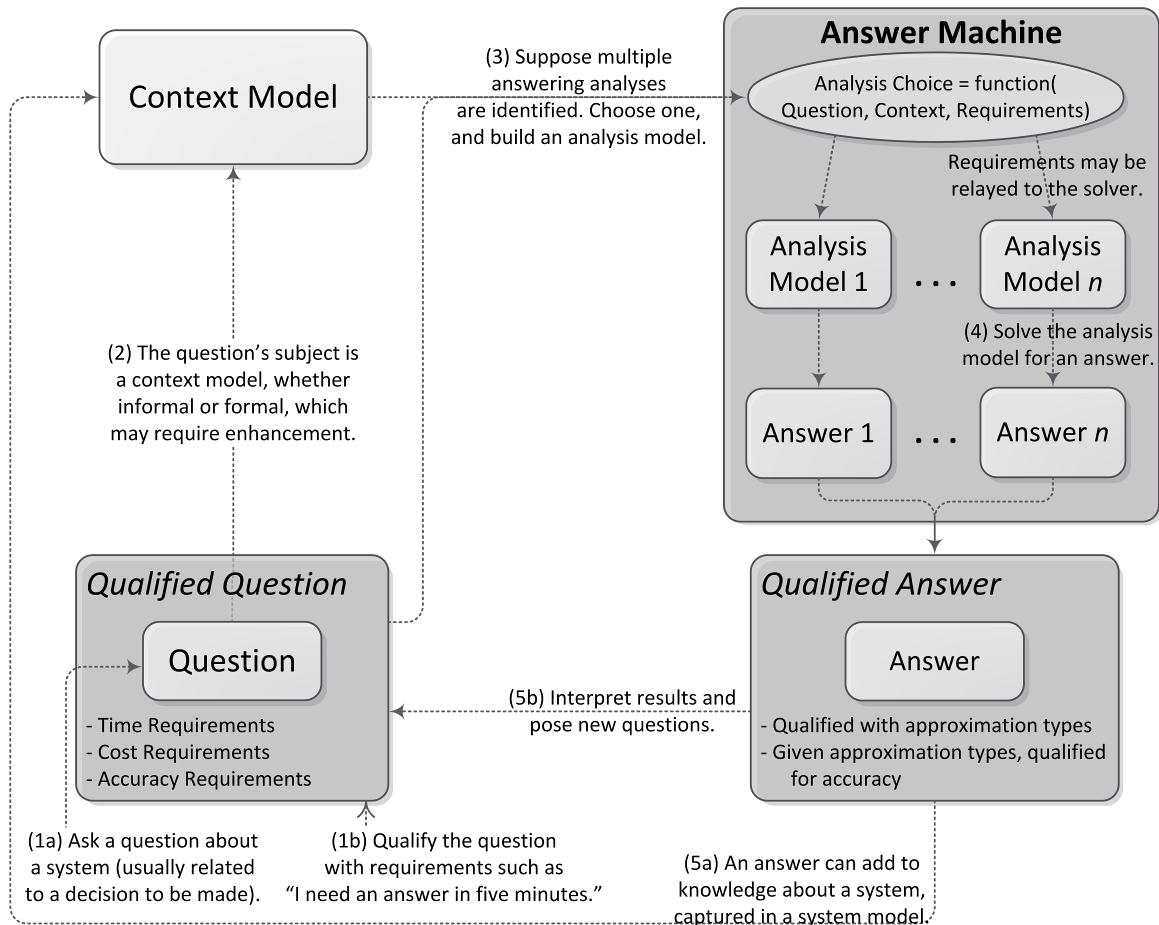


Figure 68: The Process in this Dissertation's Methodology, with Question and Answer Qualifications.

The rationale behind *Qualified Question* was just explained, and the rationale behind *Qualified Answer* is to provide information helpful or essential for interpreting the answer. Different types of analysis have different ways to quantify accuracy, for example confidence intervals for statistical estimates and integrality gaps for integer programming, and these are *qualifications of accuracy given approximation types.* What is harder to quantify is inaccuracy resulting from model assumptions such as linearity or normality, which at least can be stated and if possible quantified; these are *qualifications of approximation types.*

Combining figures 7 and 68 results in a complete illustration of the process in this dissertation's methodology, shown in figure 69.

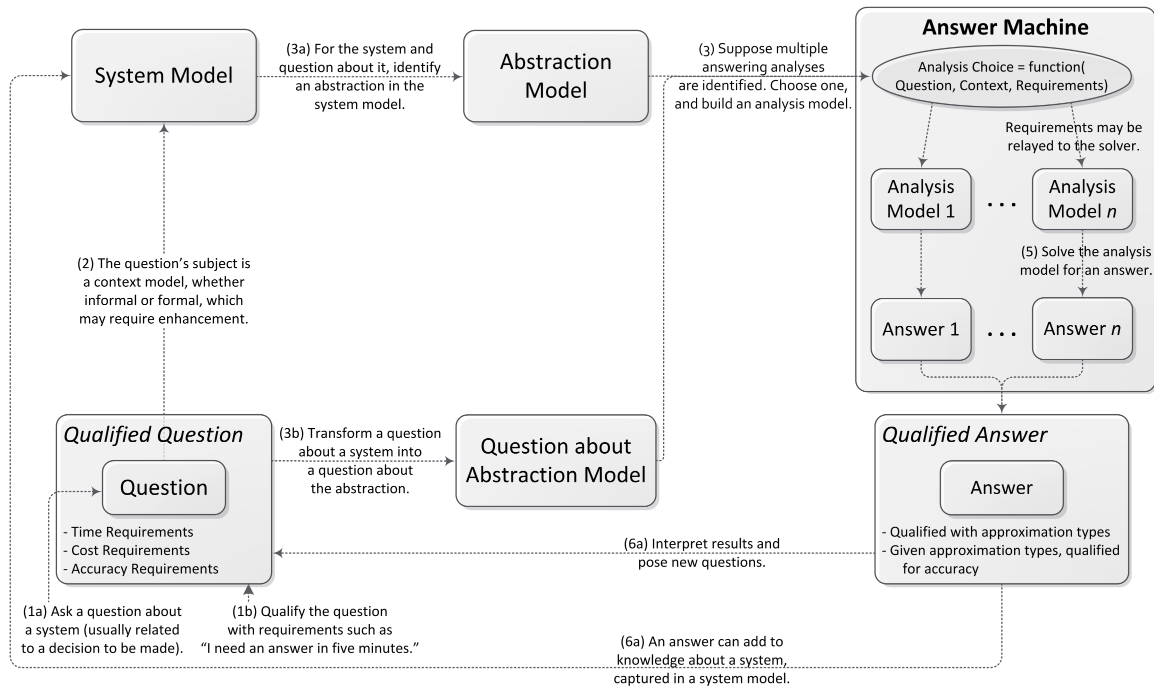


Figure 69: A Complete Illustration of the Process in this Dissertation's Methodology.

5.5 Summary

Analysis is performed for a purpose, and in this dissertation's methodology a question supplies that purpose. The semantics of a taxonomy in sections 5.1 and 5.2, patterns for each question category in section 5.3, and auxiliary information such as qualifications in section 5.4 - are first steps towards characterizing the space of well-formed questions about a model, and towards algorithmically determining if a question is well-formed.

The results of this initial study are instructive for Model-Based Systems Engineering methodology and tools. A language advocated for formal system modeling is SysML, which is paired with the query language OCL. A next step might be expressing semantics developed in this chapter using OCL's syntax, but there are obstacles. One obstacle is integration between SysML user models and conforming instance models. An integration point unsupported by many contemporary SysML tools is indexing over multiple instance models, for example as snapshots in time. The first two categories of questions in section 5.1 can be asked about a model which contains only structure and no behaviors. Even the simplest questions about describing structure at a single point in time have a predicate

modifier identifying a point in time which is not necessarily the present. If conforming instance models are snapshots of state at regular time intervals, indexing by timestamps could be useful but is not naturally supported by SysML authoring tools, instead delegated to version control tools.

Questions about describing behavior over multiple points in time have a similar issue. These questions concern describing observations of change and can often be answered without knowing the causes or dynamics of change, meaning they require no behavioral model. The obstacle is again no naturally supported way to capture structural instance models' time-varying state. The obstacle is relaxed for questions about predicting behavior because time-varying state can be described in a behavioral model. Questions about controlling behavior encounter a different obstacle - missing higher-level semantics for *control* in the SysML language and hence no defined way to express a *control change*. This category of questions must also capture an initial condition and a desired future state, which might be done with structural instance models and so require enhanced indexing of them, although in simple cases an initial condition and desired future state might be stated compactly in a question (e.g. "from value a in the present to value b in t time units"). A final obstacle is that OCL may lack syntax to express qualifications of questions, because time, cost, and accuracy requirements concern choosing and solving answering analysis rather than any immediate elements in a SysML user model or conforming instance models.

A conclusion is that question semantics developed in this chapter are partially but not fully supported by existing Model-Based Systems Engineering tools. Next steps might include introducing a new category of questions about *designing* behavior spanning multiple points in time, better characterizing the range of questions which can be asked about a model, and exploring possible alignment between question categories and analysis categories. Next steps for implementation might be expressing semantics in a more practical syntax than Reed-Kellogg sentence diagrams, and also developing integration between user models and conforming instance models of system structure. This requires an indexing scheme which might include *timestamp* for the past, *candidate* and *what-if* for the present, and *scenario* for the future.

CHAPTER VI

ABSTRACTION AND AUTOMATION METHODS

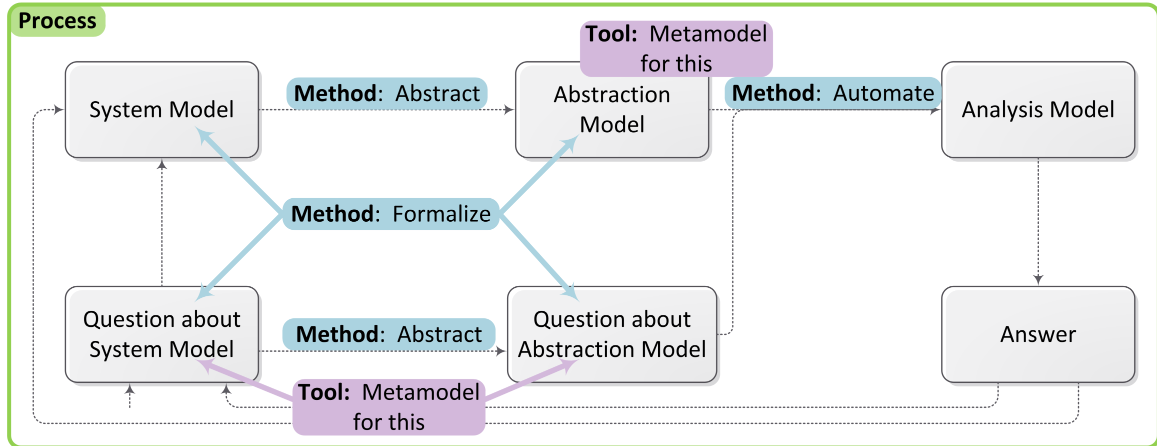


Figure 8: Process, Methods, and Tools in this Dissertation's Methodology.

The purpose of this chapter is explaining the methods in this dissertation's methodology and their application to the process. Formalization tools for an abstraction model and questions about it were the subject of the previous three chapters, and a formalization method using SysML for system models is the subject of [Friedenthal et al., 2011]. In need of explanation are the abstraction and automation methods. The tool of a token-flow network definition supports the abstraction method, but little has been said about how to actually use it. The automation method targets the building of analysis models, claimed to be mechanical and amenable to automation, and this claim must be supported.

The abstraction and automation methods each involve model-to-model transformations, which are two transformation stages needed to get from a system model and question about it to an answering analysis model. These two transformation stages, which the methodology proposes to execute quite differently, are the subject of this chapter.

6.1 *First-Stage Abstraction by Stereotype Application*

In the SysML language, formally abstracting a system model to a token-flow network model can be done by stereotype application. A *stereotype* is a SysML language element which can modify or extend the language itself. Given a SysML user model, applying token-flow network stereotypes can be thought of as overlaying a token-flow network screen over the system model, hiding certain model elements and properties and abstracting vocabulary of retained elements. Stereotype application enables defining model-to-model transformations within a system modeling environment, and also makes it easy to change levels of abstraction by clearing existing stereotypes and reapplying at the desired level of abstraction. While stereotypes and their collection in *profiles* are metamodeling elements, application to a SysML user model results in a transformed SysML user model.

[Friedenthal et al., 2011] loosely classify stereotypes into two categories:

- A source of ancillary data and rules. An example of stereotype usage in this category is to add standardized version or audit information to model elements.
- A domain-specific language in which stereotypes act more like metaclasses. Into this category fall chapter 3 and 4's token-flow network stereotypes including *Network*, *Node*, and *Edge*.

[Berner et al., 1999, p.252] classify stereotypes into four categories, which are illustrated in figure 70 and explained below.

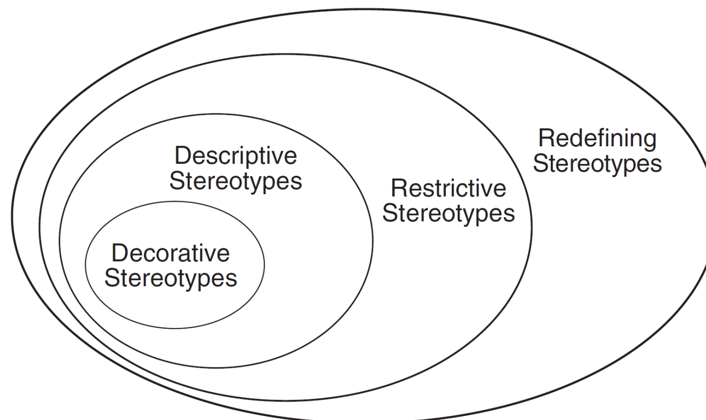


Figure 70: [Berner et al., 1999]'s Classification of UML Stereotypes.

- "A *decorative stereotype* modifies the concrete syntax of a language element and nothing else."
- "A *descriptive stereotype* extends or modifies the abstract syntax of a language element. ... The semantics of the base language remains unchanged."
- "A *restrictive stereotype* is a descriptive stereotype that additionally defines the semantics of the newly introduced element. ... A restrictive stereotype does not change the semantics of the base language - it only extends it by the semantics of the stereotype. ... Restrictive stereotypes are typically used to add missing features to some elements of a language, to strengthen weak features, or to introduce a metalanguage on top of a given language."
- "A *redefining stereotype* redefines a language element, changing its original semantics. ... With decorative, descriptive, and restrictive stereotypes, instances of the stereotype remain valid instances of the stereotyped language element. For redefining stereotypes, this is no longer true. A redefining stereotype can introduce a new language element that is no longer related to the element of the base language that it stereotypes."

Token-flow network stereotypes defined in chapters 3 and 4 fall into the third category as *restrictive*, because they extend the SysML language with new semantics.

While stereotype application is a mechanism for abstracting a system model to a token-flow network model, what about abstracting questions? In simple cases there is little to do - just replace the question's subject in the system model with its abstracted model elements or properties in the token-flow network model. When a question's subject does not have an abstract analog in the token-flow network model, however, then there are two possibilities. Either the token-flow network definition can be enhanced, or the question falls outside this dissertation's boundary because it is fundamentally not a question about describing, predicting, or controlling token-flow network structure or behavior.

6.2 Modeling Levels of Abstraction

The abstraction and automation methods are two model-to-model transformation stages needed to get from a system model and question about it to an answering analysis model. The previous section 6.1 described first-stage abstraction by stereotype application, and the next section 6.3 will describe second-stage identifying and building analysis models and automation. In preparation, this section defines an Object Management Group (OMG)

convention for object-oriented modeling layered abstraction levels, which is to label them M_3, M_2, M_1, and M_0 as illustrated in figure 71.

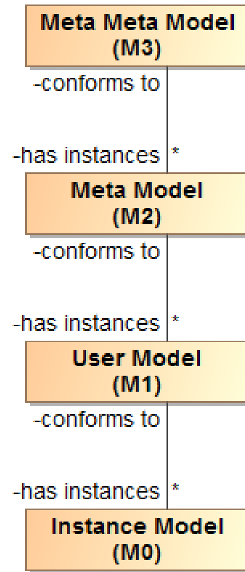


Figure 71: OMG's Layered Abstraction Levels for Object-Oriented Modeling.

Instance models live at the M_0-level, user models live at the M_1-level, metamodels live at the M_2-level, metametamodels live at the M_3-level, and infinite recursion is avoided only by authoring metametamodels in a self-defining language such as MOF [OMG MOF, 2013] or ECORE [EMF ECORE, 2014]. When a user opens a SysML authoring tool, creates a new model, and adds elements such as blocks and associations, an M_1-level user model is created which defines a schema for any conforming M_0-level instance model. An example of an M_1-level SysML user model is shown in figure 72.

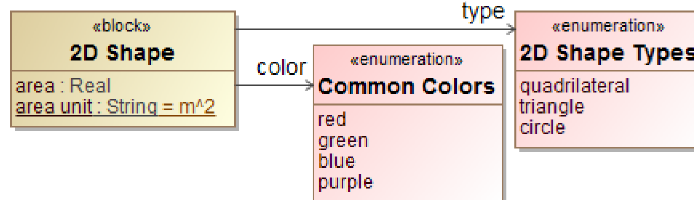
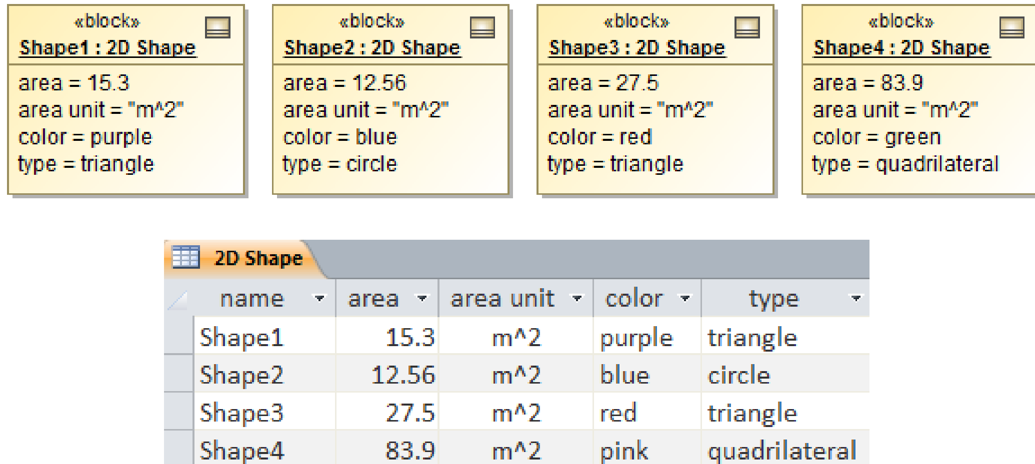


Figure 72: A Simple SysML User Model with One Block and Two Supporting Data Types.

M_0-level instance models conforming to the SysML user model in figure 72 can be captured in a variety of ways. One way is in a SysML authoring tool, which captures instance models with an underlying XMI representation. Another way better suited to large-scale

instance models is to capture them in a database whose schema is the M_1-level user model. Database options include an object-oriented database, flattening into a relational database[1], and more. Examples of an M_0-level instance model conforming to the SysML user model in figure 72 are shown in figure 73, captured in both a SysML authoring tool (top) and also in a relational database (bottom).



2D Shape

name	area	area unit	color	type
Shape1	15.3	m^2	purple	triangle
Shape2	12.56	m^2	blue	circle
Shape3	27.5	m^2	red	triangle
Shape4	83.9	m^2	pink	quadrilateral

Figure 73: Instance Models Conforming to the SysML User Model in Figure 72.

Contemporary SysML authoring tools include numerous and useful features for working with M_1-level user models, but to the best of our knowledge none have a scalable solution to build and maintain conforming M_0-level instance models. This may be a consequence of many contemporary SysML authoring tools being first and foremost UML authoring tools.

. . . . . .

A question not yet addressed is why figure 71 depicts exactly four abstraction levels and not some other number. This question is answered directly in [OMG MOF, 2013, p.8]:

> "*One of the sources of confusion in the OMG suite of standards is the perceived rigidness of a 'Four layered metamodel architecture' that is referred to in various OMG specifications. Note that key modeling concepts are Classifier and Instance*

[1] Flattening refers to creating a relational data representation of objects. An example of the conversions required is in how generalization relationships are stored in a relational database, for example *2D Shape* with subclasses *Quadrilateral*, *Triangle*, and *Circle*. Subclasses might be stored as records in a *2D Shape* table, meaning schema information is stored as data, because relational databases can only store data.

or Class and Object, and the ability to navigate from an instance to its classifier. This fundamental concept can be used to handle any number of layers."

The takeaway is that there can exist as many layers as are usefully needed. For UML structural models, four is a useful number - MOF is defined at M_3, the UML language conforms to MOF and is defined at M_2, user models of structure conform to UML and are defined at M_1, and object instances are at M_0. Note that many abstraction levels exist below M_0, including interpreted computer programs on top of compiled computer programs on top of assembly instructions on top of machine code instructions, but those levels are unimportant for designing object-oriented software or solver-ready Operations Research analysis models. Behavioral models such as UML Activities and State Machines can also be defined in four layers - MOF at M_3, UML defining *Activity* at M_2, usages at M_1, and *Activity Executions* as object instances at M_0. [OMG UML, 2011, p.327] says:

"*An activity execution represents an execution of the activity. An activity execution, as a reflective object, can support operations for managing execution, such as starting, stopping, aborting, and so on; attributes, such as how long the process has been executing or how much it costs; and links to objects, such as the performer of the execution, who to report completion to, or resources being used, and states of execution such as started, suspended, and so on.*"

While this is literally correct, activity executions as instance objects is not a useful level of abstraction when the goal is generating solver-ready Operations Research analysis models. Activity executions as instance objects is important to a solver, especially for a model with stochastic components requiring sampling, replications, and estimation, but for building the solver's input this is one abstraction level lower than important. For this dissertation's goals, behavioral models including UML Activities, State Machines, and discrete-event simulation models are effectively defined over three interesting abstraction levels rather than four. This discussion is continued with an example in section 6.3.3.

6.3 Second-Stage Analysis Model-Building and Automation

Given a token-flow network model and question about it, what does it mean to identify an answering analysis, and given the identification what does it mean to build an answering analysis model?

Identifying an answering analysis means creating a declarative specification of its formulation. This means describing which question and network model elements transform to which analysis model elements, although without any need to describe control flow for how to actually execute the transformation [2]. Given a declarative specification, building an analysis model means executing the transformation - adding imperative control flow, executing transformation rules, and outputting an analysis model. The claim that building analysis models can be mechanical and amenable to automation is supported by the fact that it is already functionally done in Model-Driven Architecture of software, for example with tools in the Eclipse Modeling Project. [3] This offers hope that Model-Based Systems Engineering (MBSE) can build upon the work of Model-Driven Architecture (MDA), with benefits such as explicit metamodels not embedded in transformations and declarative specification separated from imperative control flow whenever possible. However, it must first be considered if MDA's paradigm and tools are appropriate for this dissertation's methodology before adapting them.

An MDA paradigm for model-to-model transformation is shown in figure 74 [4].

[2]Can a transformation always be expressed declaratively? [Gardner et al., 2003] say "*In the experience of the authors, a declarative approach is useful for specifying simple transformations and for identifying relations between source and target model elements. However, many transformations are not straightforward. . . . An imperative language is preferable for the definition of complex many-to-many transformations which involve detailed model analysis.*" Therefore, identifying an answering analysis means knowing a declarative specification of its formulation *to the extent possible*, complemented by imperative specification if needed.

[3]The Eclipse Modeling Project is hosted at *http://www.eclipse.org/modeling/*, viewed 14feb2014. Its Model to Model Transformation subproject is hosted at *http://www.eclipse.org/mmt/*, and hosts three subprojects of its own - ATL, QVT Declarative, and QVT Operational.

[4]Image taken from *http://wiki.eclipse.org/ATL/Concepts*, viewed 11feb2014.

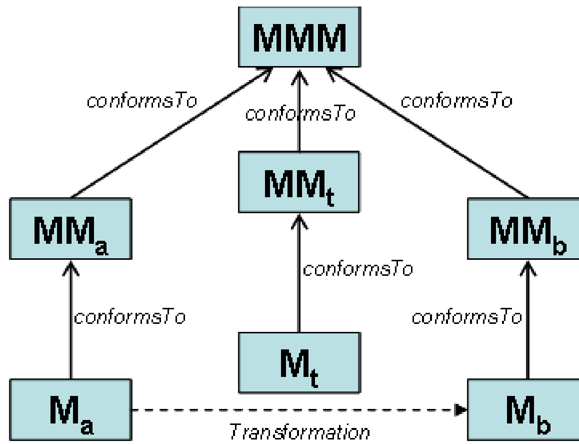


Figure 74: A Model-to-Model Transformation Paradigm.

For MDA object-oriented code-generation, levels of abstraction in figure 74 are M_3, M_2, and M_1 from top-to-bottom. MM_a is a metamodel for M_a; one example is MM_a the UML metamodel defining elements like *Class*, *Property*, and *Operation* and M_a a conforming user model of object-oriented software. MM_b is a metamodel for M_b; one example is MM_b the Java language definition and M_b a conforming Java program. M_t is a transformation instance conforming to MM_t, which defines a language for model-to-model transformations. Implicit in figure 74 is that while a transformation is *executed* to produce M_b from M_a, it is *defined* between metamodel elements in MM_a and MM_b, made explicit in figure 75.

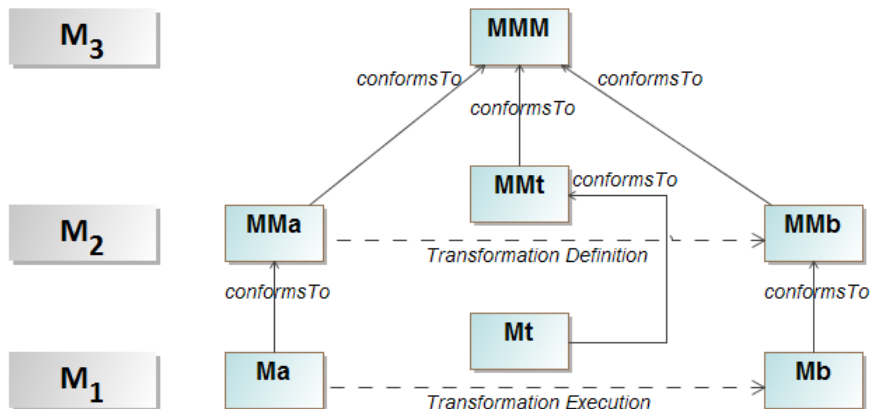


Figure 75: Model-to-Model Transformation for MDA Object-Oriented Code Generation.

Object-oriented code generation is effectively a syntax translation between an object-oriented UML model and object-oriented programming code. Note that the instance

level M_0 is not shown in figures 74 and 75 because it is unimportant to object-oriented code generation. This is a major distinction between model-to-model transformations in MDA object-oriented code generation versus MBSE analysis model generation - the former outputs M_1-level object-oriented programming code, and the latter outputs M_0-level analysis instance models. MBSE analysis model generation can follow the same model-to-model transformation paradigm as MDA object-oriented code generation, but shifted one level of abstraction lower, as illustrated in figure 76.

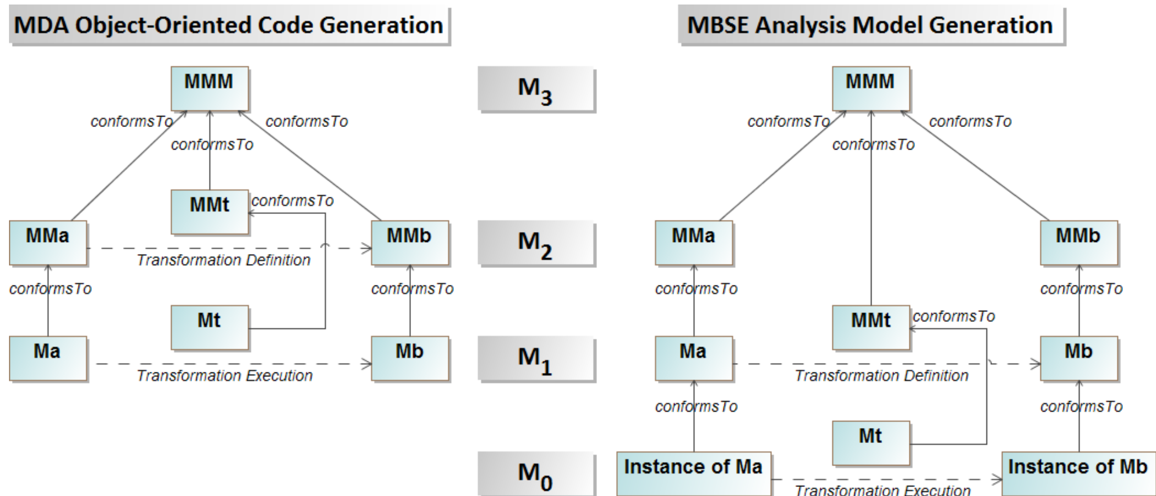


Figure 76: Model-to-Model Transformations in MDA Object-Oriented Code Generation and MBSE Analysis Model Generation Use Cases.

Given a token-flow network model and a question about it, figure 76 allows another explanation of identifying versus building an answering analysis - identifying an answering analysis means defining a model-to-model transformation at the M_1 level, and building an analysis model means executing the transformation at the M_0-level. The function of automated analysis model-building programs this dissertation's methodology is to execute semantic and syntactic transformations of token-flow network instance models at the M_0-level.

Supporting what is claimed in figure 76 requires developing the right-hand sides of transformations, to show that Operations Research analyses of discrete-event logistics systems can be defined in an object-oriented way across multiple layered abstraction levels. As indicated in the previous section, assignment of semantics to numbered abstraction

levels is relative rather than absolute. Guidelines for what follows are that the highest-level definition of analysis resides at M_2 and a solver-ready analysis model with instance model elements and data resides at M_0. This works for optimization, statistics, and discrete-event simulation analysis, although a process model subject to discrete-event simulation experiments is fundamentally a behavioral model and is effectively defined using one less level of abstraction, as previously explained.

6.3.1 Example: Optimization Analysis

AMPL (A Mathematical Programming Language) [Fourer et al., 2002] defines canonical semantics for mathematical optimization analysis, and its syntax can be translated into various vendors' languages for solution. An M_2-level definition of AMPL includes semantics shown in figure 77.

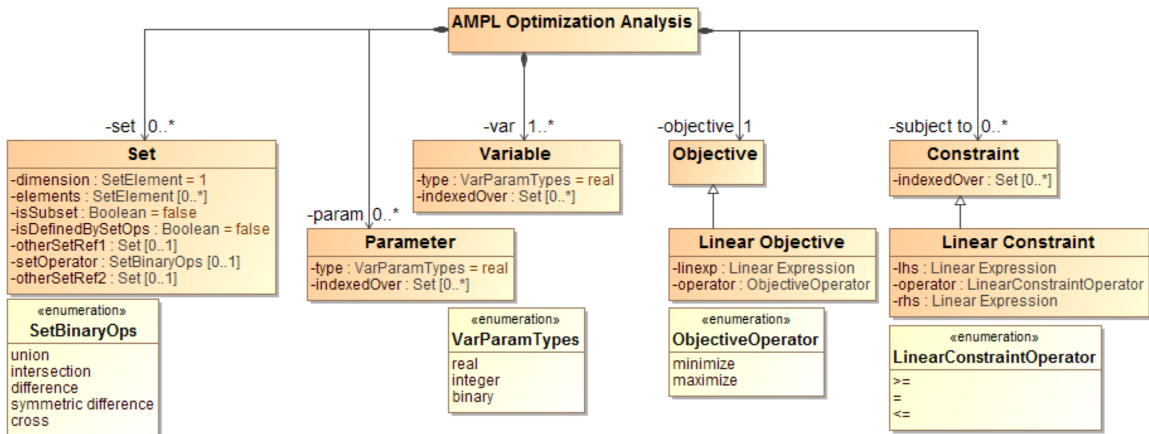


Figure 77: Definition of an AMPL Optimization Analysis Model.

Intuitive semantics of any optimization analysis are variable, objective, and constraint, which are all included in figure 77. AMPL models are fundamentally set-based, which is why Parameter, Variable, and Constraint are each indexed over any number of sets. Note that figure 77's definition is incomplete - it does not elaborate Linear Expression, and only shows an abbreviated definition of Set. Also not shown in figure 77 is that AMPL contains explicit syntax for network flow optimization analysis, including "Node" and "Arc".

Using these M_2-level semantics plus syntax defined in [Fourer et al., 2002], an M_1-level model for network flow optimization analysis is shown in model 6.1.

```
set FlowNode;
set FlowEdge within (FlowNode cross FlowNode);
set TokenType;

#Sign convention for netFlow:  Demand is positive, Supply is negative
param netFlow {FlowNode, TokenType};
param flowUnitCost {FlowEdge, TokenType};
param typeCapacity {FlowEdge, TokenType};
param grossCapacity {FlowEdge};

var flowAmount {FlowEdge, TokenType};

minimize netFlowCost:
sum {(i,j) in FlowEdge, c in TokenType}
flowUnitCost[(i,j),c] * flowAmount[(i,j),c];

subject to flowBalance {n in FlowNode, c in TokenType}:
sum {(i,n) in FlowEdge} flowAmount[(i,n),c]
   = netFlow[n,c] + sum{(n,j) in FlowEdge} flowAmount[(n,j),c];

subject to flowBounds {(i,j) in FlowEdge, c in TokenType}:
0 <= flowAmount[(i,j),c] <= typeCapacity[(i,j),c];

subject to edgeGrossCapacity {(i,j) in FlowEdge}:
sum {c in TokenType} flowAmount[(i,j),c] <= grossCapacity[(i,j)];
```

Model 6.1: An M_1-level network flow optimization analysis model in the AMPL language.

Model 6.1 contains three usages [5] of the M_2-level *Set* (named FlowNode, FlowEdge, and TokenType), four usages of *Parameter*, one usage of *Variable*, one usage of *Objective*, and three usages of *Constraint* (named flowBalance, flowBounds, and edgeGrossCapacity). An experienced optimization analyst will recognize that this model is incomplete without data. "Data" is in fact a conforming M_0-level instance model, shown in model 6.2, and in AMPL can even be placed in the same input file as the M_1-level user model.

[5] A vocabulary convention used here is that instantiation of M_2-level elements in M_1-level user models are *usages*, and instantiation of M_1-level elements in M_0-level instance models are *instances*. The two words are functionally equivalent.

```
set FlowNode := Node1 Node2 ;
set FlowEdge := (Node1, Node2) ;
set TokenType := sku001 sku002 ;

param netFlow: Node1  Node2 :=
sku001          -20     20
sku002          -30     30;

param flowUnitCost: (Node1, Node2) :=
sku001          15
sku002          12;

param typeCapacity: (Node1, Node2) :=
sku001          20
sku002          30;

param grossCapacity :=
(Node1, Node2)     50;
```

Model 6.2: An M_0-level AMPL instance model conforming to M_1-level Model 6.1.

"Data" is misleading if it only suggests literal numbers or strings; model 6.2 also instantiates sets. In AMPL, the only usages which need be instantiated are sets and parameters, from which all instances follow for all *Variable* usages, the *Objective* usage, and all *Constraint* usages. In the network flow optimization example in model 6.1, *Set* instances correspond to network elements, meaning a solver-ready optimization analysis instance model follows given a network instance model [6].

Now that an AMPL optimization analysis model has been defined at the M_2, M_1, and M_0 levels, revisit the claim illustrated in figure 76 that MBSE analysis model generation can follow the same model-to-model transformation paradigm as MDA code generation but shifted one abstraction level lower. Network flow optimization analysis may be identified as an answering analysis for various questions about a token-flow network. "Identified" implies a declarative specification of an analysis model at the M_1 level, and that declarative specification is exactly the content of model 6.1 (declarative "rules" include that the cross product of all FlowEdge instances and TokenType instances maps to a set of Variable instances, and that the cross product of all FlowNode instances and TokenType instances maps to a set of *flowBalance* Constraints). This M_1-level specification of the analysis can be

[6]This also suggests an interesting analogy between a mathematical *Set* and SysML's *Block*. An unexplored tangent is if *set-based* is a mathematical analog for *object-oriented*.

written once and infinitely reused, and a model-to-model transformation to automatically build it would be a poor investment. It is the M_0-level portion in model 6.2 which may be repeatedly rebuilt, and its content follows from a syntactically transformed token-flow network instance model [7]. Therefore, an automated analysis model-building program which may be valuable here is defined at the M_1-level and executed at the M_0-level.

6.3.2 Example: Statistical Regression Analysis

A definition of statistical analysis is inferred from [Kiefer, 1987] and illustrated in figure 78.

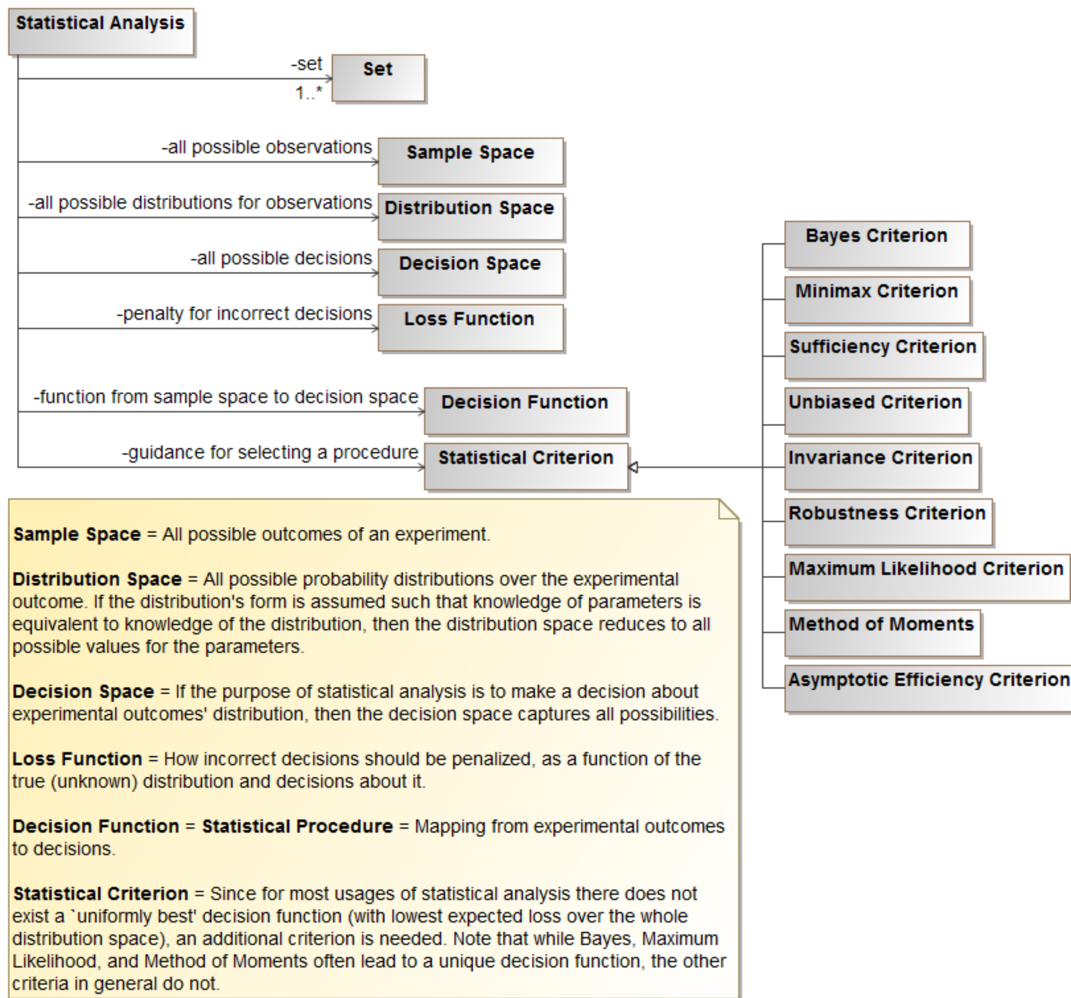


Figure 78: Definition of a Statistical Analysis Model.

[7] Executing a model-to-model transformation may only involve syntax translation as in this example, but in other cases it can involve semantic manipulation of the network instance. Semantic manipulations may include overriding nodes' *consumption* and *production* values with ones at specific nodes for shortest-path analysis, or zeroing out FlowEdge unit costs on all FlowEdges except a newly-created one which connects the target back to the source for max-flow analysis.

Given the M_2-level definition in figure 78, one way in which M_1-level usages of *Statistical Analysis* follow is by refining structure of the *Decision Space*, for example the usages of point, interval, and region estimation, hypothesis testing, regression analysis, multiple decisions, and ranking. Consider the M_1-level usage of multiple linear regression analysis, an analysis model reproduced from section 1.1:

$$
\begin{aligned}
&y = X\beta + \epsilon \qquad \textbf{OR} \qquad && y_i = \beta_1 x_{i,1} + \ldots + \beta_p x_{i,p} + \epsilon_i \qquad i = 1, \ldots, n \\
&\epsilon \sim N\left(0, \sigma^2 I\right) && \epsilon_i \sim N(0, \sigma^2) \qquad (1)
\end{aligned}
$$

Equation 1's analysis model leaves a large amount of knowledge implicit, including the semantics Predictor, Response, Parameter, Observation, and Decision. Figure 79 makes these semantics explicit, and is a partial M_1-level definition of multiple linear regression analysis, "partial" because while it shows usages of M_2-level *Set* and *Loss Function*, it does not show required usages of M_2-level *Sample Space*, *Distribution Space*, *Decision Space*, and *Decision Function*.

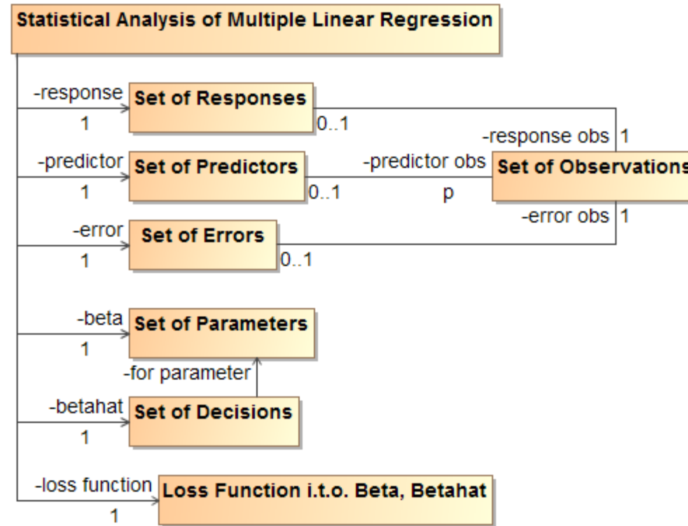


Figure 79: An M_1-Level Partial Definition of Multiple Linear Regression Analysis.

For multiple linear regression analysis, consider the simplest interesting example of estimating β with σ^2 known. The missing usages in figure 79 are:

- An M_1-level usage of **Sample Space** is all possible observations of the p predictors and one response (multiple responses induce multiple independent analyses). M_0-level

instances follow given instances of the Predictor and Response sets and associated Observation sets, e.g. given p and n in $\mathbb{R}_{n,p+1}$ [8].

- An M_1-level usage of **Distribution Space** is all possible conditional distributions of the response given the predictors: $y \sim N\left(X\beta, \sigma^2 I\right)$. In this case knowledge of the distribution reduces to knowledge of the parameters β. M_0-level instances follow given an instance of the Parameter set, e.g. given the p in $\mathbb{R}_p$.

- Decisions $\hat{\beta}$ are made for the parameters β, and an M_1-level usage of **Decision Space** is all possible values for decisions. M_0-level instances follow given an instance of the Decision set, e.g. given the p in $\mathbb{R}_p$.

- An M_1-level usage of **Loss Function** is a penalty for incorrect decisions expressed in terms of decision $\hat{\beta}$ and the true (unknown) β. Possibilities include (stated generally for decision d and true distribution θ) squared-error loss $L = (\theta - d)^2$ integral to the least-squares solution algorithm, 0-1 loss $L = \mathbb{I}_{\{\theta = d\}}$, and absolute loss $L = |\theta - d|$. Here the squared-error loss function is $L\left(\beta, \hat{\beta}\right) = \sum_{i=1}^{n}\left(\left(\sum_{j=1}^{p}\beta_j x_{i,j}\right) - \left(\sum_{j=1}^{p}\hat{\beta}_j x_{i,j}\right)\right)^2 = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$. M_0-level instances follow given instances of the Predictor and Response sets, associated Observation sets, and the Decision set.

- An M_1-level usage of **Decision Function** is a mapping from observations of predictors and response to decisions. The least-squares solution and also maximum likelihood solution is $\hat{\beta} = \left(X^T X\right)^{-1} X^T y$, assuming $\left(X^T X\right)^{-1}$ exists. M_0-level instances follow given instances of the Predictor and Response sets and associated Observation sets.

6.3.3 Example: Discrete-Event Simulation Analysis

Popular languages for discrete-event simulation analysis include AnyLogic, Arena (SIMAN), SimEvents, Simio, Tecnomatix, and more. It is argued here that none of these languages, at least as exposed to a user, defines discrete-event simulation analysis in a general sense. [Schruben and Yücesan, 1993] identify three discrete-event modeling paradigms or worldviews - *process-interaction*, *activity-scanning*, and *event-scheduling*. [Miller et al., 2004] identifies those three as *process-oriented*, *activity-oriented*, and *event-oriented*, and adds *state-oriented* with an illustration in figure 80.

[8] Restrictions on $\mathbb{R}$ might change the analysis type. Bounded or discrete response y, for example, may require a Generalized Linear Model.

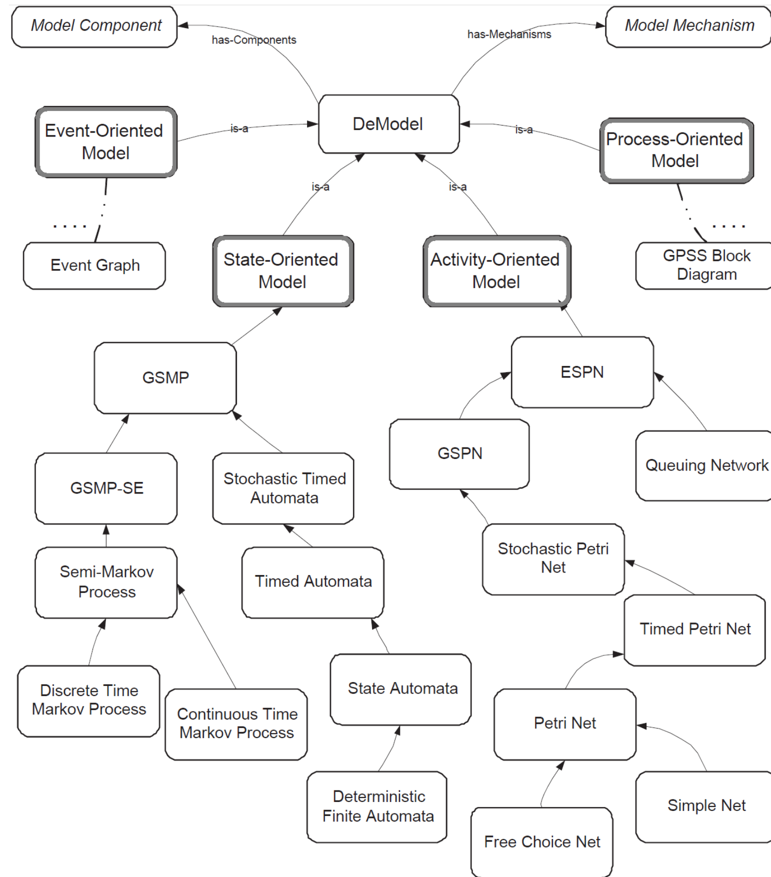


Figure 80: [Miller et al., 2004]'s Four Defining Paradigms for Discrete-Event Modeling.

[Miller et al., 2004] propose M_2-level semantics for each paradigm, and also shared fundamental concepts:

- *State-Oriented* and *Event-Oriented*: StateSpace, EventSet, TimeSet, TransitionFunction, ClockFunction, and InitialState.
- *Activity-Oriented*: PlaceSet, ActivitySet, TimeSet, IncidenceFunction, InitialMarking, and ClockFunction (where ClockFunction has a slightly different meaning than in the state-oriented paradigm).
- *Process-Oriented*: ProcessSet, ProcessStateSet, TimeSet, TransitionFunction, and ClockFunction.
- *Shared Fundamental Concepts*: State, Event, Time, Transition, Activity, Place, Token, Entity, Process, Resource, Color, and Clock.

Semantics overlap, and also may be mapped from one paradigm into another. This is true of the SIMAN language; while users create process-interaction models in block diagrams,

internally a solver follows an event-scheduling paradigm. [Pegden et al., 1995, p.579] explain:

> "*Nearly all modern discrete simulation systems are internally implemented using a discrete-event world view. ... In many respects, the discrete-event world view is a much simpler framework for modeling than the process-interaction world view. ... Unfortunately, in most cases, the effort actually required to develop a model with this view is significantly greater. The reason for this is that the logic you must develop to define each event in the model is often complex and difficult to code.*"

The existence of four different paradigms or worldviews suggests that discrete-event simulation analysis does not enjoy the same conceptual simplicity as optimization analysis. [Zeigler, 1976] proposed DEVS (Discrete Event System Specification) as a unifying framework, defined as an extension of a finite state automaton. [Schruben and Yücesan, 1993, p.267] comment "*Zeigler's formalism provides conceptual tools for model specification; however, it has yet to be developed into a practical guide for model implementation.*"

If the highest-level definition of an analysis resides at M_2, then this is where definitions of the *process-interaction*, *activity-scanning*, *event-scheduling*, and *state-oriented* paradigms belong. A four-layer architecture might define *ProcessSet* at M_2, usages at M_1 (a set of business processes, a set of engineering processes, a set of accounting processes, etc.), and populate the set with instances at M_0. In the status quo, however, neither the SysML Activity definition nor process-oriented discrete-event simulation languages make an M_2/M_1 distinction, and for a solver-ready analysis model the SysML Activity metamodel sits one level above instantiation (evidence supporting this is the existence of tools to execute Activity Diagrams). With or without an M_2/M_1 distinction, this section argues that SysML Activities, SysML State Machines, and behavioral models in many process-oriented discrete-event simulation languages can be defined in an object-oriented way across multiple layered abstraction levels, and therefore supports what is claimed in figure 76 - that MBSE analysis model generation can reuse MDA's model-to-model transformation paradigm, but shifted one abstraction level lower than object-oriented code generation.

For discrete-event behavioral models and simulation analysis, an excerpt from an M_1-level definition of the SIMAN language is inferred from [Pegden et al., 1995] and shown in figure 81.

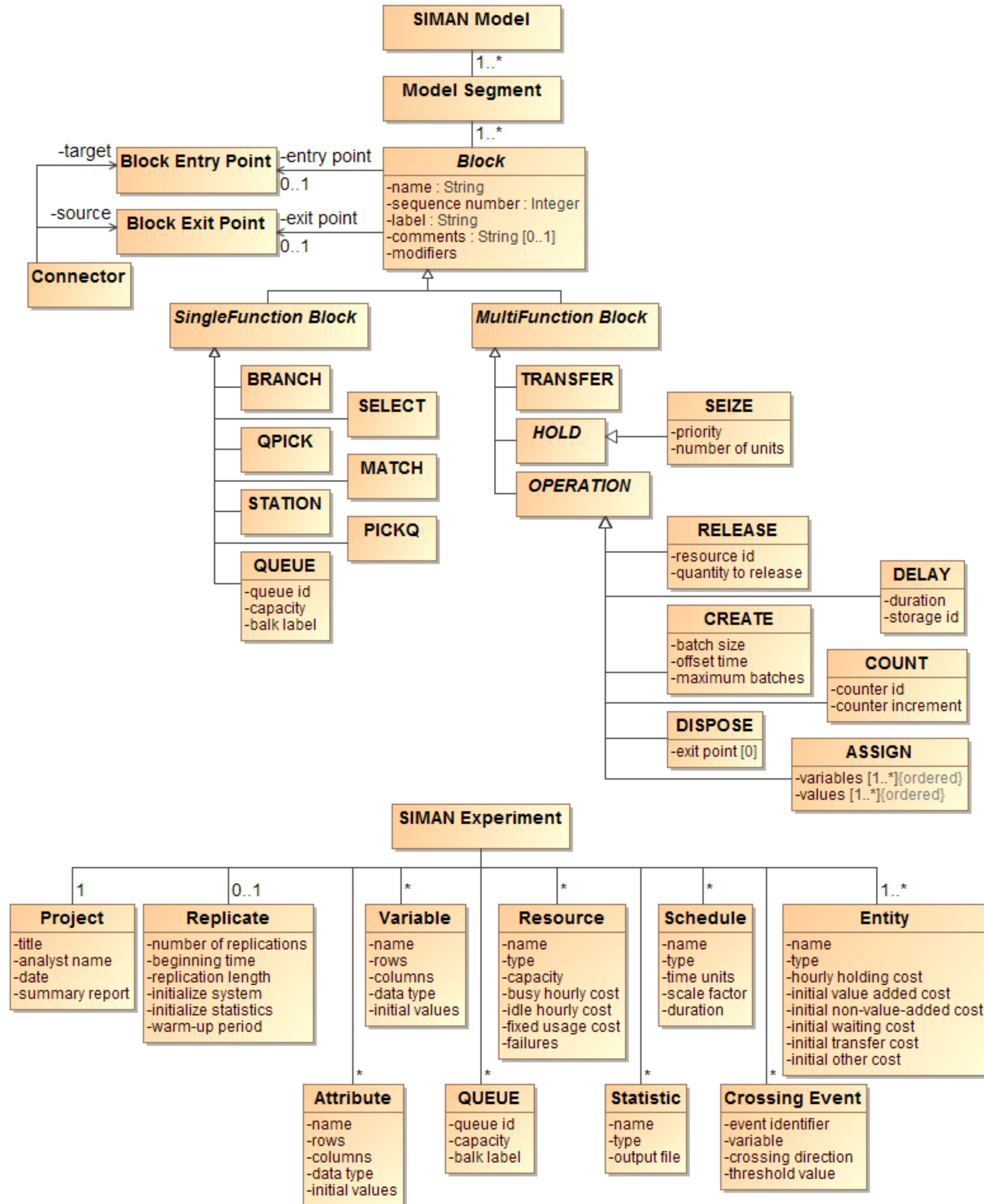
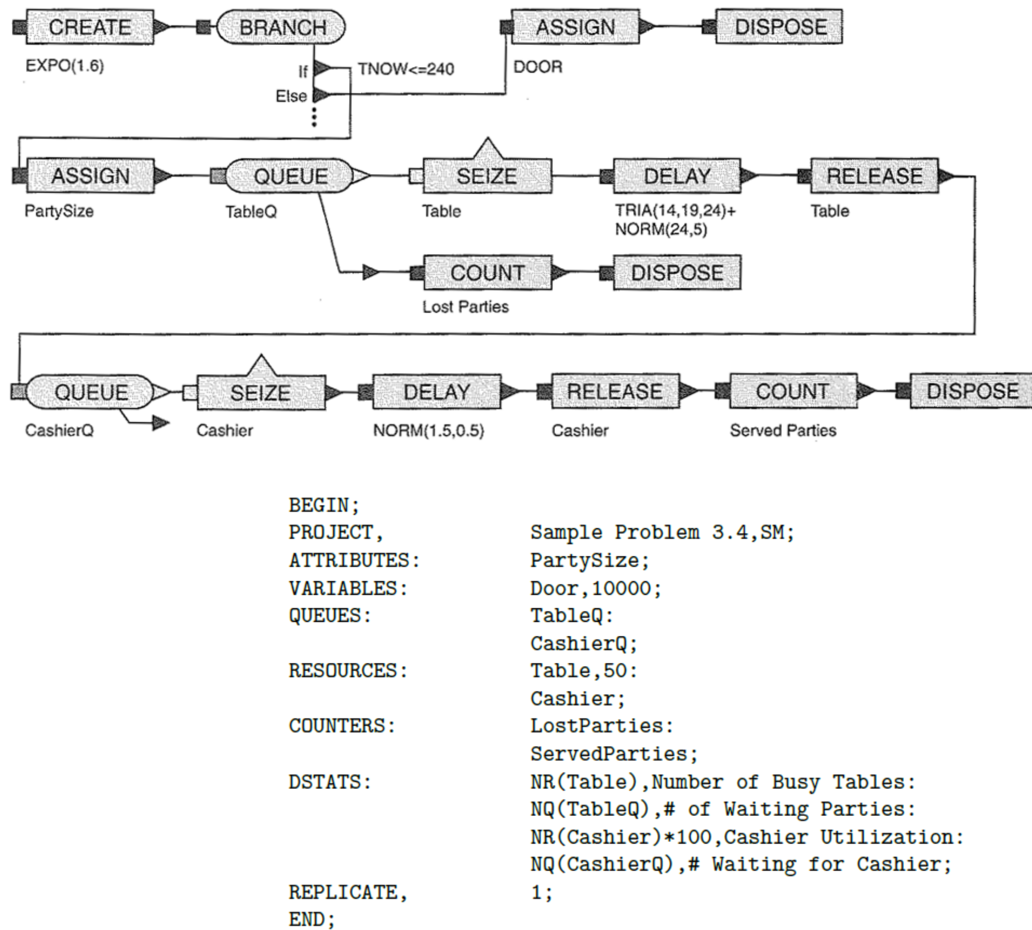


Figure 81: An M_1-Level Partial Definition of the SIMAN Language.

Figure 81 shows an important difference between a SysML Activity and a process-oriented discrete-event simulation analysis model - the latter defines both a model and experiments on that model. Figure 81 also shows semantics but not syntax, meaning hidden

are constraints for instantiating model elements. One such constraint is that a user cannot directly instantiate *Entity*, which can only be done indirectly by other model elements. Similar syntax constraints for instantiating model elements existed with the AMPL language in section 6.3.1.

An M_0-level instance of both a *SIMAN Model* and a *SIMAN Experiment* which conform to figure 81's semantics are shown in figure 82 [Pegden et al., 1995, p.116].




```
BEGIN;
PROJECT,          Sample Problem 3.4,SM;
ATTRIBUTES:       PartySize;
VARIABLES:        Door,10000;
QUEUES:           TableQ:
                  CashierQ;
RESOURCES:        Table,50:
                  Cashier;
COUNTERS:         LostParties:
                  ServedParties;
DSTATS:           NR(Table),Number of Busy Tables:
                  NQ(TableQ),# of Waiting Parties:
                  NR(Cashier)*100,Cashier Utilization:
                  NQ(CashierQ),# Waiting for Cashier;
REPLICATE,        1;
END;
```

Figure 82: A Solver-Ready Instance of a SIMAN Model and a SIMAN Experiment.

The analysis model in figure 82 is solver-ready and regarded here as M_0 instance-level. There exist lower abstraction levels including UML *Activity Executions*, which are important to a solver especially for a model with stochastic components requiring sampling, replications, and estimation, but for building the solver's input this is one abstraction level lower than important.

6.4 Model-to-Model Transformation Tools

This section considers tools which can execute a model-to-model transformation at the M_0 level. Within a SysML user model is the model itself (an XML document conforming to a standardized schema) and its human interface (diagrams). Because of the underlying XML representation, model-to-model transformations operating on a SysML user model can be written in any programming language with XML parsing capabilities. Suppose that for several questions about an instance model conforming to a SysML user model, ad-hoc transformation programs to build analysis models are written in general purpose languages such as Java or C++. While best practices for MBSE analysis model generation are not yet mature, this is not one. Output analysis models conform to metamodels themselves, which should be explicit, formal, and machine-interpretable. Further, methodology for model-to-model transformation may be common to all such programs, providing an opportunity to express transformations declaratively and abstract away control flow.

The software community long ago realized that model-to-model transformations have much in common, and abstracted commonalities into the paradigm previously shown in figure 74. The genesis of the paradigm was an OMG request for proposals to standardize mappings between models in MOF-defined languages. The request for proposals included the following mandatory requirements [OMG QVT RFP, 2002, p.23-24]:

1. "*Proposals shall define a language for querying models. The query language shall facilitate ad-hoc queries for selection and filtering of model elements, as well as for the selection of model elements that are the source of a transformation.*"
2. "*Proposals shall define a language for transformation definitions. Transformation definitions shall describe relationships between a source MOF metamodel S, and a target MOF metamodel T, which can be used to generate a target model instance conforming to T from a source model instance conforming to S. The source and target metamodels may be the same metamodel.*"
4. "*The transformation definition language shall be capable of expressing all information required to generate a target model from a source model automatically.*"
6. "*The transformation definition language shall be declarative in order to support transformation execution with the following characteristic: Incremental changes in a source model may be transformed into changes in a target model immediately.*"

As a result, today the Eclipse MMT project [9] contains several tools following figure 74's paradigm including ATL and Operational QVT. Their transformation languages share overlapping semantics including what ATL calls a *rule* and Operational QVT calls a *mapping*; both offer the hope of a pre-existing tool which can be re-purposed for MBSE analysis model-building.

At the time of writing, however, two major obstacles to these tools are a software bias and immaturity. While figure 74's paradigm should apply to any consecutive pair of abstraction levels, in reality the top box being a single self-defining language (like MOF or ECORE) suggests an M_3, M_2, and M_1 assumption consistent with MDA object-oriented code-generation. At the time of writing ATL can only accommodate MM_a in the ECORE metamodeling language, requiring unnatural conversion for M_1-level SysML user models. ATL can also only accommodate M_a and M_b in XML Metadata Interchange (XMI) format, requiring unnatural conversion for instance models in databases, and requiring second-stage model-to-text syntax translation if desired output is analysis model code. Despite these limitations, the existence of these tools is promising, and these objections only concern bias and capabilities and not the underlying model-to-model transformation paradigm.

Other approaches are also being explored for automated analysis model-building. [Sprock and McGinnis, 2014] leverages a stable network definition and software engineering design patterns and is especially useful for outputting analysis models containing a rebuilt network instance, a category which includes discrete-event simulation.

6.5 Summary

The purpose of this chapter is to explain the methods in this dissertation's methodology and their application to the process, specifically the abstraction and automation methods. For abstraction, UML stereotype application is proposed to formally abstract system models to token-flow network models. For automation, a model-to-model transformation paradigm and tools already exist in Model-Driven Architecture of software, and an original contribution follows from considering if and how these tools can be re-purposed for MBSE

[9] *http://www.eclipse.org/mmt/*, viewed 20feb2014.

analysis model generation. Section 6.3 contrasted model-to-model transformations in the two use cases, argued that they share a common transformation paradigm executed at different relative levels of abstraction, and supported the argument by showing how Operations Research analyses can be defined in an object-oriented way across multiple layered abstraction levels. The result is identifying one feasible way to automatically build analysis models, which should not exclude other possibilities.

Beyond this point, the methodology is developed no further. The next chapter contains two full-length examples of executing the methodology, of executing the process using methods and tools described in this and preceding chapters.

CHAPTER VII

EXAMPLES

This chapter contains two full-length examples of executing the methodology, of executing the process using methods and tools described in preceding chapters. The most complete illustration of the methodology's process is in figure 69, which shows all of the elements demonstrated in each example.

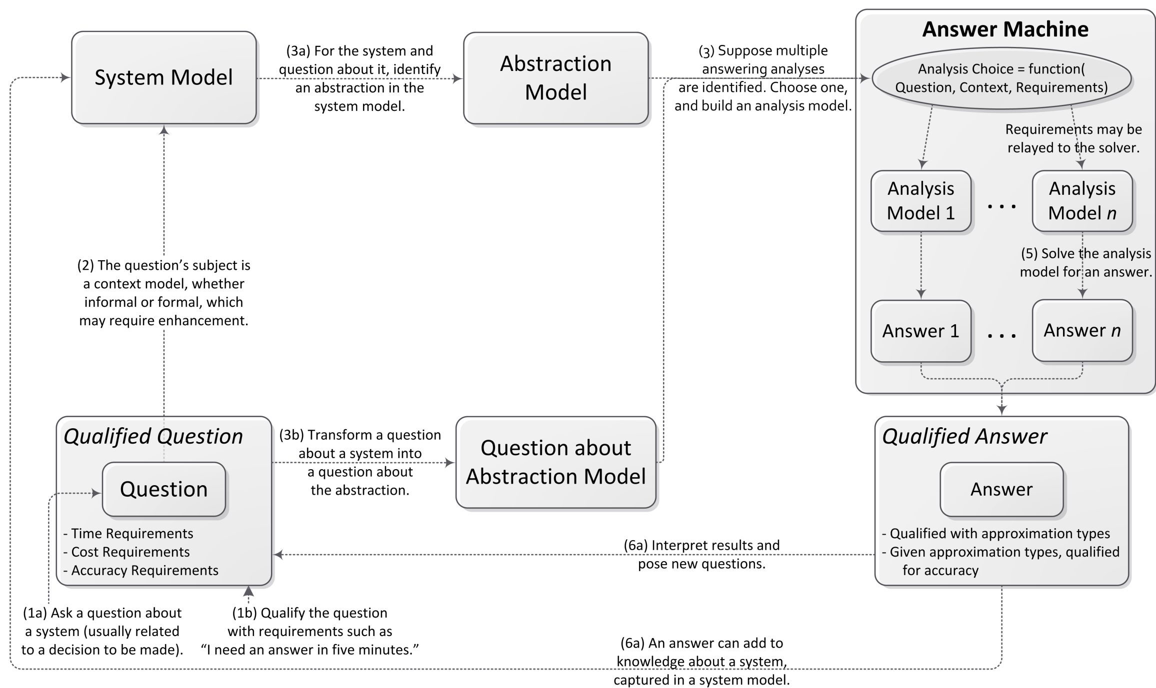


Figure 69: A Complete Illustration of the Process in this Dissertation's Methodology.

Basic elements demonstrated in each example are a formal system model, question about the system model, a formal abstraction model, question about the abstraction model, and an answering analysis model. Additional elements demonstrated are question qualifications, answer qualifications, and a choice among multiple answering analyses.

7.1 *Example using Basic Structural Semantics*

System Model:

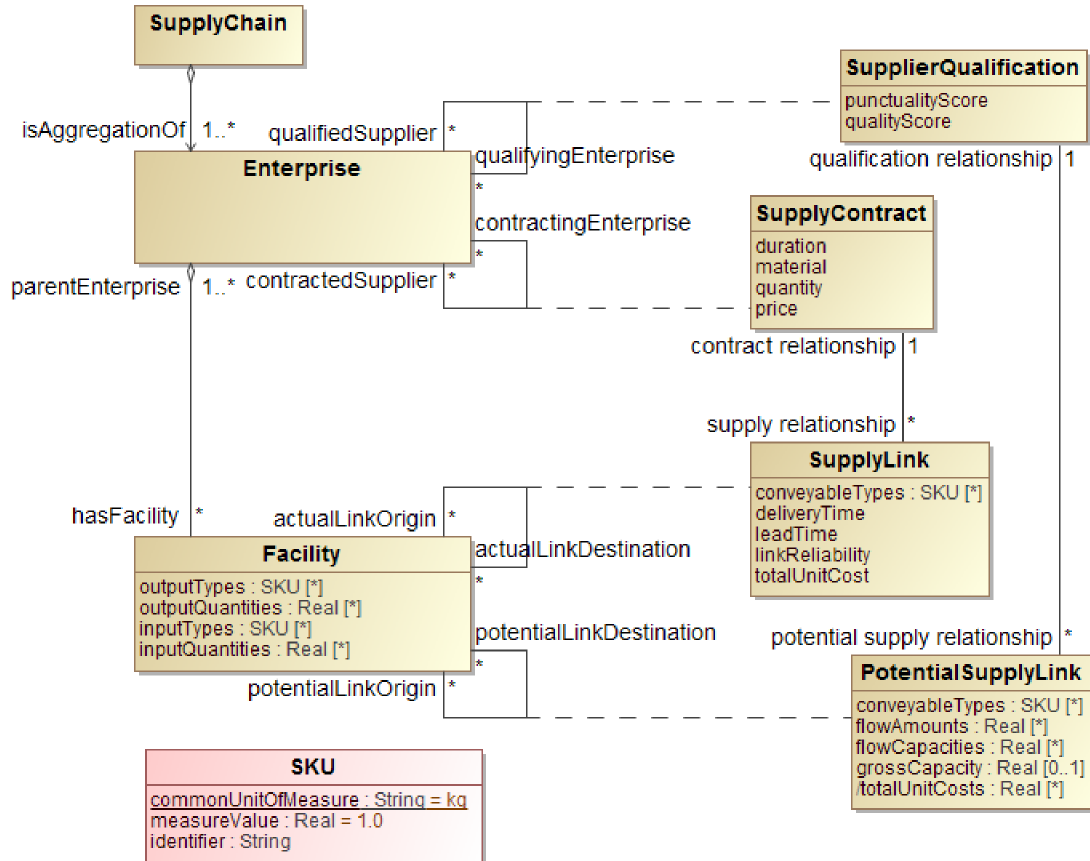


Figure 83: Semantics for a Supply Chain.

Figure 83 shows an M_1-level user model defining the structure of a supply chain. This model may be paired with conforming M_0-level instance models, and should be paired with at least one because the dissertation restricts attention to questions about instance models with concrete instances and data values. However, this first example will not show any conforming instance models to elide details and better illustrate the process.

Question about System: *What are the most influential enterprises in a supply chain at which to invest in helping the supplier adopt new enterprise-level quality control standards? Influence should be measured by diffusion potential.*

A Reed-Kellogg diagram for the question "What are the most influential enterprises in a supply chain at the present time?" is shown in figure 84 [1].

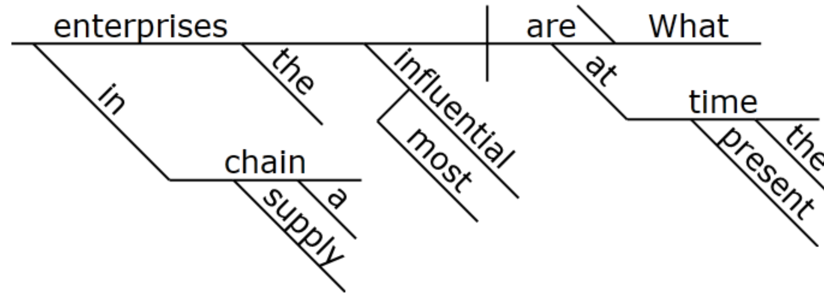


Figure 84: Reed-Kellogg Diagram for the Question in Example 1.

The question concerns describing structure at a single point in time, and figure 84 conforms to the pattern for this category of questions shown in figure 57. The question's subject is the set of instances of the *Enterprise* model element and specifically their influence. The concepts of "influence" and "diffusion potential" may be ambiguous, which can be resolved in two ways: (1) Carefully define "influence" and "diffusion potential" in terms of existing model elements, or (2) Interpret them using human discretion, and state what is actually computed as an approximation type in a qualified answer. The latter way is used in this example.

Abstraction Model: For the given system model and question about it, only basic structural semantics of a token-flow network are needed, shown in figure 85.

[1]Created by the Reed-Kellogg Diagrammer at *http://1aiway.com/nlp4net/docs/help_reed_kellogg.aspx*, used 21march2014.

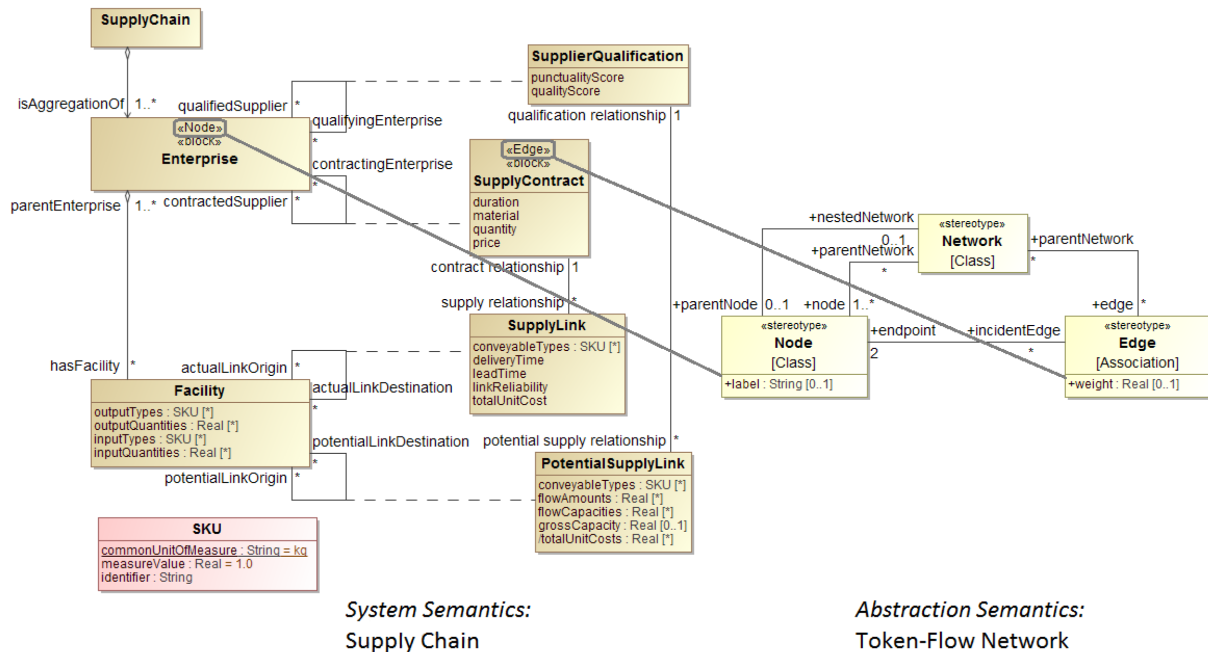


Figure 85: Applying Basic Network Stereotypes to a Supply Chain User Model.

Question about Abstraction: *What are the most influential* <u>*nodes*</u> *in a* <u>*network*</u>*? Influence should be measured by diffusion potential.*

It was claimed in section 6.1 that abstracting a question about a system to a question about a token-flow network can be done in simple cases by replacing the question's subject with its abstracted semantic in the token-flow network model. This is done here, replacing "Enterprise" with "Node" and also "Supply Chain" with "Network".

Qualified Question: A *Qualified Question* may contain qualifications such as time, cost, accuracy, and other requirements for obtaining the answer. Qualifications can influence a choice among multiple answering analyses, and also be passed to a solver to influence solution approximation.

- *Qualification on Time*: An answer is desired in five minutes or less.
- *Qualification on Cost*: The question must be answered using only pre-existing data, meaning properties explicitly recorded in figure 83.

Multiple Answering Analyses:

(1) Centrality measures can quantify the relative importance or influence of a node, and several are defined in section 3.1. Basic centrality measures include *Degree*, *Katz*, *Closeness*, and *Betweenness*. *Eigenvector Centrality* computes a node's value relative to other node values, and has an analogy to equilibrium in a diffusion process. *Alpha Centrality* is a variation of *Eigenvector Centrality* and allows nodes to have external sources of influence.

(2) Influence propagation in social networks is concerned with "*If we can try to convince a subset of individuals to adopt a new product or innovation, and the goal is to trigger a large cascade of further adoptions, which set of individuals should we target?*" [Kempe et al., 2003, p.137] Two basic behavioral models are the *Linear Threshold* and *Independent Cascade* for propagation, and can be combined with optimization to find a subset of most-influential nodes.

Choose an Answering Analysis: How can the choice among multiple answering analyses be narrowed or filtered by indexable information?

- When the *Edge* stereotype is applied to supply contracts, if an Edge's *weight* property is not identified then this prohibits any answering analysis relying on that property. In this example this may exclude Closeness Centrality and Betweenness Centrality.
- The time requirement of five minutes or less prohibits any computationally expensive analysis. Betweenness Centrality requires finding shortest paths between all pairs of nodes, which can be done using the Floyd-Warshall algorithm which has worst-case performance $\mathcal{O}(n^3)$ for a network with n nodes. Given an approximate relationship for the time required by a number of computations in a certain programming language on a certain computer, and given n, $\mathcal{O}(n^3)$ can be converted into concrete time units.

A human analyst might decide that influence propagation models are most appropriate for answering the question, and choose a combination of influence propagation using a linear threshold model and optimization to find a subset of most-influential nodes. Finding a set of k nodes with maximum influence is NP-hard, which may violate the time requirement of five minutes or less depending on network size. However, [Kempe et al., 2003] propose a greedy hill-climbing algorithm whose solution is provably no worse than 63% of optimal.

As illustrated in figure 69, the time requirement must be passed to the solver, which must know how to interpret it and use the approximation algorithm for sufficiently large network instances.

Qualified Answer: Given a token-flow network instance model with n nodes and a user's choice of $k \leq n$, the analysis will output k nodes with maximum collective influence. However, the answer should be qualified:

- *Qualification of Approximation Types*: Influence is measured by influence propagation in the sense of "*If we can try to convince a subset of individuals to adopt a new product or innovation, and the goal is to trigger a large cascade of further adoptions, which set of individuals should we target?*" Influence propagates using a linear threshold model.
- *Given the Approximation Types, Qualification of Accuracy*: If the analysis model is solved to optimality, no further qualification is needed. If the network instance is sufficiently large and the solver uses the approximation algorithm from [Kempe et al., 2003], the total number of network nodes active at the end of a diffusion process beginning with only the answer's k nodes activated is no less than 63% of optimal.

7.2 *Example using Behavioral Process Semantics*

System Model: A manufacturing process plan is modeled as a SysML Activity, in this dissertation considered at the M_0 instance level, and shown in figure 86.

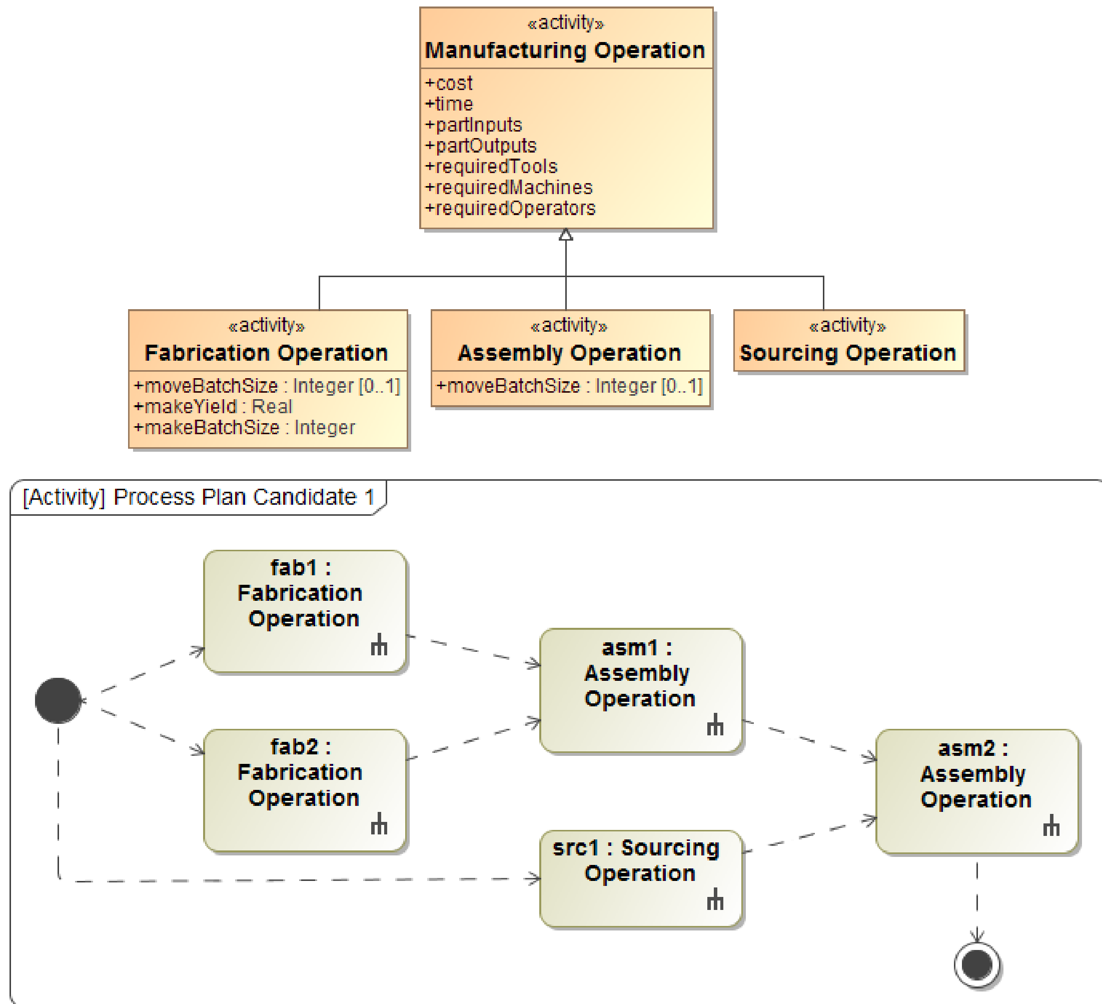


Figure 86: SysML Activity Model of a Manufacturing Process Instance.

In the activity *ProcessPlanCandidate1*, process steps are Call Behavior Actions to the activities *Fabrication Operation*, *Assembly Operation*, and *Sourcing Operation*, which are further defined in the top half of figure 86.

Question about System: *What is the expected raw process time of manufacturing process* ProcessPlanCandidate1?

A Reed-Kellogg diagram for the question "What is the expected interval between start time and end time after executing this process?" is shown in figure 87 [2].

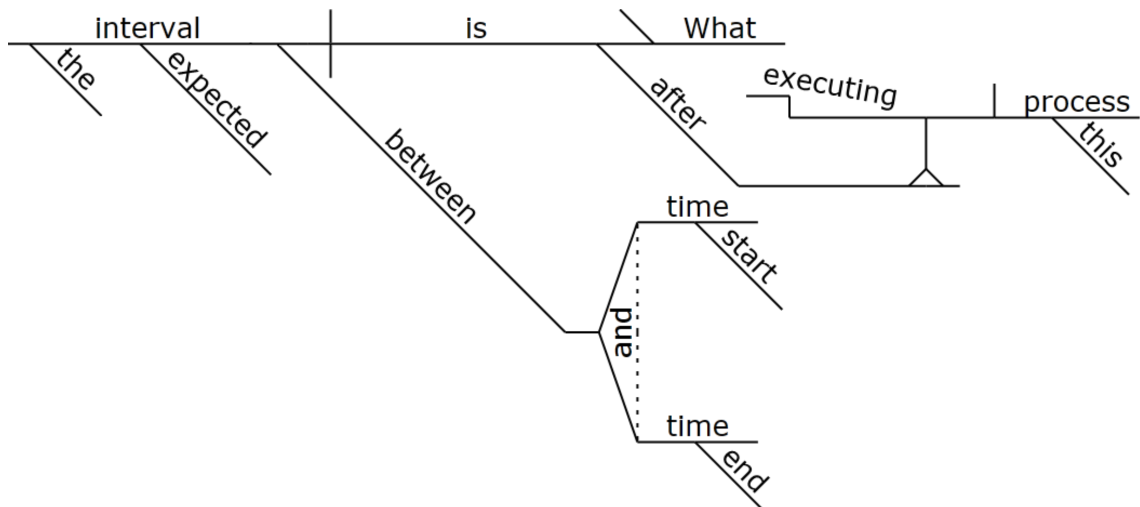


Figure 87: Reed-Kellogg Diagram for the Question in Example 2.

The question concerns predicting behavior spanning multiple points in time, and figure 87 conforms to the pattern for this category of questions shown in figure 61. The question's subject is the expected length of a time interval, specifically *raw process time* defined by [Hopp and Spearman, 1996, p.225] as "the sum of long-term average process times of each workstation in the line" or "the average time it takes a single job to traverse the empty line". Predicting behavior requires a behavioral model, which here is contained in the system model and specifically in the SysML language definition of an Activity.

Abstraction Model: For the given system model and question about it, behavioral semantics for a token-flow *Process Network* are needed, shown in figure 88.

[2] Created by the Reed-Kellogg Diagrammer at *http://1aiway.com/nlp4net/docs/help_reed_kellogg.aspx*, used 22march2014.

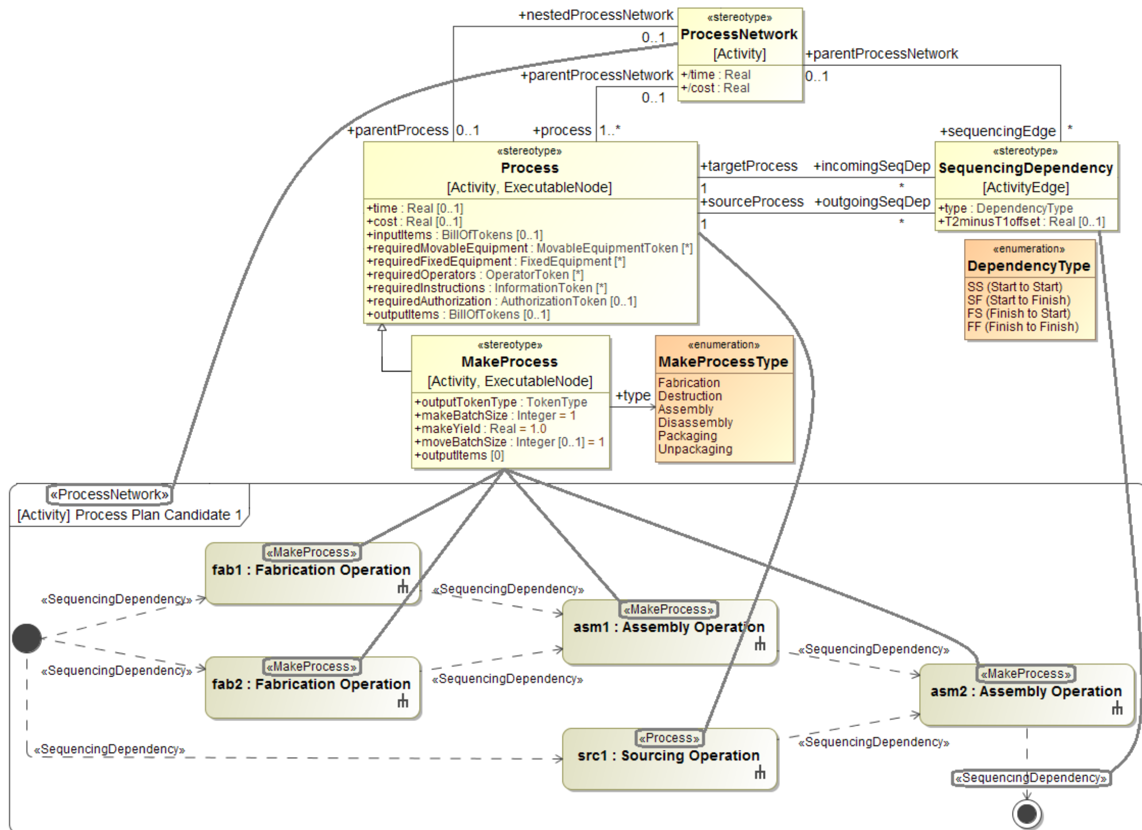


Figure 88: Apply Process Network Stereotypes to a SysML Activity Model of a Manufacturing Process.

Question about Abstraction: *What is the expected raw process time of process network* ProcessPlanCandidate1?

It was claimed in section 6.1 that abstracting a question about a system to a question about a token-flow network can be done in simple cases by replacing the question's subject with its abstracted semantic in the token-flow network model. This is done here, replacing "Manufacturing Process" with "Process Network" and *raw process time* with an analogous concept for a Process Network.

Qualified Question: A *Qualified Question* may contain qualifications such as time, cost, accuracy, and other requirements for obtaining the answer. Qualifications can influence

a choice among multiple answering analyses, and also be passed to a solver to influence solution approximation.

- *Qualification on Time*: An answer is desired in five minutes or less.
- *Qualification on Cost*: The question must be answered using only pre-existing data, meaning properties explicitly recorded in the upper half of figure 86 and any Event Logs. Suppose that Event Logs are recorded from executions of other manufacturing process instances (which may include the same operation types used in the lower half of figure 86), but not the specific manufacturing process instance shown.
- *Qualification of Tools*: No discrete-event simulation solver (AnyLogic, Arena, SimEvents, Simio, Tecnomatix, etc.) is available. However, compilers for general-purpose languages with mathematical libraries are available (e.g. gcc, *javac* with a runtime environment, perhaps even MATLAB).

Multiple Answering Analyses:

(1) A rough approximation is to add the time required by each component process. Depending on the M_0-level process network instance, this may be wildly inaccurate because it ignores that processes may execute in parallel. This yields an analysis model $E[T] = \sum_{i=m}^{n} T_i$ for a Process Network instance with m processes, which may be incomplete depending on models for the T_i (literal numbers, deterministic formulas, random variables, etc.)

(2) Given a process network instance, interpret sequencing dependencies for an analytical formula $E[T] = f(T_1, \ldots, T_6)$ involving the T_i and summation and maximization operators. Definition of an analysis to interpret sequencing dependencies to determine f is part of the M_1-level analysis model, for example a definition of the Critical Path Method (the CPM in PERT/CPM) to find a process network's longest path. This analysis definition is included here only as an algorithm for simple cases in appendix C. As before, the analysis model may be incomplete depending on models for the T_i.

(3) Discrete-event simulation is an answering analysis which can be useful if the T_i have probability distribution models. Discrete-event simulation analysis may also be extensible; for example, the analysis model might be extended at a later time to relax *raw process time* assumptions, which included an empty production line and full resource availability.

Choose an Answering Analysis: How can the choice among multiple answering analyses be narrowed or filtered by indexable information?

- The time requirement for an answer in five minutes or less prohibits computationally expensive analysis.
- The cost requirement here is actually enabling, because if any operation types shown in figure 86 were previously executed in different manufacturing process instances, then data from Event Logs can be used to estimate probability distributions for the T_i.
- The tool limitation prohibits discrete-event simulation analysis. However, access to general-purpose languages with mathematical libraries enables sampling random variables. This means that simulation can still be identified as an answering analysis type, but only simple simulation in the sense of sampling, replications, and estimation and not more complex discrete-event simulation logic.

Given the requirements, a human analyst might decide to interpret sequencing dependencies for an analytical formula $E[T] = f(T_1, \ldots, T_6)$ involving the T_i and summation and maximization operators. The analyst might further decide to model the T_i as random variables and estimate distributions from previous instances' execution times, extracted from Event Logs. Thus, a complete M_1-level analysis definition is the formula $E[T] = f(T_1, \ldots, T_6)$, a model of analysis to determine f, models of the T_i (such as $N(\mu_i, \sigma_i^2)$), and a model of statistical estimation analysis (for estimating μ_i and σ_i^2).

Answering Analysis Model: Let T model the process network's time and T_i the time required by each component process. Suppose previous operation instances' execution times follow a normal distribution; note that this allows negative process times, and a better model might be a half-normal distribution. An M_0-level analysis model for the Process Network instance in figure 88 is:

$$
\begin{aligned}
E[T] = E\Big[T_{asm2} + \max\left\{T_{asm1} + \max\{T_{fab1}, T_{fab2}\},\ T_{src1}\right\}\Big] \\
T_{fab1}, T_{fab2} \sim N\left(\mu_{fab}, \sigma^2_{fab}\right) \\
T_{asm1}, T_{asm2} \sim N\left(\mu_{asm}, \sigma^2_{asm}\right) \\
T_{src1} \sim N\left(\mu_{src}, \sigma^2_{src}\right)
\end{aligned}
$$

No explicit model of the analysis for determining f is included, only an algorithm in appendix C. No explicit model of statistical estimation analysis is included, only the output μ_i and σ_i^2 for $i \in \{fab, asm, src\}$. An M_0-level instance model requires the normal distribution parameters to be literal numbers, so for the sake of solution arbitrarily assume that $\mu_{fab} = \mu_{asm} = \mu_{src} = \sigma^2_{fab} = \sigma^2_{asm} = \sigma^2_{src} = 1$.

Solution and a Qualified Answer: For the random variable $N(1,1) + \max\Big\{N(1,1) + \max\big\{N(1,1), N(1,1)\big\},\ N(1,1)\Big\}$, no analytical result is known for its mean. While a discrete-event simulation solver is unavailable, general purpose programming languages are and can be used for sampling, replications, and estimation. Using MATLAB and an arbitrary choice of 50 replications, the solver returns $E[T] \approx 3.6$ However, the answer must be qualified:

- *Qualification of Approximation Types*: This expected value is for *raw process time* which assumes an empty production line with no waiting and full resource availability. Models for the execution time of Fabrication, Assembly, and Sourcing operations are normally-distributed random variables whose parameters were estimated from the operation type's execution time in other processes instances, with independence and stationary assumptions.
- *Given the Approximation Types, Qualification of Accuracy*: A 95% confidence interval for $E[T]$ is approximately $(3.24, 3.96)$.

CHAPTER VIII

CONCLUSIONS AND FUTURE WORK

What connects this dissertation to past, present, and future output from the same research laboratory is the goal of enabling Operations Research analysis of discrete-event logistics systems to be more widely used to support decision-making. Sophisticated methodologies already exist to design mechanical, electrical, and even software products, but to the best of our knowledge methodologies of similar sophistication do not exist to design those products' manufacturing facilities, warehouses, and supply chains [McGinnis et al., 2006]. If Operations Research analysis is used to describe, predict, control, and design discrete-event logistics systems in an ad-hoc manner, large amounts of time and labor can be consumed, verifying and validating analysis can be challenging, little knowledge may be captured, and little is learned. This dissertation relies on an observation that "Operations Research analysis" includes a well-understood and routine subset which is used repeatedly in similar abstract contexts. In this subset, every instance of an analysis model need not be a hand-crafted art, and the methodology can enable this subset to be more widely used in a cost-effective and correct manner.

8.1 Contributions

This dissertation's primary contribution is a Model-Based Systems Engineering methodology enabling Operations Research analysis of discrete-event logistics systems to be more widely used in a cost-effective and correct manner. The process of interest is posing a question about a system model and then identifying and building answering analysis models. Methods include automation, abstraction, and formalization. Tools supporting the methods are where two additional contributions are found - a formal definition of the abstract concept of a token-flow network to support the abstraction method, and a formal definition of a well-formed question to support the formalization method. Investigating tools for the automation method leads to one additional contribution - addressing if and how existing model-to-model

transformation tools can be re-purposed from Model-Driven Architecture of software.

A consequence of the methodology is helping to elucidate exactly what information is needed to answer a question about a system. Token-flow network stereotypes applied to a system model function as a filter and isolate the abstract concepts relevant to formulating answering analysis. Within the question is analysis' purpose, and between a question and token-flow network model must be all information needed to formulate answering analysis, which can then be solved for an answer. Therefore, the methodology isolates the knowledge required to answer a question about a system, and also considers fundamental questions about how to capture that knowledge and what that capture enables. Developments here are promising, but provide only limited answers and leave much room for future work.

8.2 Boundary

The process, methods, and tools of this dissertation's methodology each induce different facets of a boundary which delineates when the methodology is useful and when it is not.

The process - posing a question about a system model and then identifying and building answering analysis models - effectively changes the user interface for Operations Research analyses from analysis models themselves to system models and questions about them. While the process is designed to capture something regularly executed in the status quo, working with formalized system models, formalized abstraction, and formalized questions may be unusual for analysts because these artifacts are often informal and implicit in the status quo. Therefore, a boundary condition is that an organization's business processes for using modeling and analysis to support decision-making should be isomorphic to the process in this dissertation's methodology.

8.2.1 Boundary Induced by the Formalization Method

For formalization, only one set of tools has been identified in this dissertation - formalize system models as SysML user models, the token-flow network as a UML profile, and questions using semantics developed in chapter 5. It is also possible to formalize system models in UML or other MOF-defined languages, although outside of UML/SysML a stereotype-like language element may not exist and a model-to-model transformation to abstract system

to network semantics must be defined differently. The UML/SysML languages induce a boundary condition that users must understand *object-oriented* modeling.

For formalizing questions, a simplifying assumption was that questions about *designing* behavior spanning multiple points in time are out-of-scope. Also, many questions' subject is a subset of model elements and their properties, so a boundary for questions follows from a boundary for the content of token-flow network models. Accommodating well-understood analysis semantics such as a node's degree in place of *number of incident edges* is proposed by an ontology mapping certain analysis semantics back to token-flow network semantics, but this ontology was left for future work. Another limitation on questions is lacking definitions for high-level concepts such as *control*, *risk*, and more.

8.2.2 Boundary Induced by the Abstraction Method

Stereotype application is the mechanism proposed to formally abstract a system model to a token-flow network model. However, this leads to the restrictive boundary condition that a SysML user model must "align with" or "look like" the token-flow network profile applied. Scenarios can be imagined in which this does not hold, for example user models with associations in Block Definition Diagrams or connectors in Internal Block Diagrams which cross the boundary between a parent and a nested network but not through an interface. User models which do not "align with" or "look like" the token-flow network defined in chapters 3 and 4 are outside the boundary of this dissertation's methodology. This restriction can be viewed in both a negative and a positive light; negative regards the profile as constraining design and content of a SysML user model, and positive regards the profile as a design pattern guilding users as they construct a SysML user model.

Section 6.1 stated a boundary condition for abstracting a question: "When a question's subject does not have an abstract analog in the token-flow network model, however, then there are two possibilities. Either the token-flow network definition can be enhanced, or the question falls outside this dissertation's boundary because it is fundamentally not a question about describing, predicting, or controlling token-flow network structure or behavior." Both of these cases depend on a boundary for the token-flow network definition, which

is considered in the following subsections and partitioned into a *defined* boundary based on the token-flow network exactly as defined in chapters 3 and 4 and a *conceptual* boundary based on any possible extensions.

Defined Boundary for a Token-Flow Network

Abstraction reduces information content. Therefore, the token-flow network definition's inclusions and omissions decide which discrete-event logistics system concepts can be the subject of a question. The definition need not include every imaginable token-flow network detail because a question can define its subject in terms of included model elements, but there are some missing concepts whose definition in a question would be very verbose. Concepts which were defined in chapters 3 and 4 include basic structure, tokens, flow, interfaces, levels of abstraction, time, events, a process behavioral model, and resources. Concepts which are known omissions include additional behavioral models, continuous flow, the concept of control, and various omitted properties and subclasses of existing model elements mentioned throughout chapters 3 and 4. Between the inclusions and omissions lies the defined boundary.

The defined boundary can be extended, although some extensions are easier than others. The difficulty of an extension is determined by both the intricacy of a concept and also its dependencies on existing token-flow network elements. Easier extensions include:

- Adding properties to existing model elements.
- Adding subclasses of existing model elements, for example knowledge-related subclasses of *Process* in section 4.2.

New properties and subclasses of existing model elements become more difficult extensions when a property or subclass associates with other token-flow network model elements, creating a dependency on those elements. Harder extensions also include:

- New behavioral models whose underlying conceptual model is a token-flow network, for example as a State Machine. Well-understood definitions already exist including

Markov chains, Finite-state automata, Moore machines, Mealy machines, and Turing machines, but what makes this extension difficult is unifying the variants and integrating a new behavioral model with existing token-flow network semantics. Structure-behavior integration is particularly important given chapter 5's exposition that questions spanning multiple points in time concern structural model elements and properties, integrated with change semantics, and supporting behavioral models for prediction and control.

- The concept of continuous flow. Flow discretized into tokens may be unhelpful for modeling the flow of continuous commodities such as liquid matter, gaseous matter, or analog information signals. What makes this extension difficult is that a continuous flow definition must also be consistent with discretized flows of tokens.
- The concept of *control* is undefined, first discussed in section 4.5, and semantically limits the questions which can be asked about controlling token-flow network behavior. A control metamodel might include semantics such as *allocation*, *plan*, *planning horizon*, *policy*, *decision*, *re-evaluation frequency*, and more. What makes this extension difficult is both the intricacy of the concept and its integration with existing token-flow network semantics.

Conceptual Boundary for a Token-Flow Network

Suppose the token-flow network definition in chapters 3 and 4 is modified and extended in any imaginable way. What types of semantics will never appear, and therefore can never be the subject of a question? Answering this question would reveal the conceptual boundary of this dissertation's methodology. It is surprisingly difficult; a token-flow network is such a broadly-applicable abstraction that it is difficult to find system and question scenarios which are completely unrelated.

For an example, consider an aspect of a system which seems to have no connection

to a token-flow network - managing employees. The subjects of questions may be goal-setting, communication, guidance, correction, discipline, and performance evaluation. In isolation, managing employees has no obvious connection to a token-flow network. However, as soon as an employee associates with a token-flow network, for example as a resource, then goal-setting, communication, guidance, correction, discipline, and performance evaluation now concern controlling human resources and are inside the conceptual boundary. While semantics for controlling human resources are absent from chapters 3 and 4, they can be added if needed to support important questions. A similar example is price negotiations, which in isolation have no obvious connection to a token-flow network, but acquire a connection if price negotiation is within a supply chain or any other discrete-event logistics system. A supply contract defines a relationship (as shown in section 7.2's example), and price negotiation is a control mechanism for setting the relationship's properties. A similar example is risk assessment.

It is an open question how far the token-flow definition can be extended and should be extended. Ever-increasing detail can be added to the definition, but some questions' subjects may require details not appropriate to the general case. At that point it may be prudent to consider partial analysis model generation instead of full, analogous to code-generation of just architecture, with humans manually adding the internal logic of methods at a later time.

8.2.3 Boundary Induced by the Automation Method

It was claimed in the beginning of this chapter that Operations Research analysis includes a well-understood and routine subset which is used repeatedly in similar abstract contexts. "Well understood and routine" means that the analysis is well-defined in an object-oriented way across multiple layered abstraction levels. This is actually a boundary condition for analysis model generation in this dissertation's methodology, if an analysis can be defined in this way, enabling a declarative specification of its formulation to be defined at the user model level and executed at the instance model level. Examples of object-oriented analysis definitions across multiple layered abstraction levels are included in sections 6.3.1, 6.3.2,

and 6.3.3.

8.3 Future Work

Possibilities for future work include:

- *Enhancing the token-flow network definition*: The token-flow network definition's inclusions and omissions determine which discrete-event logistics system concepts the methodology can support. While several omitted concepts were mentioned with the defined boundary, it may be prudent to first invest in developing and prioritizing use cases for discrete-event logistics system modeling and analysis. This can help organize and prioritize enhancements, rather than adding them arbitrarily.

- *Enhanced understanding of the abstraction method's conceptual boundary*: It is an open question how far the token-flow network definition can be expanded and should be expanded. What limits does this place on the range of meaningful questions which can be asked about discrete-event logistics systems?

- *Further development of question semantics*: Semantics need expression in a syntax to comprise a query language for object-oriented models of systems and abstractions. What might section 5.3's semantics look like in the syntax of the OCL language? Section 5.5 suggests some prerequisite investments to integrate structural user models and conforming instance models, including an instance model indexing scheme possibly including *timestamp* for the past, *candidate* and *what-if* for the present, and *scenario* for the future.

- *Expanding the scope of questions to user models and metamodels*: A simplifying assumption in chapter 5 was that questions are asked about M_0-level instance models. There also exist interesting questions about M_1-level user models and M_2-level metamodels. Do semantics, categories, and patterns in chapter 5 generalize?

- *Better understanding a choice among multiple answering analyses*: Multiple analyses may be identified to answer a single question about a single system or network instance model. Can a choice among them always be framed in terms of indexable information,

e.g. can all the criteria a human analyst uses to make a choice be identified and characterized? If not all criteria, which ones?

- *Better understanding the nature of engineering analysis languages*: Is mathematical "set-based" an analog for "object-oriented"? Better understanding model-to-model transformation outputs' languages may help with finding and abstracting patterns in these transformations. More generally, is object-oriented a fundamental paradigm for modeling systems, abstractions, questions, analyses, and model-to-model transformations?

- *Verification and validation*: Whether built manually or automatically, an analysis model is often subject to verification and validation. For simulation analysis models, [Law and Kelton, 2000, p.264-265] define the terms as "Verification *is concerned with determining whether the conceptual simulation model has been correctly translated into a computer program, i.e. debugging the simulation computer program* ... Validation *is the process of determining whether a simulation model (as opposed to the computer program) is an accurate representation of the system.*" Verification concerns if an analysis model was built correctly, whereas validation concerns if an analysis model is suitable to answer a particular question about a system model. Methodologies for verification and validation are out-of-scope in this dissertation but important candidates for future work. [Kleijnen, 1995] surveys verification and validation techniques for simulation analysis models, and in addition this dissertation's methodology offers novel opportunities for verification. One example is that if Process Network semantics in chapter 4 have a mapping to Petri Net semantics, then automatic analysis model-building programs which output discrete-event simulation analysis models might at the same time output a simulatable Petri Net which can be checked for reachability, safeness (boundedness), and liveness (absence of deadlock).

- *Shortcomings of discrete-event simulation languages*: Section 6.3 reveals that discrete-event simulation analysis is not as well-defined and well-structured as other Operations Research analyses. Similar to AMPL for optimization analysis, a canonical language

for discrete-event simulation analysis would greatly help to build these analysis models. This requires integrated definitions for the process-oriented, activity-oriented, event-oriented, and state-oriented discrete-event modeling paradigms, which might be done by formalizing and standardizing mappings from higher-level behavioral models such as a process to lower-level behavioral models such as DEVS to an underlying conceptual model such as a token-flow network. Other suggestions for a discrete-event simulation canonical language are to retain a separation of model and experiments (analogous to a system model and questions about it) and also to add a separation of functional process definitions from physical facility definitions.

- *Implications for teaching*: Industrial Engineering involves both domain knowledge and Operations Research analysis. Industrial Engineering systems have both concrete realizations and abstractions such as a discrete-event logistics system and ultimately a token-flow network. Both the concrete realizations and abstractions of them are already understood in the status quo, but a frequently missing method is formalization, whose purpose is to define something before attempting to describe, predict, control, and design it. Formalization can be skipped, but a likely consequence is shallow, incomplete, and ad-hoc understanding of the system being studied, regardless of how sophisticated the analysis.

8.4 What If the Hypothesis Is Untrue?

A hypothesis is that this dissertation's methodology can enable Operations Research analysis of discrete-event logistics systems to be more widely used in a cost-effective and correct manner. An experiment to rigorously test the hypothesis is time- and resource-prohibitive, so this dissertation's contribution is a methodology to prepare for and shape that experiment. However, it is instructive to consider if and why the hypothesis may be untrue. Suppose that years from now this dissertation's methodology has been heavily modified, shown ineffectual, or abandoned for something better. Reasons might include:

- *The Law of Leaky Abstractions*: The issue is described in [Kiczales, 1992] and the term is coined by [Spolsky, 2002]:

> “*The law of leaky abstractions means that whenever somebody comes up with a wizzy new code-generation tool that is supposed to make us all ever-so-efficient, you hear a lot of people saying ‘learn how to do it manually first, then use the wizzy tool to save time.’ Code generation tools which pretend to abstract out something, like all abstractions, leak, and the only way to deal with the leaks competently is to learn about how the abstractions work and what they are abstracting. So the abstractions save us time working, but they don't save us time learning ... And all this means that paradoxically, even as we have higher and higher level programming tools with better and better abstractions, becoming a proficient programmer is getting harder and harder.*”

An implication is that this dissertation's methodology might make engineering analysis more cost-accessible to trained and experienced analysts who could produce the same output manually, but it may have limits to adoption by decision-makers who are not fluent in engineering analysis. On the other hand, almost any person can use a navigation application on a smartphone to find shortest-path or shortest-time driving directions between two points, without any knowledge of how to construct a network model and compute a shortest path. Somewhere in between is a boundary involving both human and engineering factors which must be understood and possibly manipulated.

- *Integration between SysML models and enterprise information systems*: Many enterprises' status quo experience with formal modeling is in the schemas of information systems, including CAD, PDM, MES, MRP, ERP, and CRM. Implementation of this dissertation's methodology requires integrating process, methods, and tools with enterprises' business processes and methodologies. Maintaining an object-oriented SysML user model whose semantics may vary from enterprise information systems' schemas may prove onerous; at a minimum tool support may be needed to keep the two consistent, and a long-term change needed to business processes is elevating the

importance of formal ontologies from which information system schemas follow, rather than letting schemas be arbitrarily decided by data collected.

- *Issues with formal modeling in object-oriented languages*: While the previous point concerned *if* formal system models in a language such as SysML will be used, this point concerns *how*. Mature design patterns and best practices exist for object-oriented software modeling, but are less mature for systems modeling. One example of a modeling pattern which may prove to be a liability is a recursive pattern in modeling structure, with two possible realizations illustrated in figure 89.

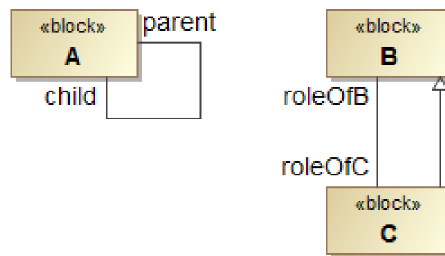


Figure 89: A Recursive Design Pattern for System Structure.

Self-association is the simplest recursive pattern in modeling structure; on the left-hand side of figure 89, an instance of block A is associated with an instance of block A, which may be associated with an instance of block A, etc. On the right-hand side of figure 89, block C inherits a self-association from block B. Recursion is intellectually pleasing but adds complexity to the token-flow network definition. An example used in chapter 3 is that a FlowNode may have FlowNode interfaces, which may have FlowNode interfaces, etc, and each level of abstraction must have consistent *production* and *consumption* properties with the levels above and below. Another example in section 4.5.2 is a Process with ConversionNode interfaces, which themselves host StorageProcesses, which themselves might have FlowNode interfaces, with the last possibility explicitly prohibited because it introduced semantic ambiguity.

APPENDIX A

PETRI NETS

A **Petri Net** is a bipartite directed graph. The two types of nodes are *places* and *transitions*, connected by *directed edges*. Because the graph is bipartite, edges may have (place, transition) or (transition, place) endpoints but not (place, place) or (transition, transition). *Tokens* reside at places, and their position and number change as a Petri Net executes by *firing* transitions. A mathematical definition is a five-tuple $N = \{P, T, I, O, M_0\}$ where:

- P is a finite set of places and T is a finite set of transitions. $P \cup T$ must be nonempty and $P \cap T$ must be empty.
- $I : P \times T \rightarrow \mathbb{Z}_+$ is an *input function* assigning a non-negative integer to each of a transition's incoming edges. If I assigns a positive integer k to an edge, then there exist k parallel directed edges between the (place, transition) pair. This can also be drawn by a single directed edge with weight k, and I is sometimes called an edge weighting function. $O : T \times P \rightarrow \mathbb{Z}_+$ is a transition *output function* assigning a non-negative integer to each of a transition's outgoing edges with analogous meaning.
- $M_0 : P \rightarrow [\mathbb{Z}_+]_{|P|\times 1}$ is an *initial marking* assigning to each place a non-negative number of tokens. M is effectively the Petri Net's state vector.

A transition is *enabled* if the place at the origin of each incoming (place, transition) edge contains a number of tokens greater than or equal to the edge's weight. An enabled transition t fires by, for each incoming edge (q, t), removing from q a number of tokens equal to the edge's weight and depositing in each output place p as many tokens as the weight of outgoing edge (t, p). Each firing updates the marking M, effectively a state transition.

A Petri Net is a model for event-driven dynamic systems. Higher-level coordination concepts for firing transitions can be modeled including sequential firing, concurrent firing, mutually exclusive firing, synchronized firing, prioritized firing [1], and a decision for transitions in conflict. These concepts are built from elementary semantics in ways illustrated in figure 90 [Wang, 2007, p.4].

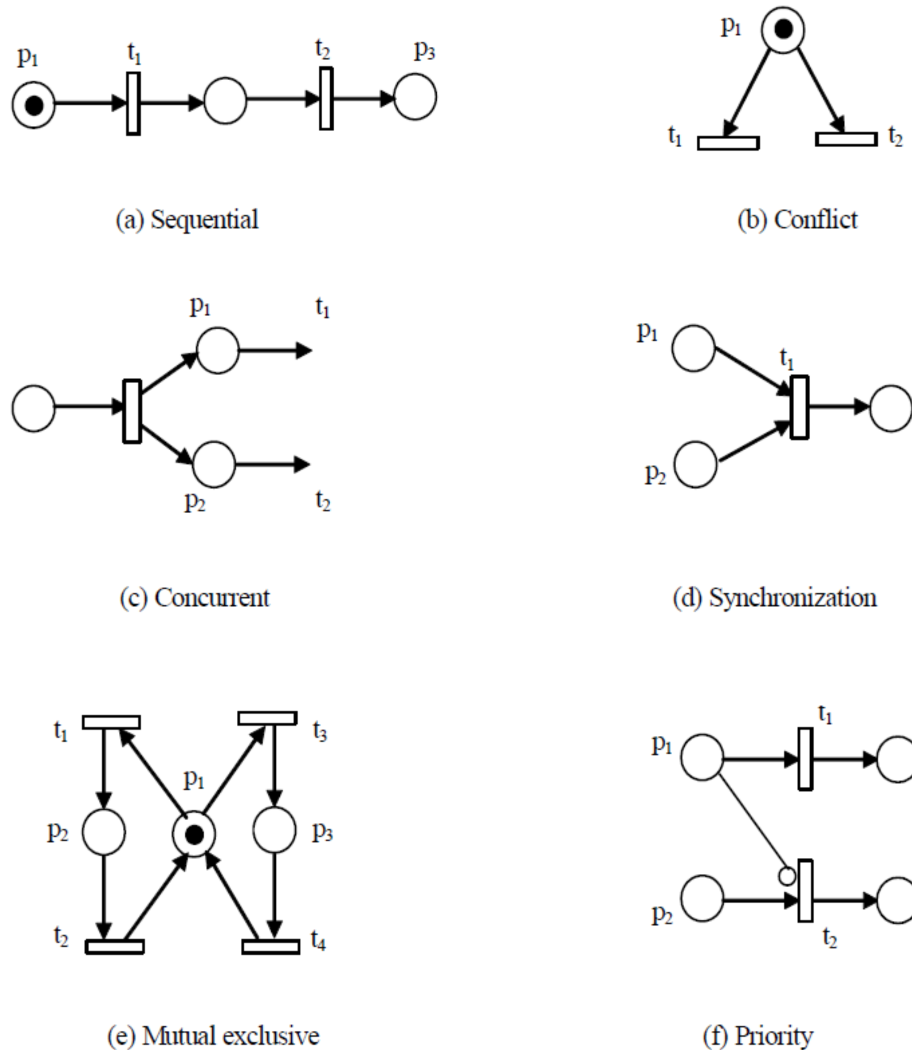


Figure 90: Higher-level Coordination Concepts for Firing Transitions in a Petri Net.

Important behavioral properties of Petri Nets (and a frequent subject of analysis) are reachability, safeness, and liveness. *Reachability* of a specific marking M_i requires proving

[1] Modeling prioritized firing between two transitions requires introducing an *inhibitor edge*, which conditions a transition's enabling on the absence of tokens at the inhibitor edge's source.

the existence a transition firing sequence from M_0 to M_i. A Petri Net is *safe* if every place is safe, and a place is safe (k-bounded) if its number of tokens will never exceed k for any reachable marking from M_0. A Petri Net is *live* if for any reachable marking, it is possible to fire any transition in the net by progressing through some firing sequence. Otherwise, the net may experience *deadlock*.

A Petri Net is a token-flow network with two types of nodes, one type of directed edge, and one token type. If a Petri Net's *state* is its number and position of tokens, then a Petri Net can also be viewed as a state transition system. There are useful extensions, including Colored Petri Nets (multiple types of tokens), hierarchical Petri Nets (multiple levels of abstraction), timed Petri Nets (explicitly introducing time with either deterministic or stochastic intervals), prioritized Petri Nets (adding a priority attribute to transitions), and also adding edge types including reset and inhibitor edges. Many Petri Net extensions are grouped under the title *High-Level Petri Nets*.

APPENDIX B

UML 2.0 ACTIVITIES

Semantically, "An activity is the specification of parameterized behavior as the coordinated sequencing of subordinate units whose individual elements are actions." [OMG UML, 2011, p.324] Syntactically, a UML 2.0 activity is a graph of ActivityNodes and ActivityEdges. Syntactically defining a UML activity as a graph on whose edges flow object and control tokens means that in creating an activity, one creates a token-flow network. A portion of the metamodel is:

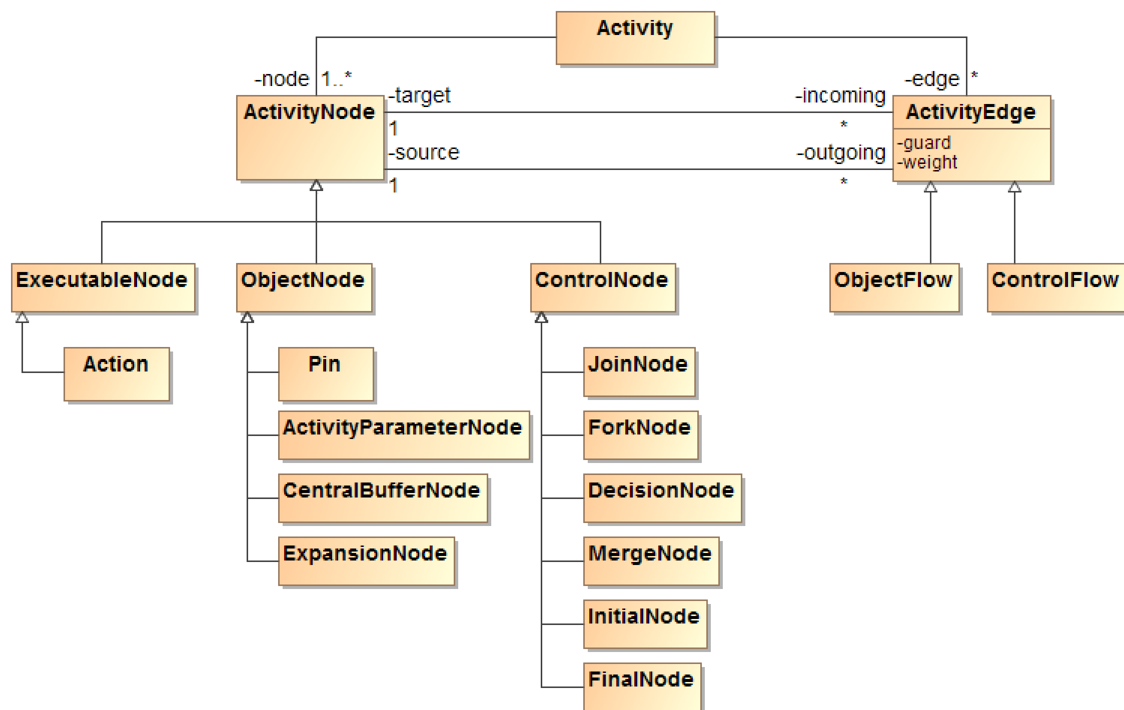


Figure 91: A Portion of the UML 2.0 Activity Metamodel.

Important components are:

- **ExecutableNode** is a type of ActivityNode. Action is its primary subclass, and action has many subclasses. "An action is simple from the point of view of the activity containing it, but may be complex in its effect and not be atomic. As a piece

of structure within an activity model, it is a single discrete element; as a specification of behavior to be performed, it may invoke referenced behavior that is arbitrarily complex." [OMG UML, 2011, p.319] Advanced activity modeling introduces another subclass of ExecutableNode called StructuredActivityNode, which has subclasses LoopNode, ConditionalNode, and SequenceNode.

- **ObjectNode** is a type of ActivityNode. "Object nodes represent objects and data as they flow in and out of invoked behaviors, or represent collections of tokens waiting to move downstream." [OMG UML, 2011, p.325] Since an ObjectNode may hold multiple tokens, it has an *ordering* property whose value is one of {unordered, ordered, LIFO, FIFO}, meaning it can function as a queue. Object Nodes are places where tokens can rest; note that tokens cannot rest on ActivityEdges nor at ControlNodes. The most commonly-used subclasses are Pin and ActivityParameterNode.

- **ControlNode** is a type of ActivityNode. "Control nodes structure control and object flow. These include *decisions* and *merges* to model contingency. These also include *initial* and *final nodes* for starting and ending flows. In IntermediateActivities, they include *forks* and *joins* for creating and synchronizing concurrent subexecutions." [OMG UML, 2011, p.325]

- ActivityEdge has subclasses **ControlFlow** and **ObjectFlow**. "A *control flow* is an activity edge that only passes control tokens. Tokens offered by the source node are all offered to the target node." [OMG UML, 2011, p.366] "An object flow is an activity edge that only passes object and data tokens. Tokens offered by the source node are all offered to the target node ... in the same order as they are taken from the source." [OMG UML, 2011, p.401] This implies that object and control flow networks may overlap in nodes but not edges. The specification says nothing about time elapsed while a token traverses an edge; the semantic of clock time is absent from the metamodel of UML 2.0 activities.

- Object and Control **tokens** flow through the network. "A *token* contains an object, datum, or locus of control, and is present in the activity diagram at a particular

node." [OMG UML, 2011, p.326] Tokens are by nature are discrete, although this does not preclude continuous flows - see the discussion in [Bock, 2006]. In SysML, tokens may also stream across an activity boundary mid-execution, not just at activity initialization and termination. Concerning where they may reside at any moment during activity execution, the specification says "Tokens cannot 'rest' at control nodes, such as decisions and merges, waiting to move downstream. Control nodes act as traffic switches managing tokens as they make their way between object nodes and actions, which are the nodes where tokens can rest for a period of time. Initial nodes are excepted from this rule." [OMG UML, 2011, p.327] There is another exception: "OMG introduced a special rule for ForkNodes saying that they accept incoming tokens when at least one outgoing edge can traverse a token. The tokens offered on the other edges leaving the ForkNode are buffered [at the ForkNode]." [Schattkowsky and Forster, 2007, p.3]

Concerning flow semantics analogous to the enabling and firing of transitions in Petri Nets, "Nodes and edges have token flow rules. Nodes control when tokens enter or leave them. Edges have rules about when a token may be taken from the source node and moved to the target node. A token traverses an edge when it satisfies the rules for target node, edge, and source node all at once." [OMG UML, 2011, p.327]

APPENDIX C

INTERPRETING SEQUENCING DEPENDENCIES IN A PROCESS NETWORK

This appendix provides an example of interpreting a ProcessNetwork's Sequencing Dependencies to aggregate a ProcessNetwork's execution time from the execution times of its component Processes. In a manufacturing context the result is called *raw process time*, the time required for a single job to traverse an empty line with no waiting. Assumptions include no cycles and only finish-to-start sequencing dependencies. It should not matter if nested processes' times are derived, deterministic, or random; the algorithm's purpose is simply to aggregate and respect sequencing constraints, and the output is an expression for raw process time using only summation and maximization operators.

Because of the simplifying assumptions, the algorithm's nature is proof-of-concept. Future work might consider other dependency types and also offsets. Since the four dependency types {FS, SF, SS, FF} are common to project management and PERT/CPM literature, it is possible (but not investigated) that this algorithm or a more general version may already exist in that literature.

C.1 The Algorithm

Initialize Find all terminal operations, with zero outgoing finish-to-start (FS) sequencing dependencies.

- If none, then the process has no end. Stop, and return an exception / error.
- If one or more, open a summation thread for each terminal operation. Into each thread, introduce a summand variable (such as T_i) modeling the terminal operation's time.

Loop For each open summation thread,

- If a operation has no incoming FS sequencing dependencies, close its thread.
- If a operation has exactly one incoming FS sequencing dependency, follow it backwards to the source, and add the source operation's variable T_j as a summand.
- If a operation has two or more incoming FS sequencing dependencies, open a *max* operator, follow each dependency backwards to the source, and add the source operation's variable T_j as an argument of the *max*. Each argument created a new (nested) summation thread.

End For Loop.

Close Finish by taking the *max* of all parallel summation threads.

Simplify

- (By merging threads) For any *max*, commonalities among all arguments can be pulled outside the operator.
- (By eliminating redundant sequencing) For a non-negative process propety such as time, redundant sequencing follows when sequencing dependencies form triangles. This follows from the the triangle inequality - the non-negative measures of a triangle's two legs will always sum greater than or equal to the measure of the triangle's hypotenuse. This allows replacing a *max* operator in which one argument is always greater than or equal to the others by the dominant argument.

C.2 Example 1

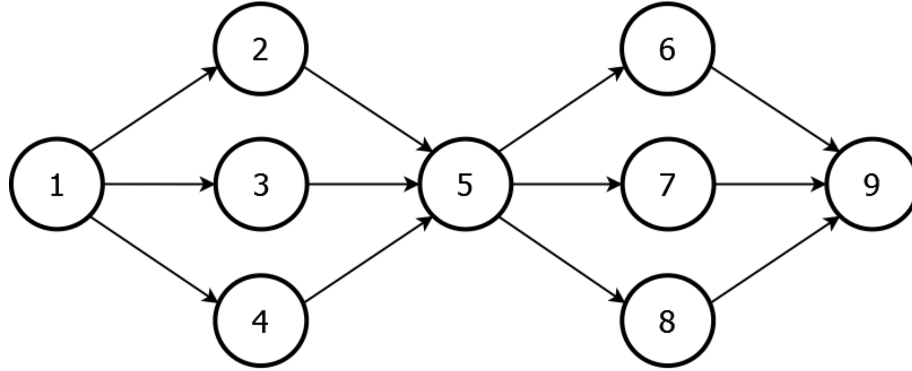


Figure 92: A process composed of sequenced operations. Assume that each directed edge is a finish-to-start sequencing dependency. Assume that each operation has a property T_i modeling time (although no assumptions are yet made about the model's form - literal value, deterministic formula, random variable, etc).

Initialize T_9

Loop

$$
\begin{array}{llllllllllll}
+\max\{T_6, & & & & & T_7, & & & & T_8\} & & \\
+T_5 & & & & & +T_5 & & & & +T_5 & & \\
+\max\{T_2, & T_3, & T_4\} & & +\max\{T_2, & T_3, & T_4\} & & \max\{T_2, & T_3, & T_4\} & \\
+T_1 & +T_1 & +T_1 & & +T_1 & +T_1 & +T_1 & & +T_1 & +T_1 & +T_1 &
\end{array}
$$

Vertical stacking is used to show summation threads. Written conventionally:

$$
\begin{aligned}
T_9 + \max\Big\{ \quad & T_6 + T_5 + \max\{T_2 + T_1,\ T_3 + T_1,\ T_4 + T_1\}, \\
& T_7 + T_5 + \max\{T_2 + T_1,\ T_3 + T_1,\ T_4 + T_1\}, \\
& T_8 + T_5 + \max\{T_2 + T_1,\ T_3 + T_1,\ T_4 + T_1\}\Big\}
\end{aligned}
$$

Simplify by pulling out common arguments:

$$
\begin{aligned}
T_9 + \max\Big\{ \quad & T_6 + T_5 + T_1 + \max\{T_2, T_3, T_4\}, \\
& T_7 + T_5 + T_1 + \max\{T_2, T_3, T_4\}, \\
& T_8 + T_5 + T_1 + \max\{T_2, T_3, T_4\}\Big\}
\end{aligned}
$$

Simplify by pulling out common arguments:

$$
T_9 + T_5 + T_1 + \max\left\{T_2, T_3, T_4\right\} + \max\left\{T_6, T_7, T_8\right\}
$$

The result for raw process time can be verified by inspection.

C.3 Example 2

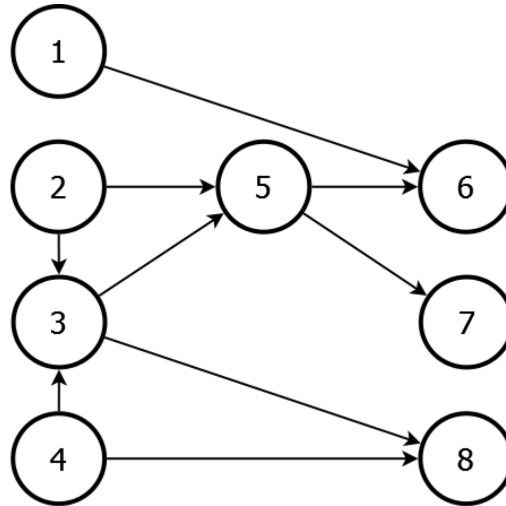


Figure 93: A process composed of sequenced operations. Assume that each directed edge is a finish-to-start sequencing dependency. Assume that each operation has an property T_i modeling time (although no assumptions are yet made about the model's form - literal value, deterministic formula, random variable, etc).

Initialize by identifying three terminal operations T_6, T_7, and T_8. Open a summation thread for each operation. Because of space limitations, **Loop** on each thread separately:

$$\begin{aligned} & T_6 \\ & + \max\Big\{T_1, \quad T_5 \\ & \qquad\qquad + \max\big\{T_2, \quad T_3 \\ & \qquad\qquad\qquad\qquad + \max\{T_2, \quad T_4\} \big\} \Big\} \end{aligned}$$

$$\begin{aligned} & T_7 \\ & + T_5 \\ & + \max\big\{T_2, \quad T_3 \\ & \qquad\qquad + \max\{T_2, \quad T_4\} \big\} \end{aligned}$$

$$\begin{aligned} & T_8 \\ & + \max\big\{T_4, \quad T_3 \\ & \qquad\qquad + \max\{T_2, \quad T_4\} \big\} \end{aligned}$$

Close with an outer MAX of the three unjoined threads, and write conventionally:

$$\max\Bigg\{ \; T_6 + \max\Big\{T_1, T_5 + \max\big\{T_2, T_3 + \max\{T_2, T_4\}\big\}\Big\}, \\ T_7 + T_5 + \max\big\{T_2, T_3 + \max\{T_2, T_4\}\big\}, \\ T_8 + \max\big\{T_4, T_3 + \max\{T_2, T_4\}\big\} \Bigg\}$$

Simplify by eliminating redundant sequencing. Two triangles exist, from $2 \to 3 \to 5$ versus $2 \to 5$ and $4 \to 3 \to 8$ versus $4 \to 8$. This realizes as max operators in which one argument

dominates the other.

$$\max\left\{ \begin{array}{c} T_6 + \max\Big\{T_1, T_5 + T_3 + \max\{T_2, T_4\}\Big\}, \\ T_7 + T_5 + T_3 + \max\{T_2, T_4\}, \\ T_8 + T_3 + \max\{T_2, T_4\} \end{array} \right\}$$

Simplify by pulling out common arguments:

$$\max\left\{ \begin{array}{c} T_6 + \max\Big\{T_1, T_5 + T_3 + \max\{T_2, T_4\}\Big\}, \\ \\ \max\Big\{T_7 + T_5, T_8\Big\} + T_3 + \max\{T_2, T_4\} \end{array} \right\}$$

While more difficult than the previous example, the result for raw process time can again be verified by inspection.

C.4 Sums and Maxima of Random Variables

Given a result for raw process time such as $T = T_1 + T_2 + max\Big\{T_3, T_4\Big\}$, how to actually compute a number? If the T_i are deterministic (whether literal numbers or nested deterministic formulas) then solving for T only requires a calculator, where the sophistication of the calculator is determined by the sophistication of the mathematical operators in any nested formulas. If the T_i are random variables, however, then T is also a random variable, and the analysis model must be extended to compute a random variable feature such as its mean, variance, or distribution.

Suppose a raw process time formula involves only sums. For random variables X and Y defined on the same probabillity space:

- The mean of $X + Y$ can be computed $E[X + Y] = E[X] + E[Y]$ by the linearity of expectation, true whether or not X and Y are independent.
- The variance of $X+Y$ can be computed $var(X+Y) = var(X)+var(Y)+2cov(X, Y)$, and the covariance term disappears if X and Y are independent.
- No general formula exists for the distribution of $X+Y$; it depends on the distributions of X and Y and their dependence. Results include:
  - If $X \sim Normal(\mu_x, \sigma_x^2)$, $Y \sim Normal(\mu_y, \sigma_y^2)$, and X and Y are independent, then $X + Y \sim Normal(\mu_x + \mu_y, \sigma_x^2 + \sigma_y^2)$. This generalizes for a sum of more than two terms.

- If $X, Y \sim Uniform(0,1)$ and independent, then $X + Y \sim Triangular(0,2)$.
- If $X_1, \ldots, X_n \sim Bernoulli(p)$ and independent, then $X_1 + \ldots + X_n \sim Binomial(n,p)$.
- If $X_1, \ldots, X_n \sim Geometric_0(p)$ and independent, then $X_1 + \ldots + X_n \sim NegativeBinomial(n,p)$.

If a formula for raw process time includes only sums, then features of T including its mean, variance, and in some cases distribution have closed-form solutions. However, if a formula for raw process time also includes *max* operators, then closed-form solutions for features of T are difficult to find. Options include:

- Use rough (and possibly inaccurate) approximations, such as replacing each maximization by the argument with the largest mean.
- Use simulation. Sample each component random variable $T_1, \ldots, T_n$ from their known distributions and compute T. Perform replications, then return estimators such as sample mean for $E[T]$ and sample variance for $var(T)$.

References

ALANEN, M. AND PORRES, I. 2003. A relation between context-free grammars and meta object facility metamodels. Technical Report no. 606, Turku Centre for Computer Science (TUCS).

ALEXANDER, C., ISHIKAWA, S., SILVERSTEIN, M., JACOBSON, M., FIKSDAHL-KING, I., AND ANGEL, S. 1977. A Pattern Language. Oxford University Press.

ALLEN, T. T. 2011. Introduction to discrete event simulation and agent-based modeling: voting systems, health care, military, and manufacturing. Springer.

BATARSEH, O., MCGINNIS, L., AND LORENZ, J. 2012. MBSE supports manufacturing system design. *In* 22nd Annual INCOSE International Symposium, Rome, Italy.

BATARSEH, O. AND MCGINNIS, L. F. 2012. System modeling in SysML and system analysis in arena. *In* Proceedings of the 2012 Winter Simulation Conference, pp. 2924–2935. IEEE.

BAYER, T., CHUNG, S., COLE, B., COOKE, B., DEKENS, F., DELP, C., GONTIJO, I., LEWIS, K., MOSHIR, M., RASMUSSEN, R., ET AL. 2012. Early formulation model-centric engineering on NASAs Europa mission concept study. *In* 22nd Annual INCOSE International Symposium, Rome, Italy. Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2012.

BELLAMY, M. A. AND BASOLE, R. C. 2012. Network analysis of supply chain systems: A systematic review and future research. *Systems Engineering* .

BERNARDI, S., DONATELLI, S., AND MERSEGUER, J. 2002. From UML sequence diagrams and statecharts to analysable petri net models. *In* Proceedings of the 3rd international workshop on Software and performance, pp. 35–45. ACM.

BERNER, S., GLINZ, M., AND JOOS, S. 1999. A classification of stereotypes for object-oriented modeling languages, pp. 249–264. *In* UML'99 - The Unified Modeling Language. Springer.

BERNERS-LEE, T., HENDLER, J., AND LASSILA, O. 2001. The Semantic Web. *Scientific American* 284:28–37.

BERTSIMAS, D. AND TSITSIKLIS, J. N. 1997. Introduction to Linear Optimization. Athena Scientific, Nashua, NH.

BISWAS, S. AND NARAHARI, Y. 2004. Object oriented modeling and decision support for supply chains. *European Journal of Operational Research* 153:704–726.

BOCK, C. 2003. UML 2 activity and action models. *Journal of Object Technology* 2:43–53.

BOCK, C. 2006. SysML and UML 2 support for activity modeling. *Systems Engineering* 9:160–186.

BUTLER, C. A. AND ROBINSON, R. S. 2013. Introduction: 2012 Daniel H. Wagner prize for excellence in operations research practice. *Interfaces* 43:397–399.

CHAMBERLIN, D. D. AND BOYCE, R. F. 1974. Sequel: A structured english query language. *In* Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on data description, access and control, pp. 249–264. ACM.

CLEENEWERCK, T. AND KURTEV, I. 2007. Separation of concerns in translational semantics for DSLs in model engineering. *In* Proceedings of the 2007 ACM symposium on applied computing, pp. 985–992. ACM.

CLOUTIER, R. J. AND VERMA, D. 2007. Applying the concept of patterns to systems architecture. *Systems Engineering* 10:138–154.

COLLINS, S. T., YASSINE, A. A., AND BORGATTI, S. P. 2009. Evaluating product development systems using network analysis. *Systems Engineering* 12:55–68.

CUTTING-DECELLE, A.-F., YOUNG, R. I., MICHEL, J.-J., GRANGEL, R., LE CARDINAL, J., AND BOUREY, J. P. 2007. ISO 15531 mandate: A product-process-resource based approach for managing modularity in production management. *Concurrent Engineering* 15:217–235.

DIESTEL, R. 2010. Graph Theory, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Heidelberg.

DRATH, R., LUDER, A., PESCHKE, J., AND HUNDT, L. 2008. AutomationML - The glue for seamless automation engineering. *In* IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2008), pp. 616–623. IEEE.

EMF ATL 2014. ATLAS Transformation Language (ATL). Eclipse Modeling Project: Model to Model Transformation Subproject, http://www.eclipse.org/atl/. Viewed 31march2014.

EMF ECORE 2014. ECORE Metamodeling Language. Eclipse Modeling Framework (EMF), http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html. Viewed 20feb2014.

ESTEFAN, J. A. 2008. Survey of model-based systems engineering (MBSE) methodologies, revision B. Research Report from Jet Propulsion Laboratory, California Institute of Technology.

FALTINSKI, S., NIGGEMANN, O., MORIZ, N., AND MANKOWSKI, A. 2012. AutomationML: From data exchange to system planning and simulation. *In* IEEE International Conference on Industrial Technology (ICIT 2012), pp. 378–383. IEEE.

FISHWICK, P. A. 2007. The languages of dynamic system modeling, pp. 1.1–1.12. *In* Handbook of Dynamic System Modeling, Computer and Information Science Series. Chapman & Hall/CRC.

FOURER, R., GAY, D. M., AND KERNIGHAN, B. W. 2002. AMPL: A Modeling Language for Mathematical Programming. Duxbury Press / Brooks/Cole Publishing Company, 2nd edition.

FRIEDENTHAL, S., MOORE, A., AND STEINER, R. 2008. A Practical Guide to SysML, Second Edition: The Systems Modeling Language. Elsevier, 1st edition.

Friedenthal, S., Moore, A., and Steiner, R. 2011. A Practical Guide to SysML, Second Edition: The Systems Modeling Language. Elsevier, 2nd edition.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1995. Design patterns: elements of reusable object-oriented software. Addison-Wesley.

Gardner, T., Griffin, C., Koehler, J., and Hauser, R. 2003. A review of OMG MOF 2.0 query/views/transformations submissions and recommendations towards the final standard. *In* MetaModelling for MDA Workshop, pp. 178–197.

Guerra-Zubiaga, D. A. and Young, R. I. 2008. Design of a manufacturing knowledge model. *International Journal of Computer Integrated Manufacturing* 21:526–539.

Haskins, C. 2008. Using patterns to transition systems engineering from a technological to social context. *Systems Engineering* 11:147–155.

Hazelrigg, G. A. 1998. A framework for decision-based engineering design. *Journal of mechanical Design* 120:653–658.

Hoare, C. A. R. 1972. Notes on data structuring, pp. 83–194. *In* O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare (eds.), Structured Programming. London: Academic Press Ltd.

Hopp, W. J. and Spearman, M. L. 1996. Factory physics: Foundation of manufacturing management. Irwin.

Huang, C.-C. 2011. Discrete Event System Modeling Using SysML and Model Transformation. PhD thesis, Georgia Institute of Technology, http://smartech.gatech.edu/handle/1853/45830.

INCOSE 2007. Systems Engineering Vision 2020. Technical Product INCOSE-TP-2004-004-02 version 2.03, International Council on Systems Engineering (INCOSE), http://www.incose.org/ProductsPubs/pdf/SEVision2020_20071003_v2_03.pdf.

Kempe, D., Kleinberg, J., and Tardos, É. 2003. Maximizing the spread of influence through a social network. *In* Proceedings of the ninth ACM SIGKDD international conference on Knowledge Discovery and Data Mining, pp. 137–146. ACM.

Kiczales, G. 1992. Towards a new model of abstraction in the engineering of software. *In* International Workshop on Reflection and Meta-Level Architecture, pp. 67–76.

Kiefer, J. C. 1987. Introduction to Statistical Inference. Springer.

Kim, H., Fried, D., Menegay, P., Soremekun, G., and Oster, C. 2013. Application of integrated modeling and analysis to development of complex systems. *Procedia Computer Science* 16:98–107.

Kiron, D., Shockley, R., Kruschwitz, N., Finch, G., and Haydock, M. 2011. Analytics: The widening divide. *MIT Sloan Management Review* 53:1–22.

Kleijnen, J. P. 1995. Verification and validation of simulation models. *European Journal of Operational Research* 82:145–162.

KLEIN, D. AND MANNING, C. D. 2003. Fast exact inference with a factored model for natural language parsing. *In* Advances in Neural Information Processing Systems 15 (NIPS 2002), pp. 3–10. Cambridge, MA: MIT Press.

KLEPPE, A., WARMER, J., AND COOK, S. 1999. Informal formality? The object constraint language and its application in the UML metamodel, pp. 148–161. *In* The Unified Modeling Language. UML98: Beyond the Notation. Springer.

KLEPPE, A. G., WARMER, J. B., AND BAST, W. 2003. MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Professional.

KOKA, B. R., MADHAVAN, R., AND PRESCOTT, J. E. 2006. The evolution of interfirm networks: Environmental effects on patterns of network change. *Academy of Management Review* 31:721–737.

LARSEN, G. 2006. Model-driven development: Assets and reuse. *IBM Systems Journal* 45:541–553.

LAVALLE, S., HOPKINS, M., LESSER, E., SHOCKLEY, R., AND KRUSCHWITZ, N. 2010. Analytics: The new path to value. MIT Sloan Management Review Special Report in collaboration with IBM Institute for Business Value, Fall 2010.

LAW, A. M. AND KELTON, W. D. 2000. Simulation Modeling and Analysis. McGraw-Hill Boston, MA, 3rd edition.

LEMAIGNAN, S., SIADAT, A., DANTAN, J.-Y., AND SEMENENKO, A. 2006. MASON: A proposal for an ontology of manufacturing domain. *In* Proceedings of the IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS 2006), pp. 195–200. IEEE.

LEWIS, T. G. 2009. Network Science: Theory and Applications. John Wiley & Sons.

MAGNANTI, T. L. AND WONG, R. T. 1984. Network design and transportation planning: Models and algorithms. *Transportation Science* 18:1–55.

MASON, S. J. 1953. Feedback theory-some properties of signal flow graphs. *Proceedings of the IRE* 41:1144–1156.

MCCARTHY, I. 1995. Manufacturing classification: Lessons from organizational systematics and biological taxonomy. *Integrated Manufacturing Systems* 6:37–48.

MCGINNIS, L. AND USTUN, V. 2009. A simple example of SysML-driven simulation. *In* Proceedings of the 2009 Winter Simulation Conference, pp. 1703–1710. IEEE.

MCGINNIS, L. F., HUANG, E., AND WU, K. 2006. Systems engineering and design of high-tech factories. *In* Proceedings of the 2006 Winter Simulation Conference, pp. 1880–1886. IEEE.

MELLOR, S. J., KENDALL, S., UHL, A., AND WEISE, D. 2004. MDA Distilled: Principles of Model-Driven Architecture. Addison-Wesley Professional.

MILLER, J. A., BARAMIDZE, G. T., SHETH, A. P., AND FISHWICK, P. A. 2004. Investigating ontologies for simulation modeling. *In* Proceedings of the 37th Annual Simulation Symposium, pp. 55–63. IEEE.

Miller, J. A., He, C., and Couto, J. I. 2007. Impact of the semantic web on modeling and simulation. *In* P. A. Fishwick (ed.), Handbook of Dynamic System Modeling. Chapman & Hall/CRC.

Molina, A. and Bell, R. 1999. A manufacturing model representation of a flexible manufacturing facility. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 213:225–246.

Mueller, R., Alexopoulos, C., and McGinnis, L. F. 2007. Automatic generation of simulation models for semiconductor manufacturing. *In* Proceedings of the 39th conference on Winter Simulation, pp. 648–657. IEEE Press.

OASIS PPS 2011. Production Planning and Scheduling (PPS) Version 1.0. Organization for the Advancement of Structured Information Standards (OASIS), http://docs.oasis-open.org/pps/pps/v1.0/cs01/pps-v1.0-cs01.html.

Oldfather, P. M., Ginsberg, A. S., and Markowitz, H. M. 1966. Programming by questionnaire: How to construct a program generator. RAND Report RM-5129-PR, RAND Corporation, Santa Monica CA.

OMG QVT RFP 2002. Request for Proposal: MOF 2.0 Query / Views / Transformations. Object Management Group, http://www.omg.org/cgi-bin/doc?ad/2002-4-10. OMG Document Number *ad/2002-04-10*.

OMG OCL 2010. Object Constraint Language Version 2.2. Object Management Group, http://www.omg.org/spec/OCL/2.2. OMG Document Number *formal/2010-02-01*.

OMG BPMN 2011. Business Process Model and Notation (BPMN) Version 2.0. Object Management Group, http://www.omg.org/spec/BPMN/2.0/. OMG Document Number *formal/2011-01-03*.

OMG QVT 2011. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Object Management Group, http://www.omg.org/spec/QVT/1.1/. OMG Document Number *formal/2011-01-01*.

OMG UML 2011. OMG Unified Modeling Language (OMG UML) Superstructure Version 2.4.1. Object Management Group, http://www.omg.org/spec/UML/2.4.1/Superstructure. OMG Document Number *formal/2011-08-06*.

OMG SysML 2012. OMG Systems Modeling Language (OMG SysML) Version 1.3. Object Management Group, http://www.omg.org/spec/SysML/1.3/. OMG Document Number *formal/2012-06-01*.

OMG MOF 2013. OMG Meta Object Facility (MOF) Core Specification Version 2.4.1. Object Management Group, http://www.omg.org/spec/MOF/2.4.1/. OMG Document Number *formal/2013-06-01*.

Paredis, C., Bernard, Y., Burkhart, R., de Koning, H., Friedenthal, S., Fritzson, P., Rouquette, N., and Schamai, W. 2010. An overview of the SysML-modelica transformation specification. *In* 20th Annual INCOSE International Symposium, Chicago, IL.

Paynter, H. M. 1961. Analysis and design of engineering systems. MIT press, Boston.

Peak, R., Burkhart, R., Friedenthal, S., Wilson, M., Bajaj, M., and Kim, I. 2007. Simulation-based design using SysML - part 1: a parametrics primer. *In* 17th Annual INCOSE International Symposium, San Diego, CA.

Pegden, C. D., Shannon, R. E., and Sadowski, R. P. 1995. Introduction to simulation using SIMAN. McGraw-Hill New York, 2nd edition.

Pfister, F., Chapurlat, V., Huchard, M., and Nebut, C. 2011. A design pattern meta model for systems engineering. 18th *International Federation of Automatic Control (IFAC) World Congress, Milano, Italy* .

Pratt, M. J. 2001. Introduction to ISO 10303 - the STEP standard for product data exchange. *Journal of Computing and Information Science in Engineering* 1:102–103.

Reed, A. and Kellogg, B. 1896. Higher Lessons In English. Revised edition.

Richters, M. and Gogolla, M. 1998. On formalizing the UML object constraint language OCL. *In* Proceedings of the 17th International Conference on Conceptual Modeling (ER98), pp. 449–464. Springer LNCS.

Roberts, F. S. 1978. Graph theory and its applications to problems of society. Number 29 in CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia: Society for Industrial and Applied Mathematics (SIAM).

Robinson, S., Nance, R. E., Paul, R. J., Pidd, M., and Taylor, S. J. 2004. Simulation model reuse: Definitions, benefits and obstacles. *Simulation Modelling Practice and Theory* 12:479–494.

Roedler, G. 2002. What is ISO/IEC 15288 and why should I care? Presentation Slides. ISO/IEC JTC1/SC7/WG7, Geneva: International Organization for Standardization.

Rogers, E. M. and Kincaid, D. L. 1981. Communication networks: Toward a new paradigm for research. Free Press New York.

Rouse, W. B. 2009. Engineering perspectives on healthcare delivery: Can we afford technological innovation in healthcare? *Systems Research and Behavioral Science* 26:573–582.

Rudtsch, V., Bauer, F., and Gausemeier, J. 2013. Approach for the conceptual design validation of production systems using automated simulation-model generation. *Procedia Computer Science* 16:69–78.

SCC SCOR 2012. Supply Chain Operations Reference Model revision 11.0. Supply Chain Council (SCC), http://supply-chain.org/scor/11.

Schattkowsky, T. and Forster, A. 2007. On the pitfalls of UML 2 activity modeling. *In* Proceedings of the International Workshop on Modeling in Software Engineering, pp. 8–13. IEEE Computer Society.

Schonherr, O. and Rose, O. 2009. First steps towards a general SysML model for discrete processes in production systems. *In* Proceedings of the 2009 Winter Simulation Conference, pp. 1711–1718. IEEE.

Schruben, L. 1983. Simulation modeling with event graphs. *Communications of the ACM* 26:957–963.

Schruben, L. and Yücesan, E. 1993. Modeling paradigms for discrete event simulation. *Operations Research Letters* 13:265–275.

SISO CMSD 2010. Core Manufacturing Simulation Data - UML Model. Simulation Interoperability Standards Organization (SISO). Document Number *SISO-STD-008-2010*.

Soley, R. and OMG Staff Strategy Group 2000. Model Driven Architecture. White Paper no. 308, Object Management Group, http://www.omg.org/cgi-bin/doc?OMG/00-11-05.pdf.

Son, Y. J. and Wysk, R. A. 2001. Automatic simulation model generation for simulation-based, real-time shop floor control. *Computers in Industry* 45:291–308.

Son, Y. J., Wysk, R. A., and Jones, A. T. 2003. Simulation-based shop floor control: formal model, model generation and control interface. *IIE Transactions* 35:29–48.

Spolsky, J. 2002. The law of leaky abstractions. Blog Post at *http://www.joelonsoftware.com/articles/LeakyAbstractions.html*, Published 11nov2002. Retrieved 24march2014.

Sprague, R. H. 1980. A framework for the development of decision support systems. *MIS Quarterly* 4:1–26.

Sprock, T. and McGinnis, L. F. 2014. Simulation model generation using software design patterns. *In* Proceedings of the 2014 Winter Simulation Conference (pending). IEEE.

Staines, T. S. 2008. Intuitive mapping of UML 2 activity diagrams into fundamental modeling concept petri net diagrams and colored petri nets. *In* 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 2008. (ECBS 2008), pp. 191–200. IEEE.

Storrle, H. and Hausmann, J. H. 2005. Towards a formal semantics of UML 2.0 activities. *In* Proceedings German Software Engineering Conference, volume P-64 of LNI, pp. 117–128.

Tako, A. A. and Robinson, S. 2012. The application of discrete event simulation and system dynamics in the logistics and supply chain context. *Decision Support Systems* 52:802–815.

Viehl, A., Schönwald, T., Bringmann, O., and Rosenstiel, W. 2006. Formal performance analysis and simulation of UML/SysML models for ESL design. *In* Proceedings of the Conference on Design, Automation and Test in Europe, pp. 242–247. European Design and Automation Association.

Wang, J. 2007. Petri nets for dynamic event-driven system modeling. *Handbook of Dynamic System Modeling* pp. 1–17.

WASSERMAN, S. 1994. Social network analysis: Methods and applications, volume 8. Cambridge University Press.

YUAN, Y., DOGAN, C. A., AND VIEGELAHN, G. L. 1993. A flexible simulation model generator. *Computers & Industrial Engineering* 24:165–175.

ZEIGLER, B. P. 1976. Theory of Modeling and Simulation. Wiley Interscience, New York, 1st edition.