Chapter 2
Ubiquitous Operations Research in Production Systems

Leon F. McGinnis

Introduction

The contemporary education of an operations research (OR) professional is structured around an artisanal model of OR practice. We teach the artistic techniques of the discipline, i.e., the "fundamental methods" of mathematics and mathematical applications, computational methods and tools, and "genres" of application domains, such as production, logistics, or health care delivery. We teach the creative part of the art of OR, i.e., "modeling"—if at all—as a "studio" course; we demonstrate for the budding OR artisan what it means "to model," pose them challenges and critique their work, in the hope that they will acquire that essential esthetic appreciation that characterizes the master OR artisan. The paradigm we teach is the hand-crafted, purpose-built model of a specific problem. We send our graduates out into the world to work as OR professionals have worked for the past 70 years, albeit with an ever-growing and improving technical toolkit. In practice, our graduates are sometimes fortunate enough to work in teams with both domain experts and IT experts to build large scale persistent OR models. These kinds of models are intended to be used routinely over time, and must accommodate changing instance data. In contemporary practice, OR professionals have access to very powerful analysis modeling tools, to IT tools that can harvest data and conform it to our models, to solvers that benefit from 40 years of algorithmic and computational research, and to computing platforms that accommodate gigabyte databases and teraflop computations.

Over the past three decades, this marriage of OR and IT has enabled our profession to accomplish some amazing feats in logistics, finance, medical decision making, and in almost all walks of modern life. One could argue, however, that the penetration of OR in production systems decision making is a fraction of what it could and should be, based on the proven results. Successful applications are not replicated nearly as often as they could be, in large part because of the time and cost for replicating them.

L. F. McGinnis (✉)
H. Milton Stewart School of Industrial and Systems Engineering,
The Georgia Institute of Technology, Atlanta, GA30332-0205 USA
e-mail: leon.mcginnis@gatech.edu

P. S. Pulat et al. (eds.), *Essays in Production, Project Planning and Scheduling,*
International Series in Operations Research & Management Science 200,
DOI 10.1007/978-1-4614-9056-2_2,

There is an emerging need, and a burgeoning opportunity, to "industrialize" OR in production systems. To industrialize OR in production decision making would make a broad range of "standard" OR applications available to the masses of decision makers whose decisions could be significantly improved through more and better OR analysis—much faster and cheaper than is possible today with the conventional approach to model development. The rapid growth of "business analytics" could be viewed as one manifestation of this need and opportunity (see, e.g., Kiron, Schockly et al. 2011) for a recent survey). One contemporary emphasis in business analytics can be viewed as the "industrialization" of statistical methods and tools to enable managers to understand and exploit transactional data without the direct involvement of statistics or IT experts. There is a similar opportunity to industrialize OR methods and tools to enable better decision making for production systems design, planning, and control.

The purpose of this chapter is to explore this concept, and in particular, to argue that methods and tools from computing and software engineering could be used to make OR applications ubiquitous in production systems. Such a transformation would have profound impacts on both the decision makers, who would gain access to these OR tools and methods, and the operations researchers, who develop, implement, and maintain production system decision support systems.

The chapter starts with perhaps the simplest possible example of an OR application in production in order to begin to frame the issues, of which knowledge capture and knowledge management are paramount. This section suggests that there are multiple categories of models that are important for OR applications in production systems. Next comes a very high level introduction to the basic concepts of "model-driven architecture (MDA)," an approach to software engineering that may not be widely familiar to the OR community. The following two sections describe how MDA concepts can be used to capture important knowledge, i.e., models, and to automate the transformation of models of one kind into models of another kind. The implications of these capabilities are explored briefly, two fundamental intellectual challenges are identified, and the chapter closes with some concluding thoughts.

No doubt, there are those in the OR community who will question the wisdom of providing powerful OR analyses to non-OR experts. That question is not the focus of this chapter and, in any event, will be answered by the non-OR experts who will decide for themselves whether or not access to powerful OR analyses will be valuable to them. Rather, the focus here is on the technologies already available to enable the industrialization of OR for particular domains of application.

OR and Production Knowledge

The native tongue of OR is mathematics. At any OR conference, in any session, on any topic, the focus of attention is almost invariably on the mathematical formulation of "the problem" and on the subsequent (mathematical or computational) analysis of

that formulation. A corollary to this phenomenon is that, almost invariably, the original problem stakeholders—those who must make actual decisions about designing or operating the system being modeled—do not speak mathematics with sufficient fluency to truly understand what is being presented. The stakeholders have their own language which is specific to the domain of the problem—a semantic model of the domain that allows them to organize information about what they observe, and communicate efficiently among themselves regarding the problems in their domain.

As an illustration, consider one of the most basic OR modeling examples. In the terms of the stakeholder, the problem is described as follows. A firm has warehouses in 10 cities, each containing a known inventory of a popular product. The firm has orders from 50 customers, scattered around the country, and must decide how to allocate the available inventories to the customer orders in hand. A reasonable way to make the allocation is to seek the largest net profit, considering the price to be charged to each customer, the cost to deliver the product to the customer, and the cost of the product in the warehouse.

The OR instructor, presenting this problem in an introductory course, will draw a network (perhaps even pointing out that it is a directed bipartite graph) to illustrate the connections between warehouses and customers. Then, perhaps implicitly, the instructor will make some associations, which often is referred to as "representing the problem mathematically":

Warehouse index, $i = 1, \ldots, 10$
Cost per unit in the warehouse, $c_i, i = 1, \ldots, 10$
Supply at the warehouse, $s_i, I, \ldots, 10$
Customer index, $j = 1, \ldots, 50$
Customer demand, $d_j, j = 1, \ldots, 50$
Price to customer, $p_j, j = 1, \ldots, 50$
Transport cost per unit between warehouse and customer, $t_{ij}, i = 1, \ldots, 10$, $j = 1, \ldots, 50$
Shipment from warehouse i to customer j, $x_{ij}, i = 1, \ldots, 10$, $j = 1, \ldots, 50$

Finally, the instructor will write out "the problem" using the usual linear programming (LP) formulation of the classical transportation problem as shown in Fig. 2.1. From this point forward, the discussion will be focused on this formulation, this mathematical statement of an analysis which is intended to indicate what the best decisions would be, i.e., the optimal values of the flow variables.

Once students are comfortable with the mathematical formulation, the discussion will then turn to how to actually solve the problem. At this point, students are introduced to a modeling language, which will allow them to prepare the input necessary for some open source or commercial solver. For example, AMPL ("A Mathematical Programming Language," http://www.ampl.com/) might be used to create a computational model of the form shown in Fig. 2.2.

Typically, the decision maker will not directly comprehend the models illustrated in either Fig. 2.1 or 2.2, although in this simple case, the OR analyst can make a direct translation to the domain semantics. The decision variables correspond to allocations, the constraints correspond to conservation relationships, etc. In more complex scenarios, such a translation may not be so easy.

Fig. 2.1 Transportation problem formulation

$$\max P = \sum_{i=1}^{10}\sum_{j=1}^{50}\left[p_j - c_i - t_{ij}\right]x_{ij}$$

$$s.t. \sum_{j=1}^{50} x_{ij} \leq s_i \; \forall \, i$$

$$\sum_{i=1}^{10} x_{ij} \geq d_j \; \forall \, j$$

$$x_{ij} \geq 0 \; \forall \, i, j$$

This simple example illustrates a fundamental aspect of OR-based decision support, namely that there are three important, related but distinct kinds of knowledge involved. The first is *domain knowledge,* which is common to the stakeholders in the domain (though sometimes tacit rather than explicit) and which has its own semantics (warehouse, customer, product, shipment, etc). The second is *analysis knowledge,* or knowledge of a particular analysis, which could be used to support a particular decision in the domain (the LP formulation of the transportation problem) which has its own (mathematical) semantics and syntax, along with, perhaps, knowledge of a particular computational modeling language, and even a particular solver. The third is the *modeling knowledge* that enables the translation of a problem from its domain semantics into the semantics and syntax of a particular OR analysis, considering the limitations of the analytic model. Each category of knowledge is essential for a successful OR decision support project, and each presents its own challenges for knowledge capture and reuse.

Domain knowledge is rarely formalized; in fact it is a common problem to find that different companies in the same industry will use different terms for the same concept, or the same term for different concepts. The standards that have been developed tend to be either very generic and high level (like the supply chain operations reference (SCOR) model for supply chains (Huan et al., 2004)) or focused on information technology (like Business Process Model and Notation (BPMN, http://www.bpmn.org/) or ISA-95 (http://www.isa-95.com/). There have been some research publications on the use of ontologies, e.g., in material handling (Libert and ten Hompel 2011), manufacturing (Jiang et al., 2010), production (Chungoora et al., 2011), but to date, there is not a commonly used, agreed-upon production system ontology. Thus, domain knowledge in production systems remains largely ad hoc, making it difficult to reuse, to teach, or to learn.

This stands in sharp contrast to analysis knowledge, which ultimately is expressed in very precise and canonical mathematical forms and in analysis-specific modeling. This knowledge typically is gained through the student's exposure to the canonical mathematical formulations and particular modeling languages, and by their creating formulations and using the modeling languages for homework and projects;

```
set SOURCE;   # sources
set DEST;   # destinations

param supply {SOURCE} >= 0;   # amounts available at sources
param demand {DEST} >= 0;   # amounts required at destinations

check: sum {i in SOURCE} supply[i] = sum {j in DEST} demand[j];

param cost {SOURCE,DEST} >= 0;   # shipment costs per unit
var Trans {SOURCE,DEST} >= 0;    # units to be shipped

minimizetotal_cost:
sum {i in SOURCE, j in DEST} cost[i,j] * Trans[i,j];

subject to Supply {i in SOURCE}:
sum {j in DEST} Trans[i,j] = supply[i];

subject to Demand {j in DEST}:
sum {i in SOURCE} Trans[i,j] = demand[j];
```

Fig. 2.2 AMPL model for transportation problem formulation

it is refined and deepened through practice in application. Analysis methods are largely mathematical and thus, by their nature, somewhat formalized. The corresponding modeling languages make it relatively easy to create, archive, teach, and learn particular modeling applications and "tricks."

This difference between domain knowledge and analysis knowledge leads to what might be called a "semantic gap" that is a key issue in the practice of OR in production systems. The OR models and OR methods invariably rely on the semantics of mathematics and particular mathematical methods and may be influenced by the analysis modeling language and even the solver to be used, while the stakeholders invariably rely on the semantics of their domain and frequently find themselves incapable of directly evaluating the fidelity between the model developed by the OR analyst and the domain problem as they understand it.

Thus, the contemporary practice of OR in production systems requires the OR analyst or team to bridge this gap by using, and often creating, "modeling knowledge" to translate between the (natural) language of the stakeholders and the (formal) language of OR. The translation from "problem" to "formulation" tends to require significant investment of time for both analysts and stakeholders, is subject to interpretation errors, and is usually static, i.e., the resulting models may not accommodate changes in the modeled system. The translation from analytic results back to the stakeholder decision space also is largely the responsibility of the analysts, and likewise may be subject to interpretation errors. The test of analysis model fidelity often is simply

whether or not the analysis results "make sense" when viewed in light of the prior experience of the domain stakeholders.

It is safe to say that this modeling knowledge is the least codified of the three kinds of knowledge needed for OR-based decision support of production systems decision making. In fact, OR faculty have struggled, almost from the emergence of OR as a discipline, to discover an effective way for students to learn "how to model," which almost always means "how to extract a mathematical model of a process or decision from a somewhat ambiguous domain-specific problem description."

In the simple transportation problem illustration given above, the semantic gap is small and, one would hope, presents no great challenge to either the domain stakeholder or the OR analyst. Likewise, the modeling process itself seems straightforward, once illustrated. In more complex scenarios, the semantic gap becomes a larger problem, as does the challenge of modeling. For example, the creation of large scale optimization or simulation models to support the design and management of global logistics systems involves translating relatively arcane considerations, such as local content requirements, or export/import duties into precise mathematical relationships. Similarly, the development of large scale optimization models to design radiation therapies also involves translating what may be known with some ambiguity about the effects of radiation into a precise mathematical structure.

One contemporary approach to bridging the semantic gap is to create "parametric" analysis models which can accommodate any instance data conforming to the parametric definitions. For our simple example, this would give the decision maker the ability to specify the warehouses and customers, perhaps extracting the supplies, demands, and transport costs from appropriate data sources. This is an important step toward ubiquitous OR, but it obscures rather than resolves the semantic gap. Bridging the semantic gap still requires tacit knowledge that is not captured in a form that is transferable, reusable, teachable, and deployable. Moreover, the domain knowledge is encoded in the specification of the parametric data for the optimization formulation. In this form, the specification of the domain knowledge will be of limited value in supporting other relevant decision support models, such as simulation or risk analysis.

Effectively managing and exploiting these three kinds of knowledge—domain, analysis, and modeling—is the key to achieving a broader and deeper penetration of OR in production system decision making. This knowledge management problem has two fundamental challenges:

- How to capture each kind of knowledge in a form that is transferable, reusable, teachable, and deployable
- How to make the three kinds of knowledge interoperable, i.e., how to use modeling knowledge to support the transformation from instances of domain models—created using domain knowledge—to instances of analysis models in an appropriate computational form

In this regard, there is much to be learned from the experience of the software engineering community about knowledge representation and model transformation.

Model-driven Architecture

Until recently, software development has also been a largely artisanal activity. The image of the "hacker" is iconic in modern society—the idiosyncratic individual who can understand the nature of the needed computation, and craft an elegant code to make it possible. The limitation of the hacker model is the realized mismatch between the supply of hackers and the demands for software in modern society. The response of the software engineering community has been to "industrialize" the production of well-understood software applications (see, e.g., the evolution of BPMN (White 2006)). This industrialization is being accomplished by an evolving suite of theories, tools, and methods that permit individuals with less than "true hacker" credentials to create satisfactory implementations of the needed software. The essential nature of these tools and methods is that they capture both domain and software engineering knowledge in a form that is transferable, reusable, teachable, and deployable. The resulting "industrialization" of the artisanal software process is aptly captured in the term "software factories" (anonymous 2012a).

This movement in software engineering has been called "model-driven architecture" (MDA) (http://www.omg.org/mda/) or "model-driven engineering" (see, e.g., Meyers and Vangheluwe 2011). The fundamental enablers of MDA are formal modeling languages and model transformation theories and tools. The Unified Modeling Language (UML) (http://www.uml.org/) has evolved over the past 20 years to dominate modeling in the software engineering process. Emerging tools like the Object Management Group's (OMG) Query/View/Transformation (QVT) standard (http://en.wikipedia.org/wiki/QVT) enable the computational transformation of a model created with one language (syntax and semantics) to a model expressed in a different language. For example, the source model could be a UML-based description of a business process, and the target model could be the Java code necessary to provide the computational implementation of the business process.

Within systems engineering there is a growing community of researchers and practitioners who are adapting the tools and methods of MDA to systems engineering, calling it "model- based systems engineering" or MBSE (Ramos, et al., 2011). The language used most often in this community is OMG's Systems Modeling Language (OMG SysML^{TM}), which is an extension of UML to expand its modeling capabilities beyond software systems to address hardware, people, requirements, and parametric relationships (http://omgsysml.org/). A great deal of effort is being directed to understanding how to use SysML to model large scale, complex systems, incorporating multiple (discipline-specific) views, and integrating multiple analysis tools (Peak et al., 2009).

The approaches and experiences of MBSE present the OR community with two tantalizing opportunities. The first opportunity arises in situations where much is already known about using OR to answer particular kinds of questions in a particular domain, e.g., cycle time estimation in electronics manufacturing, production scheduling in aircraft assembly, or vehicle routing in package delivery. The opportunity is to package that knowledge together with a formal semantic model of the domain, and

deliver to domain stakeholders the capability to describe their problem—in its own terms, which they already understand—and get immediate and transparent access to appropriate OR analyses, without the direct intervention of an OR analyst. Simply put, the opportunity is to capture what we already know, and make it transferable, reusable, teachable, and deployable. Given the enormous collective repertoire of models and analyses, this is an opportunity to increase the reach and penetration of OR manyfold. Moreover, if both domain and OR knowledge are captured in formal semantics, they become much more easily taught and learned.

The second opportunity is to leverage the first opportunity to accelerate the creation of new and valuable OR-based knowledge, and its conversion to a transferable, reusable, teachable, and deployable form. If they are based on formal languages, domain-specific semantics can be elaborated to account for newly recognized problem domain elements or factors. New OR analyses, or enhancements to existing analyses could be more rapidly deployed by elaborating an existing infrastructure of domain specific languages and integrated OR analyses.

Formal Language and Knowledge Capture

The goal of capturing knowledge in a form that that is transferable, reusable, teachable, and deployable requires making knowledge explicit. Over the past 20 years, there has been a great deal of interest in methods to accomplish this, particularly in the context of information systems and the Internet. For example Vernadet (2007) has suggested the construction of ontologies as a way to achieve information systems interoperability through the use of metadata repositories. In the computing community, "ontology" usually implies the formal definition of classes representing concepts in a domain, properties of the classes representing features and attributes of the concept, and possibly restrictions on the properties (Dieng 2000). The ontology, together with instances of its classes, will constitute a "knowledge base." In this form, a knowledge base is machine readable, and can be manipulated using software.

There are many computational tools for authoring, editing, and visualizing ontologies (see, e.g., the techwiki page http://techwiki.openstructs.org/index.php/Ontology_Tools). However, these tools tend to be somewhat arcane and are often not easily accessible by application domain experts. A different strategy developed in the software engineering community and currently gaining traction in the systems engineering community is to create domain-specific languages (DSLs) that conform to a domain-specific ontology and thus are easier for domain experts to understand and use.

The language most commonly used by software engineers in the design of software applications is UML (http://www.uml.org/). UML is a graphical, object-oriented modeling language based on 13 diagram types which provide semantics for modeling application architecture, structure, and behavior, as well as business process flows, database, and message structure. A standards-based implementation of UML will include capabilities for elaborating the semantics, e.g., by further refining the

definition of generic objects or by adding new diagram types. For example, a generic object named "class" might be used to define new objects that are special kinds of "class," such as "machine_tool" or "transport_vehicle." These new objects might then be used by a domain expert to describe a particular application.

In 2007, OMG published a standard for a new modeling language, OMG SysML™, which is based on a subset of UML, and adds new diagram types specifically to support the modeling of complex systems incorporating software, hardware, and people (http://omgsysml.org/). A derivative of UML, SysML also is object-oriented and graphical. SysML supports the modeling of systems from multiple perspectives in a unified manner (Peak et al., 2009). It is a very expressive language for system modeling because it integrates the representation of structure (classes and the multiple kinds of relationships among them) and behavior (activities, state machines, and the sequence and timing of interactions among blocks).

Despite the relatively recent emergence of SysML, there have been a number of examples of its use in manufacturing (Huang et al., 2008; Batarseh et al., 2012), and supply chains (Thiers and McGinnis 2011; Ehm et al., 2011). Modeling an electronics assembly operation is described in Batarseh and McGinnis (2012), where the goal is to significantly reduce the time and cost of developing simulation models used to support production program planning.

In the system studied in Batarseh and McGinnis (2012), the assembly process starts with populated circuit card assemblies, to which hardware, such as connectors, will be assembled, and conformal coatings will be applied. The cards are then assembled into a chassis and additional coatings may be applied. Because the products have very high reliability requirements and may operate in extreme conditions, a large amount of testing is required, leading to significant amounts of rework. SysML was used to capture the semantics of the production process. Figure 2.3 summarizes the result. It illustrates the use of the "stereotype" facility of SysML to define new modeling concepts, e.g., refining "class" to specify a set of resource types, each with its own particular set of attributes. Specific instances of each resource type can be defined and stored in a library for ease of reuse. The stereotype facility also was used to define "part" and "final product" so that bills of materials could be created, and production schedules or requirements could be associated with final products. Finally, the types of processes required to produce a product were specified as stereotypes of the SysML "call action" object, and each different process type was given a set of appropriate attributes.

The domain expert would use these stereotyped objects, and perhaps libraries of their instances, to create both a bill of materials and a process plan for each subassembly and final assembly. A simple bill of materials is illustrated in Fig. 2.4 and a simple process plan in Fig. 2.5. These examples illustrate how the expressiveness of SysML can be exploited to create a graphical DSL that is easily accessible by the domain experts.

In this approach, two kinds of domain knowledge are captured in two distinct phases. First, the generic knowledge, the domain semantics, is captured using the stereotyping facility of SysML. This requires collaboration between domain experts and SysML modeling experts. In the second phase, the "use phase," the domain

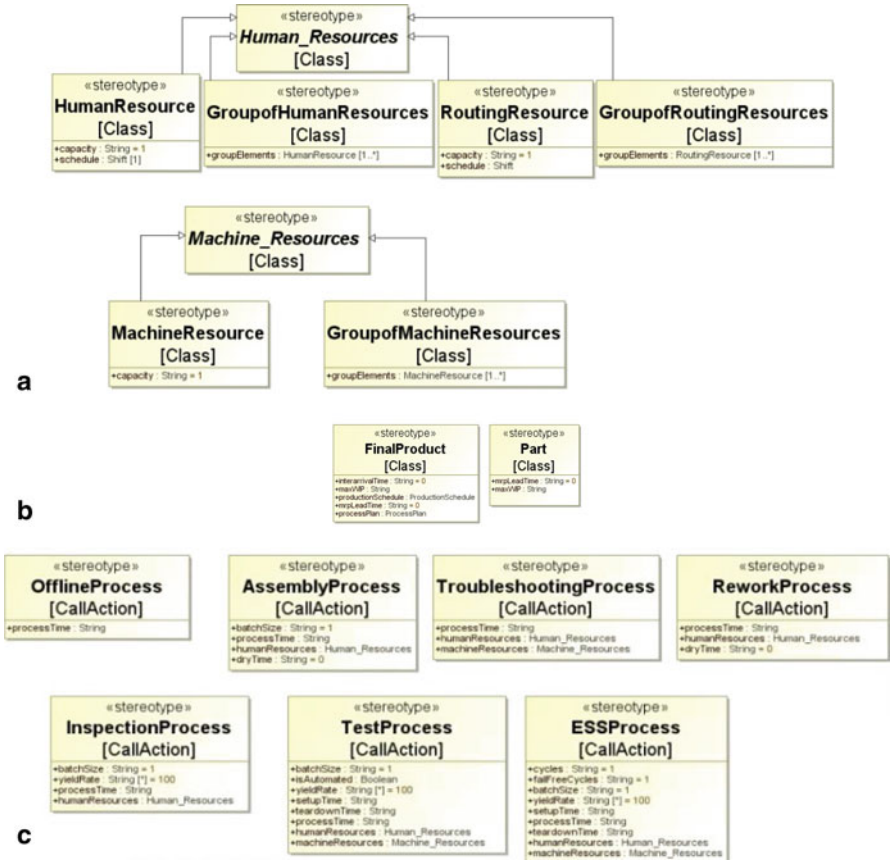

Fig. 2.3 Example of domain specific language semantics. **a** Resource semantics. **b** Product semantics. **c** Manufacturing processes semantics

specific language is used to capture knowledge of a particular application. One might reasonably ask, "how is this different from the usual OR study approach, where the OR analyst team works with domain experts to create the OR model?"

The difference, in fact, is quite significant. In the conventional approach, the knowledge captured about the domain is encoded in the OR model, severely limiting the opportunity to reuse this knowledge or to share it with other analysts. In particular, it makes it very difficult to reuse the knowledge for a different kind of analysis. For example, if the initial analysis used an optimization model, e.g., to establish capacity levels, a subsequent model using simulation, e.g., to size work-in-process buffers, would not be able to reuse the knowledge in a straightforward manner. With a DSL, reusable knowledge is captured both in the language itself, and possibly in every use of the language, as new information is added to libraries of similar objects.

http://www.springer.com/978-1-4614-9055-5

Essays in Production, Project Planning and Scheduling
A Festschrift in Honor of Salah Elmaghraby
Pulat, P.S.; Sarin, S.C.; Uzsoy, R. (Eds.)
2014, XIV, 414 p. 102 illus., 30 illus. in color., Hardcover
ISBN: 978-1-4614-9055-5