Object Oriented Modeling of Multistage Stochastic Linear Programs

Leo Lopes and Robert Fourer

Abstract We present a specialization of the Unified Modeling Language (UML) to help diverse stakeholders in an organization collaborate on the development of Stochastic Optimization Models. Our language describes, at an abstraction level distinct from that possible through algebraic notation, the relationships between decisions and parameters, the dynamics of information acquisition, and the requirements for model input and output. This paper describes the formal language and provides a few illustrative examples.

Key words: Optimization, Modeling, UML

1 Introduction

While Operations Research (OR) applications and software applications differ in fundamental ways, they also share some very important characteristics: complexity; cross-disciplinary nature; and non-expert customers. Furthermore, OR applications often include important software components, and usually reside inside Information Technology (IT) infrastructures. These observations motivated us to study how established Software Engineering (SE) techniques may be adapted to help create OR models. Our emphasis is on Multistage Stochastic Linear Programs with Recourse (MSPRs). Typical applications of MSPRs include asset and liability management

Leo Lopes
Systems and Industrial Engineering, University of Arizona, 1127 E James E Rogers Way Room 111, Tucson, AZ, 85721, e-mail: `leo@sie.arizona.edu`

Robert Fourer
Industrial Engineering and Management Sciences, Northwestern University, 2145 N. Sheridan Road Room C210, Evanston, IL, 60201 e-mail: `4er@iems.northwestern.edu`

Yu et al (2003), energy production and distribution Sen et al (2006); Beraldi et al (2008), strategic supply chain design Alonso-Ayuso et al (2003), and natural resources management Heikkinen (2003).

SE techniques facilitate analysis and documentation and enhance maintainability and reliability. The object oriented (OO) paradigm and graphical modeling languages are important components of current SE techniques. This paper addresses the question of how to adapt these components to aid the development of Stochastic Optimization Models. We have two motivations. First, OO methodology has had great success in tackling very difficult but well structured problems similar in many ways to OR problems. Second, OO methodology is pervasive and still expanding in reach within the modern cyberinfrastructure.

The remainder of this paper proceeds as follows: In Section 2 we examine related OR research on model complexity and integration, as well as similar work in other fields. In Sections 3 and 4 we briefly summarize OO SE methods and Multistage Stochastic Programming. In Section 5, we introduce the major aspects of our language with an illustrative example. In Section 6 we apply our language in a few examples from the literature. Section 7 discusses some conclusions and possible extensions.

2 Graphical Modeling and Communication in Optimization

Previous research on formal treatment of OR modeling can be grouped roughly into two approaches: one whose goal is to produce more natural formulation environments for problems that are relatively precise; and one whose goal is to produce analysis techniques suitable for problems which are very "messy", where formality may not be possible or productive.

Structured Modeling Geoffrion (1987), the Intelligent Mathematical Programming System Greenberg (1996), and Jones' work on graph-based Modeling Systems Jones (1990, 1991), are representatives of the more formal approach. Ideas from those contributions have found their way into implementations like MODLER and ANALYZE Greenberg (1993), and into graphical systems like MIMI/G Jones (1996a), LPForm Ma et al (1996), and gLPS Collaud and Pasquier-Boltuck (1994). A commercial modeling system based on ideas developed in the research above is the Enterprise Optimizer (`http://www.riverlogic.com`). Some of the work in visualization applied to optimization is summarized in Jones (1996b).

Less formal approaches include Problem Structuring Methods (PSM) Rosenhead (1996) and Soft Systems Methodology (SSM) Checkland (2000). These approaches consider conflicting objectives by different actors, group dynamics, incomplete information, ill-defined measures, and other issues that can arise in complex business models.

Our language is designed to support a formal, well structured OR technique. At the same time, it is designed to be used starting at the early stages of the modeling process, at which time some of the conditions well addressed by PSM are still present. This is an important difference between our work and the existing literature, which focuses mostly on replacing Algebraic Modeling Languages (AMLs) like AMPL Fourer et al (2002) or GAMS Brooke et al (1988) with graphical languages Collaud and Pasquier-Boltuck (1994), or with creating detailed consistent multi-level models Geoffrion (1987). In particular, this research addresses the need to maintain a Problem Owner, usually not an OR specialist, involved as an Active Modeler Powell (1997) and to communicate with other stakeholders in the OR project, like IT professionals and operational managers. Our language does not attempt to replace the AML, but to augment and support it.

The UML has been used or extended in a variety of fields including engineering design Felfernig et al (2002), data warehousing Luján-Mora et al (2002, leading to the CWM OMG Standard), groupware Rubart and Dawabi (2002), and secure systems Jrjens (2002). No similar study applying OO to OR-centric systems is available, although a study of Entity Relationship Diagrams applied to optimization Choobineh (1991) exists. The new language SysML Bock (2006), also a specialization of the UML, is of particular interest, since it deals with requirements, constraints, performance measures, and other concepts distinct but similar to those used in OR.

Many important details of stochastic optimization problems are not explicit in our diagrams. This is intentional. *Elision*, modeling an element with certain characteristics hidden in a specific view, is a powerful abstraction mechanism used extensively in the UML. The primary intent of the graphical notation is to describe problems at a high level of abstraction, to encourage discussions between analysts at early stages of the modeling process, or to serve as complementary documentation to an existing precise mathematical model. We believe that customary mathematical notation is adequate to express the relationships in the model at more detailed levels. Algebraic notation is far more concise than any graphical notation can be when representing the same objects and relationships. The algebraic notation in use today is very stable Cajori (1993). For instance, Leibniz used $\int$ for summation of integers as well as integration. Later, Euler introduced $\sum$ to indicate integer summation. Variations in the use of $\sum$ exist in the work of Lagrange, Cauchy, Fourier, and Jacobi up to the 1820s, but except for specialized uses, the use of $\sum$ has remained stable since.

With additional assumptions, in specific application areas, it is possible to devise graphical notations that capture all the detail necessary to build precise models in a practical way. For example, the commercial package Enterprise Optimizer is capable of representing a variety of deterministic linear programs arising from supply chains using icons to represent resources and arcs to represent resource flows. Thus, we have built enough expressiveness and detail into our design so that it is *possible* to represent any linear expression using element diagrams, although we don't foresee this as the primary use case. A significant level of detail can be achieved in our diagrams by using *adornments*. Adornments are optional graphical markers added to an element that add semantic value to its representation. With the aid of adornments, a significant proper subset of algebraic expressions can be rendered in a simple way. An additional function of the adornments is to provide a convenient and intuitive link to access more detailed expressions within a software system.

3 Graphical Modeling and Software Engineering

SE is inherently dynamic and multidisciplinary. As systems became more complex, new methodologies for SE evolved, each with accompanying graphical notations to describe structural or dynamic aspects of systems. Structural aspects define how parts of the system relate to each other. Dynamic aspects describe operations step by step.

The first formal diagram language to find widespread use was the fluxogram or flowchart (Figure 1). Fluxograms use special icons to indicate printing, disk storage, tests, etc. and display control flow using edges. Fluxograms provided basic concepts that would be reused by all the other modeling diagrams that describe the dynamics of systems.

The use of redirections (*i.e.* goto statements), promoted by fluxograms, made large systems difficult to manage. In response, the structured paradigm gained popularity. Its main characteristic was continuous flow of control. Small structures were used for each iterative process, and within each structure systems were again described by a continuous flow of control. Fluxograms did not encourage structured thinking, and thus fell out of favor. Multiple graphical modeling languages like Jackson diagrams Jackson (1983) and data flow diagrams DeMarco (1979) were created to support structured SE and adopted widely. People who were trained to use one set of diagramming techniques were not always comfortable using another, even when both diagrams were used in fundamentally similar ways.

As systems continued to become more complex, the structured paradigm reached its own limits, and the OO paradigm became more popular. This brought to the forefront of the modeling process many concepts which were previously dealt with less formally, like: encapsulation, the notion that an object has a clear interface, but its implementation is hidden from view; and specialization, the concept of having a general class in charge of common functionality, and specialized subclasses for specific behavior. None of the above concepts were necessarily new, but now they were handled explicitly at the modeling level and enforced by development systems.

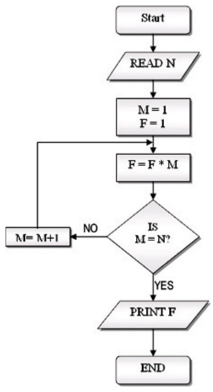


Fig. 1 A fluxogram for computing $N!$ (This image is from Wikipedia, and is in the public domain).

As with the structured paradigm, many modeling methodologies were proposed for the OO paradigm, each accompanied by a graphical notation. Three became particularly popular: the Booch method, by Grady Booch; OMT, by James Rumbaugh and associates; and OOSE by Ivar Jacobson. In the early nineties, at Rational Software Corporation, now part of IBM, they unified their methods. The result was the Unified Modeling Language. Subsequently, the language became an Object Management Group (OMG) standard, currently on version 2.1.

4 Multistage Stochastic Programming

For a more comprehensive introduction to stochastic programming, see Birge and Louveaux (1997, Chapter 1). Our main focus is on the MSPR. In the MSPR, we do not know all the data with certainty when some decisions are made. The data will only be observed at known points in the future. Unfortunately, our decisions need to be made *before* the uncertainty is resolved. Our objective is to make a decision now which minimizes its current (known) cost plus its expected future cost. In the future, there may be **recourse** decisions available *after* the uncertainty has been observed. Those decisions, in turn, may also have to be taken under uncertainty, in recursive fashion, as illustrated in Figure 2.

The periods between when portions of the uncertain data are revealed are called **stages**. To describe what information becomes available when, we created stochasticity diagrams (Section 5.2). Decisions available within a stage are typically dependent on decisions taken earlier and influence future decisions. Within a stage, there are constraints that describe what types of decisions are available and how decisions taken earlier affect the decisions available at this stage. Some of the costs associated with the decisions, the effects of previous decisions, and the limits on the available resources may be uncertain. Element diagrams (Section 5.3) help model these aspects of the MSPR.

Sets are central to practical modeling. Decisions often need to be made over sets of objects of the same kind. Input parameters are also provided over sets. The UML class models this situation. For example, in a facility location model each warehouse has a characteristics like fixed cost or maintenance cost. Each warehouse is an instance of the Warehouse class. A class is represented in UML by a box with compartments for different types of elements, as in Figure 3. UML class diagrams (Section 5.1) describe the sets in an MSPR.

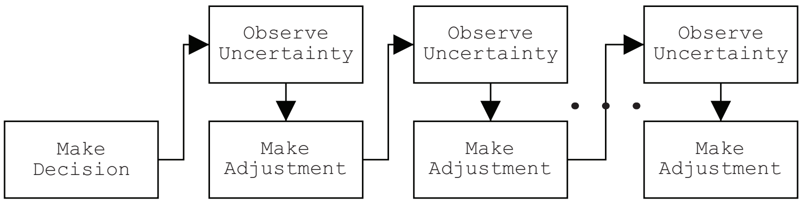


Fig. 2 A stochastic programming model.

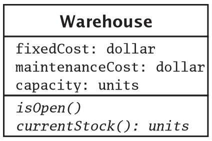


Fig. 3 A simple class.

5 A Meta-model for Multistage Stochastic Programming

The collection of definitions which include Objective, Constraint, Random Variable, etc... is part of the *Meta-model* for stochastic programming. Every problem is an instance of this Meta-model, in the same sense that *Knapsack* is an instance of a combinatorial optimization problem. A Meta-model is similar to a Graph Schema in Jones (1990, 1991).

There are three main types of diagrams in our meta-model: Class Diagrams, specialized from UML Class Diagrams; Stochasticity diagrams, specialized from UML Statechart Diagrams; and Element Diagrams, also specialized from UML Class Diagrams. In this section, we will explain each diagram and illustrate it using a stochastic location-routing problem (SLRP) with the following characteristics:

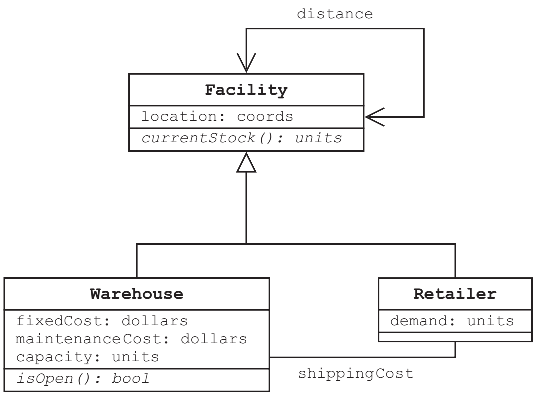
- The objective is to locate a number of facilities in order to maximize expected profit over a finite horizon.
- Facilities must be opened at one or more candidate locations *before* the demand is observed.
- At the start of each planning period, all orders to be served during that period are known. Orders are shipped in less-then-truckload quantities by a fixed number of trucks of limited capacity.
- Each truck services a number of areas, each of which has some demand associated with it. There is a limit to the time spent on each route, which includes travel time as well as service time.
- New routes can be devised at the beginning of each planning period, but a penalty is incurred for changing routes. A penalty is also incurred when a region that was served in the previous planning period is dropped, if that region's demand in the current planning period is greater than zero.
- Demand not served may be lost to a competitor.

Many formulations and solution approaches are plausible for this problem. For the purposes of this research, we will focus on devising a mechanism to communicate the essential information above to all interested parties, not only OR specialists, but also executive decision makers, IT specialists, or Floor Managers.

5.1 Class Diagrams

The function of a class diagram is to describe details of the classes in a system and the relationships between them. Classes are represented by a box with

Fig. 4 A class diagram.



compartments. The first three compartments are used for the class name, attributes, and operations. Other compartments may be used in specific applications for other characteristics.

In Figure 4, we see a *specialization* relationship between Warehouse and Facility. A specialization indicates an *is-a* type of relationship: a Warehouse *is-a* Facility, so it incorporates all its properties. We can also see an *association* relationship between two Facilities, called **distance**. The two are differentiated graphically by the type of arrow. There is also an association between Warehouse and Retailer, called **shippingCost**. Notice that this association does not have an arrow associated with it, but is still valid. Arrows are optional in associations. They are elided here to illustrate a point, but are generally recommended to increase clarity.

The class diagram summarizes all the data used by the optimization problem (parameters), and all the information provided by it (decision variables). It is important for communicating requirements to those responsible for the rest of the IT infrastructure. Therefore, we reused as much functionality from the class diagram as possible. However, elements of a class in an optimization problem have important properties that distinguish them from their SE counterparts. The UML contains mechanisms we can use to describe these differences.

There are two major types of elements in mathematical programming: decisions and parameters. In stochastic programming, parameters may be deterministic or stochastic. Both decisions and parameters can be modeled as attributes of a class. Parameters that are not stochastic are only initialized once, when the class instance is created. The UML property {frozen} (in the UML, properties are expressed by their label in curly brackets) gives an attribute the behavior we desire. In the UML, unless specified otherwise, all attributes are {changeable}. This makes sense in general SE, but is undesirable in the MSPR, since typically only a few parameters are modeled as stochastic. The existing UML construct *note* can be used to indicate that all attributes are {frozen} by default.

Stochastic parameters do not perfectly fit into any concept currently in the UML. They are certainly {changeable}, but in a more specific way. So we create the new property {stochastic}, derived from {changeable}.

The UML has several mechanisms useful for representing decisions. Unfortunately, none are quite perfect. One can think of decisions as operations on a class (*e.g.*, we *open* a facility). Both are given actions as names; and both affect other characteristics of the class. In this view, opening or closing a facility is in fact an action taken on a facility. Unfortunately this view has difficulties. Unlike operations, MSPR decisions are not allowed to affect parameters of a class, although they affect the domains available to other decisions. In the OO framework, operations cannot change other operations (there are frameworks where this is allowed, like in functional programming). Thus, while operations and decisions share some characteristics, one can not be considered a subclass of the other.

A better approach is to define decisions as subclasses of attributes. We suggest defining a stereotype. Stereotyping is one of the extension mechanisms in the UML. The mechanism is used to create meta-classes with specific characteristics. This works very well for decision variables. In particular, the *tagged values* **upperBound** and **lowerBound** can be associated with the decision (in the UML, stereotypes are expressed by their label between guillemots) stereotype. The stereotype is used to define decision variables as special types of attributes.

In Figure 5, we can see all the classes needed to define the stochastic location-routing problem (SLRP) defined earlier. Figure 5 defines *roles* for some of the associations, indicated by a verb followed by a triangle ▶ or ◀ pointing from the subject to the object of the role. The role is an *adornment*, and has no direct translation to the MSPR framework. In particular, it is *not* holding the place of a decision variable. Diagram 5 implies that we are not concerned with which truck gets assigned to each route. If that decision were part of the model, an association class would need to be created and attached to the association between Truck and Route. This association class should then be reflected in the algebraic problem description. If it is not, then there is an inconsistency. Making these inconsistencies easy to spot is what we hope to achieve.

The *maximum* cardinality of each component is unambiguous in the class diagram. Given M_1 Warehouses and M_2 Retailers, a tool can deduce that there are $M_1 \times M_2$ ShippingCosts. Such a tool can also automate parts of the data acquisition

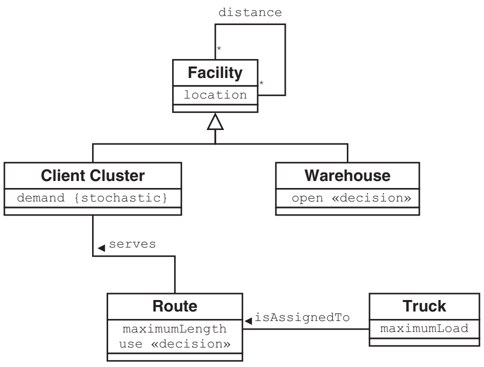


Fig. 5 Classes in the SLRP.

routines, as well as perform integrity checks. It is legal for a tool to allow certain additions to the syntax. With these additions, it is possible to generate equations automatically.

Decisions of arity > 1 present no difficulty. In particular, attributes with arrays can be used when convenient. Associations of arity > 2 are also easily handled within the UML. All that is needed is to draw the lines between classes in a manner that shows unambiguously that the associations involve tuples of classes as opposed to a set of classes taken two at a time. No normalization is required.

More important than generating equations automatically is having a graphical language that improves communication between different analysts. A professional with an IT background, who is likely to know where some of the desired information can be found, will appreciate having a description of the data in a familiar language. People interacting with the model solely at a decision maker level will also appreciate the information about the model contained in the diagram. For example, from the model in Figure 5 it is clear that direct shipments between the warehouse and individual Client Clusters are not available as a recourse action. This message can be conveyed precisely without the need for any mathematical notation or textual description. Another example is the relationship between **Truck** and **Route**. If the trucks all have the same **maximumLoad**, then the association **isAssignedTo** should not have a decision associated with it. However, if **maximumLoad** is different for each truck, and the smallest load is close to the typical demand for a cluster, then the **isAssignedTo** association could become a class containing a decision. Any person involved in the modeling process can spot this detail, and discuss its consequences, without the need for specialized training in OR.

5.2 *Stochasticity Diagrams*

The Stochasticity Diagram provides a concise view of the decision process. It documents that the right decisions are being considered at the right time, using the right information. It describes the point in time at which each piece of information becomes available with certainty; and the consequences of observing the information for the decision process. It is is a UML Statechart, with each state representing a decision process and each event representing the observation of some set of random variables.

Systems which model dynamic aspects of stochastic programming explicitly and associate single-stage LPs with each stage Fourer and Lopes (2008) are called filtration-oriented. The term comes from a type of stochastic process used to describe information release over time. The details of the definition of filtration are not relevant to this discussion, except that the filtration-oriented approach leads to a decomposition of the model that fits well with the OO paradigm. For more details on filtrations, see Neftci (2000). Figure 6 represents the filtration process for the SLRP.

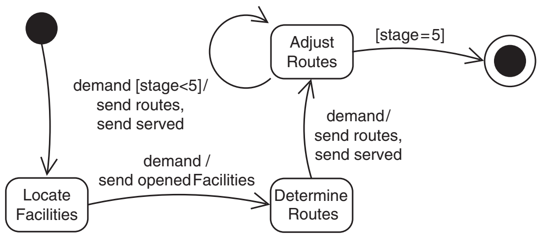


Fig. 6 A simple state machine for a location-routing problem with 5 stages.

Every diagram has an *initial state* (the solid circle) and a *final state* (the other circle). In between, there are a number of other *states* (the round-edged boxes), each representing a set of decisions taken based on the same information. Each state has a *name*.

In between each pair of states there is an *event* (represented by an arc) that describes what new information causes the state transition. Each event may have a *name* and may take parameters. The parameters should be the random variables observed. An event may also have a *guard condition* (represented in between square brackets). When an event occurs, the transition only takes place if the guard condition is satisfied. A *signal* may also be sent when an event occurs, represented in the diagram by the **send** keyword followed by the name of the signal. The signal describes the decisions at period t that must be taken into account later. Conditional probabilities are not shown directly on the diagram. They are clearly important, but they are too much detail for this level of abstraction.

5.3 Element Diagrams

Element diagrams represent the same information as the algebraic model, but at a coarser level. Unlike activity graphs Schrage (2002) or diagrams from LPFORM Ma et al (1996) or gLPS Collaud and Pasquier-Boltuck (1994), element diagrams do not necessarily translate directly into algebraic expressions. Instead, they specify *which* elements are related, rather than specifically *how* they are related. The primary reason for not tying element diagrams directly to algebraic expressions is that element diagrams are used at modeling stages where the specific expressions that determine how elements are related may not be especially relevant. A problem owner or an IT person may be unaware of, uncomfortable with, or even unconcerned with the specific dynamics of the relationships between elements. However, he or she is likely to be interested in knowing that the expert has included the relationship in the model.

As an introduction, consider a problem without stochasticity. Figure 7, has an unadorned element diagram of the Traveling Salesperson Problem (TSP). While not useful for computation, Figure 7 contains a correct and complete representation of the major components of the model. There is an objective, to minimize (thus the triangle points down) the total distance; a decision (the diamond), the route to choose; and a constraint (the box), to visit all nodes. There are also *associations*

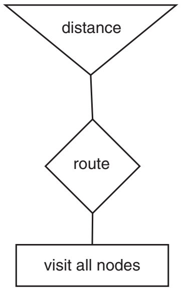


Fig. 7 An unadorned representation of the TSP.

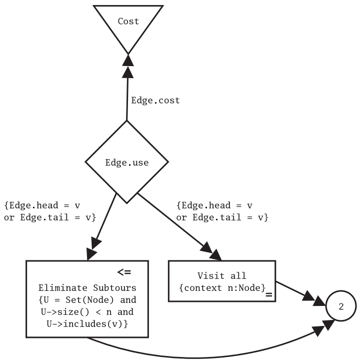

Cost
Edge.cost
Edge.use
{Edge.head = v or Edge.tail = v}
{Edge.head = v or Edge.tail = v}
<=
Eliminate Subtours {U = Set(Node) and U->size() < n and U->includes(v)}
Visit all {context n:Node}
=
2

Fig. 8 An adorned representation of a typical formulation of the TSP

between the different elements in the diagram, represented by the lines connecting them. Each symbol is a specialization of the UML Class.

If adornments are used, the element diagram can express any linear program. In fact, when the element diagram is examined one set of constraints at a time, or one objective at a time, with every relationship expressed correctly, it represents an expression tree equivalent to one generated by a modeling language.

Figure 8 represents the following classical Dantzig-Fulkerson-Johnson TSP formulation in detail: Given a graph $G(V,E)$ with n nodes, let $\delta(U), U \subset V$ be the *cut* of node set U. Define a binary variable x_e, and a cost parameter c_e associated with each edge $e \in E$:

$$\min \quad \sum_{e\in E} c_e x_e$$

$$\sum_{e\in\delta(\{v\})} x_e = 2 \quad \forall v \in V$$

$$\sum_{\{e\in\delta(U)\}} x_e \geq 2 \quad \forall U \subset V, 2 \leq |U| \leq n-1$$

Figure 9 is the corresponding class diagram. Each constraint in Figure 8 has a symbol that identifies its type. Each constraint also includes an Object Constraint Language (OCL) expression in curly brackets. OCL expressions are used to specify multiplicity, and are not related to the constraints of an optimization problem.

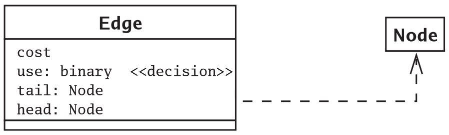


Fig. 9 The Edge and Node classes used in the TSP Element Diagram in Figure 8.

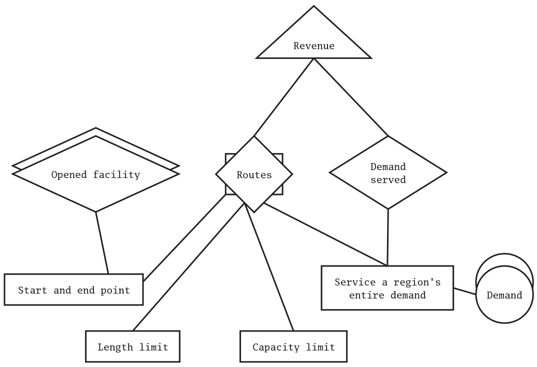


Fig. 10 The **determine route** state in the SLRP.

OCL is a formal language defined as a subset of the UML to aid in clarification of the semantics of models. In the UML, the enforcement of the OCL expressions is left to the implementation UML (2003). A computer implementation is free to replace the OCL with any other formal language, such as an AML.

Each association has a cardinality associated with it. The cardinality may be elided, as in Figure 7. In most cases the association corresponds to a summation or product. An OCL expression can be added to the association to specify which instances of constraint and variable classes are related, as in Figure 8. An implementation may use these expressions to generate AML code.

Each association in Figure 8 has a double arrow (to avoid clashing with reserved symbols in the UML) at one of its ends. The arrow indicates flow balance. An arrow pointing toward the constraint is a credit, and an arrow pointing away from the constraint is a debit. Each association may have a *name*. If so, then this name should be a parameter related with each instance of the association. It is not the instance of the parameter that is used for the name, but the class. For example, in Figure 8, c is the mathematical object corresponding to the association *Edge.Cost* between the decision and the objective, and not c_e.

Consider the SLRP. Figure 10 describes the **Determine Routes** state in Figure 6. The double lines indicate stochasticity or externality. For example, the **facility.open** decision was made before entering this state. It is represented as a decision, but it is not mutable at this stage. It was received in this state as a signal sent by another state, as indicated in Figure 6 by the **send openedFacilities** notation associated with the two states involved. Similarly, the **routes** decision has an optional special marker on it that indicates that this decision might influence decisions made in future stages. The double lines on the **demand** parameter indicate that it is a random variable. All double lines are adornments. The analyst should use judgment to determine when to use them.

6 Illustrations from the literature

We now present a few classical examples taken from Ariyawansa and Felt (2001). For each example, we will provide a description and elided class, stochasticity and element diagrams.

6.1 *Airlift Operations Schedule*

The goal of this model Midler and Wollmer (1969) is to minimize the expected cost of airlift operations. Aircraft resources are allocated based on a forecast of the demands for specific routes. However, the actual demand is unknown. The recourse actions available are: allowing allocated flight time to go unused; switching aircraft from one route to another; and purchasing commercial flights.

6.1.1 The Class Diagram

The class diagram 11 describes **Flight** and **Plane**. Each plane has available hours. The most important parameters associated with each route are: **origin** and **destination**; **demand**, which is stochastic; **spot market cost**, which is the unit cost of purchasing commercial flights on that route; and **unused capacity cost**, the unit penalty for reserving Plane capacity and then leaving it unused. Each Route has two decisions associated with it: **spot purchase**, the amount of demand that will be covered by commercial flights; and **unused**, the amount of capacity reserved but left unused for this route.

The other classes are *association classes*. The **Assignment** and **Reassignment** have the same attributes, but are modeled as different classes. They both have **hours**, the time it takes for a given plane to fly a given route; **cost**, the unit cost of flying a given route with a given plane; and **capacity**, the number of units a given plane can carry when flying a given route. Because these parameters appear as members

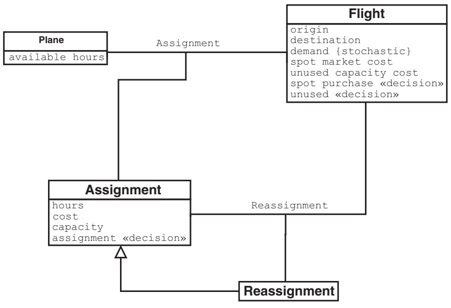


Fig. 11 Airlift operations class diagram

of Assignment instead of Plane, it is clear that in this model they depend on the specific combination of Plane and Route, as opposed to other plausible situations, where some parameters may depend only on Plane.

6.1.2 The Stochasticity Diagram

The stochasticity diagram 12 is simple. The first stage decision is to assign planes to routes; the second stage decision is to do any reassignments necessary.

6.1.3 The Element Diagrams

The first stage element diagram 13 prescribes that in this stage, assignments minimize cost, respecting the number of hours available for each flight. The second stage element diagram 14 is more sophisticated. There are two flows between **reassignment** and **Satisfy demand**. This indicates that reassignments may either increase or decrease the demand satisfied. The OCL constraint indicates which reassignments result in increases or decreases of satisfied demand. The expressions in the original paper involve each plane's carrying capacity, and ratios between flight-hours in different routes for each plane, but they are elided here.

6.2 Forest Planning

The goal of this model Gassmann (1989) is to maximize the revenue obtained from a forest area, by deciding which parts of the forest should be harvested at each point in time. The forest is segmented by the age of trees in a region, and each segment may

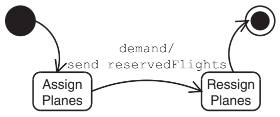


Fig. 12 Airlift operations stochasticity diagram

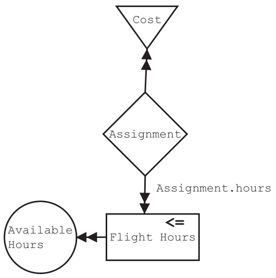


Fig. 13 Airlift first stage element diagram

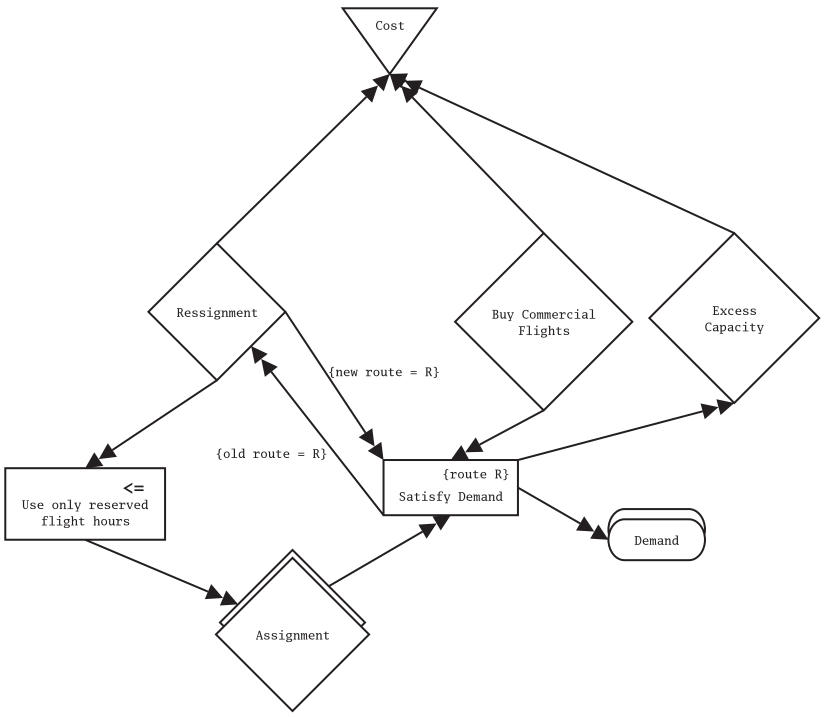
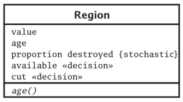


Fig. 14 Airlift second stage element diagram

Fig. 15 Forest planning class diagram.



be worth a different amount per unit area. Due to fire, disease, or other casualties, part of the trees that are not cut may be lost before they move into the next age group.

6.2.1 The Class Diagram

The class diagram 15 contains only one class and no associations. Even though an **age** method is present, it has no semantic meaning in optimization. It is present to illustrate that objects used within our language can be shared with other parties who may have different uses for them.

6.2.2 The Stochasticity Diagram

The stochasticity diagram 16 has a self-transition, which takes place every period. It is triggered by the random vector **casualties** and is processed so long as the stage is less than the total number of stages.

6.2.3 The Element Diagram

The element diagram 17 is associated with the **cut** state in Figure 16. Both cut and uncut forest have value. The amount of forest available in each category is determined by a **balance** constraint that considers the amount available in in the previous period, the amount cut in the previous period, and any casualties that may have occurred. The double arrow linking the **available** variable from the previous period to the balance constraint is hollow, in contrast to all the others. This is an optional adornment to indicate that this parameter is stochastic.

The amount of forest that can be cut now is limited by the available variable, as indicated by the **Availability** constraint. Lastly, the amount of timber the market

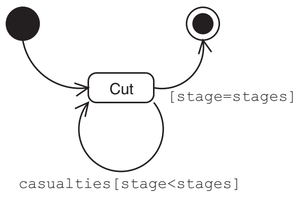


Fig. 16 Forest planning stochasticity diagram

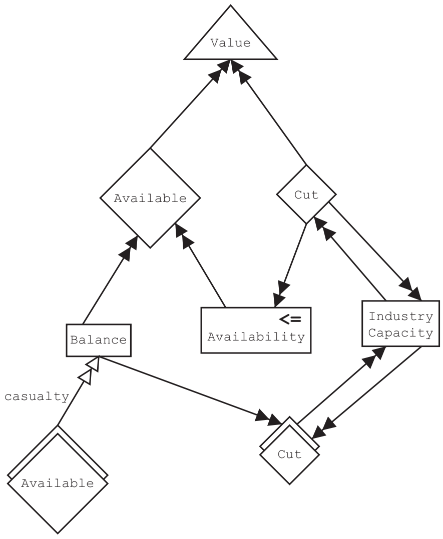


Fig. 17 Forest planning element diagram

is ready to accept is bounded above and below by the amount that was cut in the previous period. Thus, the **Industry capacity** constraint has arrows coming both in and out of both the current stage and previous stage **cut** variables. The Industry capacity constraint could be separated into a lower industry bound and an upper industry bound if desired.

6.3 Electrical Investment Planning

This model comes from Louveaux and Smeers (1998). The objective is to minimize total investment and maintenance costs for electricity generation. Several technologies are available, with different capacities, costs, availabilities, and lifetimes. Some of these characteristics are stochastic, and the goal is to balance all of them in determining an effective investment and operational schedule.

6.3.1 The Class Diagram

The class diagram 18 displays the major components of the model. Demand for electricity can be thought of as coming in different modes, which discretize the demand curve over a cycle. For a certain amount of time during a cycle (**duration** in the class diagram), an amount of power exceeding the previous mode (**power** in the class diagram) will be needed.

The total demand (**demand** in the class diagram) for each **Demand Mode** can be produced by using several **Technologies**, each of which has: an **availability**, the proportion of time a plant using the production technology can be operated in a period; an **operating cost**, which is stochastic (it may depend, for example, on fuel costs); and a **planned capacity** already contracted for.

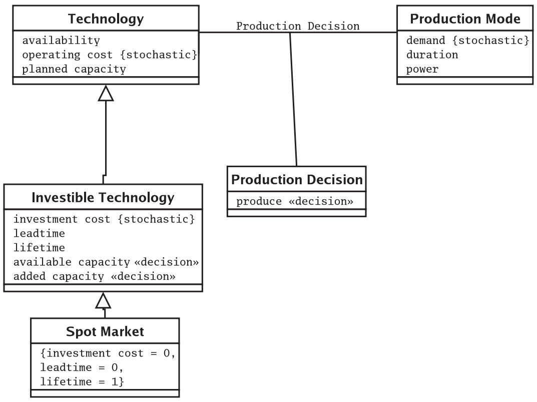


Fig. 18 Electricity investment class diagram.

In addition, it is possible to invest in certain technologies. In this case, one must take into account an **investment cost**, which is stochastic (it may depend, for example, on technological achievements), a **leadtime**, the delay between when the contract is signed and when the facility becomes operational; and also a **lifetime**, describing for how many periods the technology will be usable.

The decision maker may increase the **available capacity** of an **Investible Technology** (which is a subclass of technology), which when combined with the already contracted capacity, leadtime, and lifetime considerations, determines the **available capacity** at any point in time.

There is also a special type of Investible Technology called the **Spot Market**, in which typically the operating costs are very high. Spot Market purchases require no investment, have no leadtime, and only last for the current period.

6.3.2 The Stochasticity Diagram

The stochasticity diagram 19 illustrates that the relevant random variables to be observed are the demand and costs (both operating and investment). There is only one set of decisions, defined by the **Invest** state. The information exchanged between stages is the portfolio of current investments.

6.3.3 The Element Diagram

The element diagram 20 shows that there are two important constraints. **Satisfy demand** ensures that the amount produced or purchased in the spot market is sufficient to satisfy the demand for all modes. The **Capacity** constraint relates how capacity added in previous planning periods and capacity originally contracted affects the production capability at this stage.

The **added capacity** decision has a special marker to indicate that it will influence future decisions. **Demand**, **operating cost** and **investment cost** are marked stochastic, the first because of the ellipse with double lines, and the others because of the hollow double-arrows.

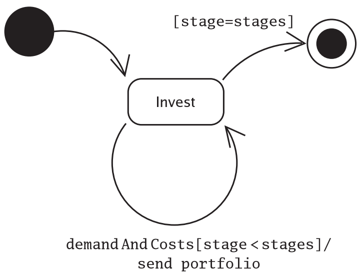


Fig. 19 Electricity investment stochasticity diagram

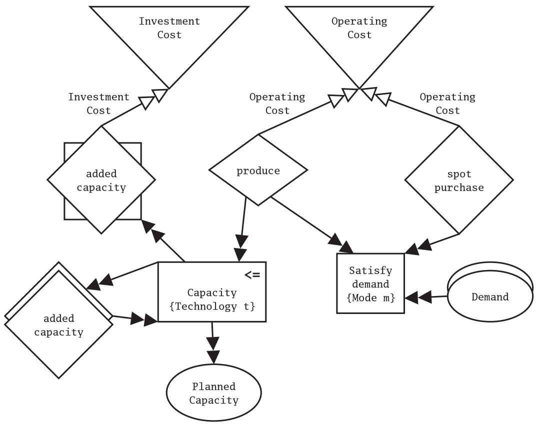


Fig. 20 Electricity investment element diagram.

7 Conclusions and Further Work

Researchers have expressed concern with the increasing disconnect between academic models and those used in business environments Sodhi and Tang (2008). This disconnect, along with changes in the availability of timely information throughout the enterprise can sometimes hinder our ability to convey the value added by OR. This research addresses a small portion of this problem that fits within our expertise. We developed a graphical modeling language based on the UML to facilitate the communication of MSPR models between diverse stakeholders.

The extensive use of elision sets this work apart from previous OR research on modeling systems. While our language can describe linear programs in their entirety, it really is designed to produce an abstract summary of the model, which is a valuable modeling aid used extensively in fields involving information and decision making. Our language is most effective when communicating and documenting the general tradeoffs in a model and the information necessary to adequately analyze the situation underlying the model. The language should be used throughout the lifetime of a model. The development of the model should start with the creation of a set of diagrams, and those diagrams should be used to organize the mathematical expressions in the model. A computer system can guarantee consistency between different levels of abstraction. Future research along this direction should exploit new tools for collaboration available to the enterprise, like the Microsoft Surface.

We believe that the UML is sufficiently expressive, tractable, and extensible to be the basis for a graphical language like ours. The occasional compromises on expressiveness or simplicity we make are worth it when juxtaposed with the benefits of the UML (*i.e.* software support, familiarity, standardization). Other researchers may disagree, and design graphical languages which would perhaps be more clear by dropping the UML assumption. Even within the UML, there are different ways of describing stochastic optimization models. Perhaps a more elegant set of classes can be devised. Empirical research will help refine the design. Future research should also produce graphical languages for other types of OR problems, or show that this particular language can be generalized to cover other classes of problems well.

Ultimately, as with any language, practice will dictate which diagrams become common. The set of objects and diagrams proposed here, however, provide a formal starting point.

References

(2003) OMG Unified Modeling Language Specification. Object Management Group, version 1.5

Alonso-Ayuso A, Escudero LF, Garín A, M T Ortu n, Pérez G (2003) An approach for strategic supply chain planning under uncertainty based on stochastic 0-1 programming. J of Global Optimization 26(1):97–124, DOI http://dx.doi.org/10.1023/A:1023071216923

Ariyawansa KA, Felt AJ (2001) On a new collection of stochastic linear programming test problems. Tech. Rep. 4, Department of Mathematics, Washington State University, Pullman, WA 99164

Beraldi P, Conforti D, Violi A (2008) A two-stage stochastic programming model for electric energy producers. Comput Oper Res 35(10):3360–3370, DOI http://dx.doi.org/10.1016/j.cor.2007.03.008

Birge JR, Louveaux F (1997) Introduction to Stochastic Programming. Springer-Verlag

Bock C (2006) SysML and UML 2 Support for Activity Modeling. Systems Engineering 9(2):160–186

Brooke A, Kendrick D, Meeraus A (1988) GAMS A User's Guide. The Scientific Press

Cajori F (1993) History of Mathematical Notations. Dover Publications, Inc.

Checkland P (2000) Soft systems methodology: a thirty year retrospective. Systems Research and Behavioral Science

Choobineh J (1991) A diagramming technique for representation of linear programming models. Omega

Collaud G, Pasquier-Boltuck J (1994) glps: A graphical tool for the definition and manipulation of linear problems. European Journal of Operations Research

DeMarco T (1979) Structured analysis and system specification. Yourdon Press Upper Saddle River, NJ, USA

Felfernig A, Friedrich G, Jannach D, Zanker M (2002) Configuration knowledge representation using uml/ocl. LNCS

Fourer R, Lopes L (2008) StAMPL: A Filtration-Oriented Modeling Tool for Stochastic Programming, upcoming in INFORMS Journal on Computing

Fourer R, Gay DM, Kernighan BW (2002) AMPL A Modeling Language For Mathematical Programming, 2nd edn. Duxbury Press

Gassmann HI (1989) Optimzal harvest of a forest in the presence of uncertainty. Canadian Journal of Forest Research 19:1267–1274

Geoffrion AM (1987) An introduction to structured modeling. Management Science

Greenberg H (1993) A Computer-Assisted Analysis System for Mathematical Programming Models and Solutions: A User's Guide for ANALYZE. Kluwer Academic Publishers

Greenberg HJ (1996) A bibliography for the development of an intelligent mathematical programming system. ITORMS

Heikkinen VP (2003) Timber harvesting as a part of the portfolio management: A multiperiod stochastic optimisation approach. Manage Sci 49(1):131–142, DOI http://dx.doi.org/10.1287/mnsc.49.1.131.12752

Jackson M (1983) Systems Development. Prentice-Hall

Jones CV (1990) An introduction to graph-based modeling systems, part i: Overview. ORSA Jorunal on Computing

Jones CV (1991) An introduction to graph-based modeling systems, part ii: Graph-grammars and the implementation. ORSA Jorunal on Computing

Jones CV (1996a) Mimi/g: A graphical environment for mathematical programming and modeling. Interfaces

Jones CV (1996b) Visualization and Optimization. Operations Research/Computer Science Interface Series, Kluwer Academic Publishers

Jrjens J (2002) Umlsec: Extending uml for secure systems development. LNCS

Louveaux FV, Smeers Y (1998) Optimal investments for electricity generation: A stochastic model and a test-problem. In: Numerical Techniques for Stochastic Optimization, SpringerVerlag, chap 24, pp 445–453

Luján-Mora S, Trujillo J, Song IY (2002) Extending the uml for multidimensional modeling. LNCS

Ma P, F H Murphy, E A Stohr (1996) An Implementation of LPFORM. INFORMS Journal on Computing

Midler JL, Wollmer RD (1969) Stochastic programming models for scheduling airlift operations. Naval Research Logistics Quarterly 16:315–330

Neftci SN (2000) An Introduction to the MAthematics of Financial Derivatives. Academic Press

Powell SG (1997) The teachers' forum: From intelligent consumer to active modeler, two mba success stories. INTERFACES

Rosenhead J (1996) What's the problem? an introduction to problem structuring methods. Interfaces

Rubart J, Dawabi P (2002) Towards uml-g: A uml profile for modeling groupware. LNCS

Schrage L (2002) Optimization Modeling with Lingo, 4th edn. LINDO Systems Inc.

Sen S, Yu L, Genc T (2006) A stochastic programming approach to power portfolio optimization. Oper Res 54(1):55–72, DOI http://dx.doi.org/10.1287/opre.1050.0264

Sodhi MS, Tang CS (2008) The or/ms ecosystem: Strengths,weaknesses, opportunities and threats. Operations Research 56(2):267–277

Yu LY, Ji XD, Wang SY (2003) Stochastic programming models in financial optimization: A survey. AMO — Advanced Modeling and Optimization 5(1), URL citeseer.ist.psu.edu/yu03stochastic.html