System Dynamics Review
System Dynamics Review (2010)
Published online in Wiley InterScience
(www.interscience.wiley.com) **DOI**: 10.1002/sdr.437

Construction by replacement: a new approach to simulation modeling

James Hines,[a] Thomas Malone,[b] Paulo Gonçalves,[c]* George Herman,[d] John Quimby,[d] Mary Murphy-Hoye,[e] James Rice,[f] James Patten[g] and Hiroshi Ishii[g]

Abstract

Simulation modeling can be valuable in many areas of management science, but it is often costly, time-consuming, and difficult to do. To reduce these problems, system dynamics researchers have previously developed standard pieces of model structure, called molecules, that can be reused in different models. However, the models assembled from these molecules often lacked feedback loops and generated few, if any, insights. This paper describes a new and more promising approach to using molecules in system dynamics modeling. The heart of the approach is a systematically organized library (or taxonomy) of predefined model components, or molecules, and a set of software tools for replacing one molecule with another. Users start with a simple generic model and progressively replace parts of the model with more specialized molecules from a systematically organized library of predefined components. These substitutions either create a new running model automatically or request further manual changes from the user. The paper describes our exploration using this approach to construct system dynamics models of supply chain processes in a large manufacturing company. The experiment included developing an innovative "tangible user interface" and a comprehensive catalog of system dynamics molecules. The paper concludes with a discussion of the benefits and limitations of this approach. Copyright © 2010 John Wiley & Sons, Ltd.

Syst. Dyn. Rev. (2009)

Introduction

Simulation models have been used with substantial success for decades in many areas of management science, from factory scheduling to financial forecasting, to supply chain planning to market analysis. One of the most important barriers to wider use of simulation modeling, however, is the difficulty of creating simulation models in the first place. Developing useful models often requires experienced modelers as well as significant amounts of, time, money, and effort.

In an attempt to improve quality and reduce the resources needed for modeling, system dynamics researchers (including Richardson and Pugh, 1981; Lyneis, 1980; Richmond, 1985; Hines, 1996) previously developed a number of standard pieces of structure (called molecules) to address frequently encountered modeling problems. The first formal compilation of system dynamics molecules (Hines, 1996) was based on

[a] Ventana Systems, Providence, RI, U.S.A.
[b] Sloan School of Management and Center for Collective Intelligence, MIT, Cambridge, MA, U.S.A.
[c] Faculty of Economics, University of Lugano, Switzerland.
[d] Center for Coordination Science, MIT, Cambridge, MA, U.S.A.
[e] Research Department, Intel Corporation, Chandler, AZ, U.S.A.
[f] Center for Transportation and Logistics, MIT, Cambridge, MA, U.S.A.
[g] Media Laboratory, MIT, Cambridge, MA, U.S.A.
* Correspondence to: Paulo Gonçalves. E-mail: paulo.goncalves@usi.ch
Received February 2006; Accepted September 2009

specialization inheritance, borrowing directly from object-oriented programming (e.g., Goldberg and Robson, 1989). Teachers who used these molecules in their intermediate system dynamics classes at MIT saw students' knowledge of standard system dynamics structures improve. However, the derived models often lacked feedback loops and generated few, if any, insights.

This paper describes a new approach to using molecules in system dynamics modeling. The heart of the approach is a systematically organized library (or taxonomy) of predefined model components (or molecules) and a set of software tools for progressively replacing one molecule with another. The framework allows users to rapidly build system dynamics models by replacing parts of models with more specialized versions of the same parts. For instance, the framework lets users successively develop models from causal loop diagrams, a process that has been traditionally difficult to teach effectively to students. We demonstrate the framework's capability by building one useful system dynamics model, and we describe a general approach to applying the framework in other situations.

While related ideas have been used in previous simulation systems for years (e.g., Goldberg and Robson, 1989; Meyer, 1992; Fleishman and Hemple, 1994), this approach takes the basic ideas further than any previous efforts. For instance, our approach has "designed-in" the composability of molecules and, by extension, models. This designed-in composability helps assure that assembled and recombined components satisfy user requirements meaningfully (Davis and Anderson, 2004).

In addition, the molecule hierarchy proposed here allows users to see the structural connections among molecules when they are deciding which replacements to make. This ability to see structural connections makes it easier for users not only to create new molecules but also to see where one molecule can be replaced with another one. Moreover, the molecule hierarchy has been designed to allow molecules to fit together in a seamless way after such replacements have been made.

Finally, the hierarchy makes it especially easy to rapidly construct simulation models using either a conventional graphical user interface (i.e., a mouse and screen) or a novel "tangible user interface" (Ishi and Ullmer, 1999), where users manipulate actual physical objects on a special table.

While this approach is not a "magic bullet" that makes the creation of simulation models instantaneous and effortless, it has the potential to significantly increase (a) the speed with which new simulation models can be created, (b) the "correctness" of those models, (c) the number of people who can create simulation models for themselves without requiring the assistance of professional programmers and modeling experts, and (d) the use of simulation modeling for facilitating conversations and collaboration.

To develop our approach, we conducted a substantial multi-year investigation to construct system dynamics simulation models (e.g., Forrester, 1961; Sterman, 2000) of corporate supply chains. This paper summarizes the results of that investigation. It describes (1) the comprehensive library of system dynamics molecules we developed, and (2) the software tools we used to combine and refine these molecules. Of particular interest is the fact that our library of system dynamics molecules constitutes a "periodic table" of the elements used in constructing any system dynamics model. In addition, since the library is open-ended, new combinations of these elements can always be added as they are identified. In fact, our process for constructing models actually aids

DOI: 10.1002/sdr

in placing new molecules in their proper places in the "periodic table". We illustrate the use of this approach with a hypothetical usage scenario based upon our extensive analysis of the supply chain of a large manufacturing company (further described in Gonçalves, 2003; Gonçalves *et al.*, 2005). The paper concludes with lessons about how this approach can be applied with other kinds of simulation and in other situations.

Background

Before describing our approach in more detail, it is useful to review the fundamental idea of reusable components in computer science as well as the difficulties in the current practice of simulation modeling.

Reusable software components

Computer scientists have noted for decades that different computer programs in the same general domain often have many commonalities. In fact, much of the progress in computer science can be seen as successive ways of capturing these commonalities in reusable tools like compilers, operating systems, subroutine libraries, and graphics packages. In this way, different programmers can reuse the standard features these tools provide, rather than having to reinvent them each time the features are needed. In computer science, such general reusable solutions are often called design patterns, after the architectural concept coined by Christopher Alexander (1977, 1979). Design patterns gained currency in computer science after the book *Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma *et al.*, 1995) was published.

While most of these reusable components have been organized in "flat" collections, one particularly important approach, *object-oriented programming*, employs a "deep" approach. In an object-oriented environment such as Smalltalk, C++, or Java, the reusable components are classified in an *inheritance hierarchy* where "child" components automatically "inherit" properties of their "parents". The specialization relationship is called an ***is–a*** relationship. "*A* ***is-a*** *B*" means that *A* is a *specialization* of *B*, and *B* is a *generalization* of *A*. For instance, a "car" ***is-a*** "vehicle," and "vehicle" is a generalization of "truck", "car", and "bus." In an ***is-a*** relationship, the "child" class *inherits* all the properties of its parent. For instance, cars inherit properties from vehicles, such as moving in a certain way.

Even though the goal of creating simulation models was historically important in the development of some of the first object-oriented programming languages (e.g., Simula and Smalltalk), most modern simulation languages still use collections of "flat" components from which programmers can choose using icon-based mouse-enabled graphical user interfaces.

Difficulties in the current practice of simulation modeling

Even though simulation models have the potential to be extremely useful, they are often difficult to create. For example, most people can create for themselves only the simplest and least flexible form of simulations: spreadsheets models. Professional modelers or programmers are almost always needed to create other common kinds of simulations (e.g., discrete event, system dynamics, and agent-based simulations).

DOI: 10.1002/sdr

The difficulty of creating simulation models is also reflected in high costs and error rates. A small professional system dynamics effort, for instance, will typically cost from U.S. $25,000 to $100,000; large efforts can range into the millions (Dalton, pers. comm. 2003; Eberlein, pers. comm. 2003). As for error rates, Panko and Haverson conclude in their survey of studies of spreadsheet models that "every study that has looked for errors has found them in significant numbers" (Panko and Halverson, 1996, p. 4).

Exacerbating the problem of the high cost of model creation is the low opportunity to amortize the investment over multiple problems or questions. In fact, surprisingly few models, including very costly ones, are ever employed again after the original problem has been solved. This means that the entire financial burden of building a new model must be borne by the potential benefit of solving only the *current* problem.

The effort to compile a hierarchy of "molecules" capturing standard structures in system dynamics had its motivation in object-oriented programming (Hines, 1996). The goal was to allow users to create simulation models faster, more accurately, and with more reusability. However, even before the explicit effort to create a hierarchy of molecules, standard structures were developed in seminal papers and tended to get repeated in models without attribution. Many of the first standard structures (or "molecules") appeared in Jay Forrester's writings in system dynamics (Forrester, 1958, 1961). Industrial dynamics (Forrester, 1961) had molecules built into its DYNAMO software (e.g., level equations, first- and third-order smoothes, and material delays). Also important was the market growth model created by Dave Packer working with Jay Forrester (Forrester, 1968). In addition, the project model, originally developed by Henry Weil, Ken Cooper, and David Peterson around 1972, contributed a number of important molecules. Jim Lyneis' book (Lyneis, 1980) contained important structures for corporate models. Equally important was the treasure trove of good structures in the MIT National Model, to which many people contributed, including Alan Graham, Peter Senge, John Sterman, Nat Mass, Nathan Forrester, Bob Eberlein, and of course Jay Forrester. George Richardson and Jack Pugh described a number of commonly occurring rate and stock equations in their excellent book (Richardson and Pugh, 1981). Adopting the term "atoms of structure", Barry Richmond (1985) described a number of common rate structures in his paper describing STELLA, the first graphical system dynamics modeling environment. Barry Richmond and Steve Peterson continued to present useful small structures in the documentation for STELLA and its sister product iThink. And, misremembering the term "atoms of structure", one of the authors of this paper used the term "molecules" in his initial attempts to extend and categorize these structures (Hines, 1996).

As the description above suggests, the idea of molecules per se is not new in system dynamics. However, the effort to compile them in a hierarchy is. The original molecule hierarchy (Hines, 1996) allowed more people to have access to standard structures disseminated in the system dynamics literature. Because the original molecule hierarchy emphasized a *specialization inheritance* and in particular ***is-a*** relationships, it was very effective for teaching purposes. For instance, if students understood the concept of a parent (e.g., a vehicle), it was much simpler to grasp new concepts for its children (e.g., cars or trucks). However, while the original molecule hierarchy helped improve students' knowledge of standard system dynamics structures and accelerate their ability to build system dynamics models, the resulting models often lacked feedback and generated few, if any, insights.

DOI: 10.1002/sdr

While it was possible to explore the *replacement* aspects of *inheritance* in the original molecule hierarchy, this traditionally was not done. Also, a *replacement* hierarchy requires the ability to "plug-and play" molecules (where a child that replaces its parent still allows the model to run). The original molecule hierarchy was not designed for this purpose, and attempts to use it in this way would have required major redesign.

The current version of the molecule hierarchy has undertaken such redesign. It expands the original set of molecules and facilitates *replacement* though a new taxonomy showing the connections between molecules. This new taxonomy also purposefully helps users see the structural connections between molecules. Our experience with this new taxonomy is limited (as described below, we built only one major model). But based on this experience, we believe that this taxonomy has the potential to substantially reduce the time, cost, and effort in creating system dynamics simulation models.

Our approach

The key to our approach is making it especially easy for people to refine and combine predefined components (or "molecules") into new models. The three prerequisites necessary for our approach are (1) a systematically organized catalog of predefined molecules, (2) automatic tools to help users replace parts of an existing molecule with more specialized versions of the same parts, and (3) automatic tools for storing and cataloguing new molecules.

A systematically organized catalog of predefined molecules

In chemistry, a molecule consists of a certain number of more elementary parts, either atoms or other molecules, and a set of linkages between these parts (i.e., chemical bonds). Similarly, in our approach to simulation modeling, a "molecule" consists of a number of more elementary parts, which are themselves (simpler) molecules, and a set of linkages between these parts. For example, a simple supply chain model might include molecules for production planning, manufacturing, assembly, and shipping finished goods. The molecule for production planning might, in turn, include molecules for storing materials in a warehouse and placing orders when the warehouse inventory levels reach a certain point. We require that all individual molecules be "run-able" *so technically a molecule is a simulation model that can be a component in a larger simulation model*. Because any model can be a component in a larger model, it is also true that *a model is a molecule.* We will use the term "molecule" when we wish to emphasize the building block nature of things, and we will use the term "model" when we wish to emphasize a usefulness beyond "just" being a component of something larger.

Before describing how we systematically organize catalogs of molecules, it is useful to see some detailed examples of molecules. Since we have applied our approach using system dynamics models, we will use system dynamics molecules for this purpose. System dynamics models are basically systems of nonlinear differential equations. In the system dynamics graphical notation, a stock (mathematically, an integral) is represented as a rectangle, a stylized bathtub. A flow (i.e., a partial derivative with respect to time) is represented as a double arrow, a stylized pipe. A policy (decision

DOI: 10.1002/sdr

rule) controlling a flow is represented as a stylized "valve" (often depicted as a simple hourglass shape) on a pipe, and sub-policies are represented as labels connected by information links depicted as skinny, curved arrows ("telephone wires"). All flows are conserved; that is, every flow comes from one stock and goes into another. When the stock in question is beyond the scope of a model, a cloud, instead of a rectangle, is used. Figure 1 illustrates the symbols.

As an example, Figure 2 shows the system dynamics *material delay* molecule. A *material delay* is a relatively low-level component that is available as a function in most system dynamics simulation environments. Very simply, a material delay allows a modeler to create a flow (outflow) that is a delayed version of another flow (inflow).

Mathematically, a material delay is a first-order linear, non-homogeneous, fixed-coefficient differential equation. It is defined by the following equations:

$$\frac{\mathrm{d}}{\mathrm{d}t} Stock_t = inflow_t - outflow_t$$

$$outflow_t = \frac{Stock_t}{timeConstant}$$

Dynamically, the *outflow* is an exponentially smoothed and exponentially delayed version of the inflow, which comes from elsewhere in the model. A stock accumulates the difference between the inflow and the outflow, ensuring that everything that goes in eventually comes out and nothing more. As an example, a modeler might use a material delay to represent the lag in realizing cash from accounts receivable. Dollar sales would be the inflow into the *stock* of accounts receivable. Flowing out of accounts receivable (and into the stock of cash) would be "cash flow", a delayed version of dollar sales equal to accounts receivable divided by the average delay (the *timeConstant*).

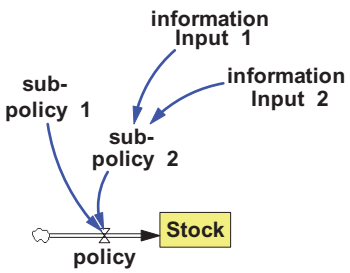

Fig. 1. Symbols used in system dynamics stock-and-flow diagrams

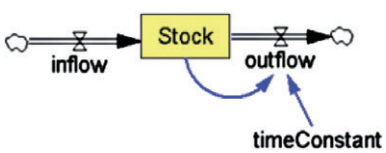

Fig. 2. System dynamics *material delay* molecule

DOI: 10.1002/sdr

Arranging molecules in a specialization hierarchy

Having predefined, reusable components can be useful in almost any software development endeavor and in particular in the creation of computer simulation models. The power of our approach, however, depends on converting these flat components into *deep* components that are related to one another in a very specific, systematic way.

In our case all the predefined molecules are arranged in a *specialization hierarchy* where each item is classified as a subtype (a kind of specialization) of one or more other items. Each item can also, in turn, have its own subtypes. In fact, it is possible to classify any simulation model, regardless of its complexity, somewhere within this specialization hierarchy, and all of its component parts can also be classified somewhere within the hierarchy too. (See Malone *et al.*, 1999, for an extensive description of the type of specialization hierarchy we use.)

For example, Figure 3 shows a unified modeling language (UML) class diagram of a small subset of the specialization hierarchy around the material delay molecule described above. Readers can find a full description of the replacement hierarchy at web.mit.edu/~paulopg/www/.

The basic element is the system dynamics (SD) molecule. The basic SD molecule has three important subtypes: stocks (i.e., accumulations), flows (which fill or deplete stocks), and policies, which control flows (cf. Forrester, 1961, pp. 93 ff.). To understand the hierarchy in more detail, it is useful to focus on a single chain (see, for example, Figure 4).

A stock can have any number of inflows and outflows. Figure 5 shows that a bathtub is a specialization of a stock. In fact, a bathtub is a stock that has a single inflow and a single outflow.

The material delay (as shown in Figure 2) is a specialization of a bathtub. The material delay uses a *specific* outflow, namely a decay outflow. The decay outflow is itself a molecule, with its own place in the hierarchy (see Figure 3).

An aging chain is a disaggregation of a (first-order) material delay into an *n*th-order one, where each outflow from a sub-material delay flows into the next sub-material delay. For example, a third-order material delay appears in Figure 6. Each of the disaggregated time constants would normally be set to one-third the original timeConstant.

Finally, a modeler might specialize the aging chain by changing the names and units and by giving different values to each of the disaggregated time constants. In so doing, the modeler might create a crude representation of a sequential supply chain (Figure 7).

This specialization, created by a modeler, is a running model and has its own place in the specialization hierarchy (see Figures 3 and 4).

One advantage of arranging components in hierarchies like this is the resulting ease with which users can find the components they need, even in very large collections of components. A user who understands the specialization hierarchy will know, for example, that an aging chain must be near the material delay and the material delay must be near the bathtub. Another important advantage of a hierarchy is that it simplifies the creation of new models by allowing one to build up a more specialized (often larger) model by replacing elements in a less specialized (often smaller) one.

DOI: 10.1002/sdr

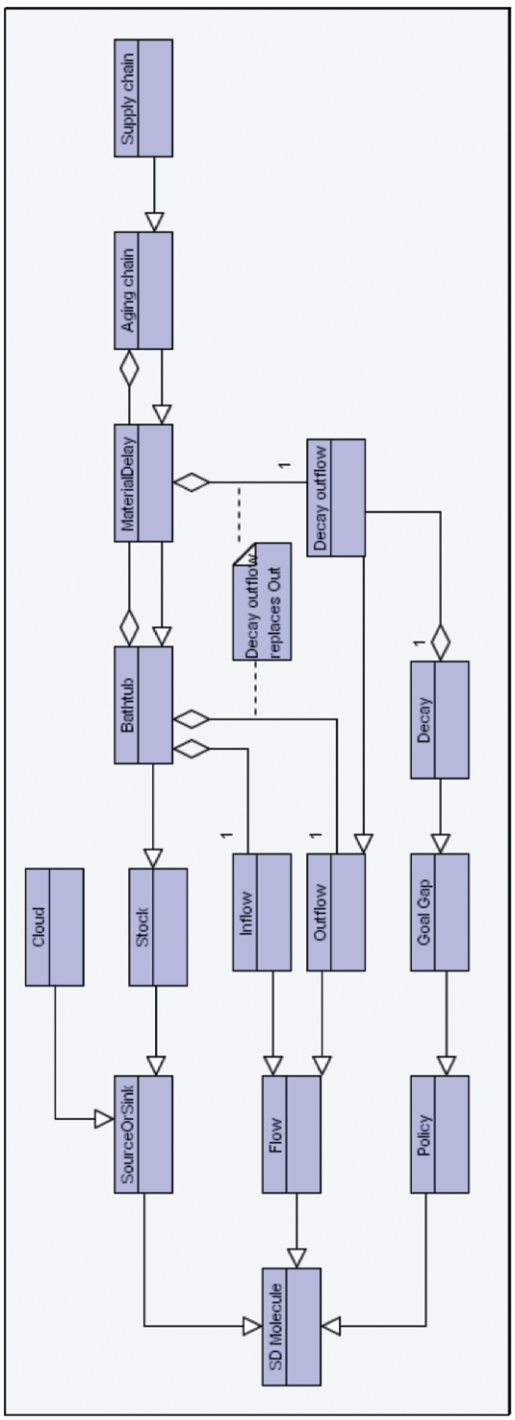

Fig. 3. Specialization hierarchy surrounding the material delay. *Note*: In a UML class diagram, classes are represented as rectangles whose first line gives the name of the class. A class is connected to its superclass by a line terminating in a triangle (i.e., the triangle marks the superclass). A class that is used in the composition of another class is connected to that class by a line terminating in a diamond (i.e., the diamond marks the containing class)

DOI: 10.1002/sdr

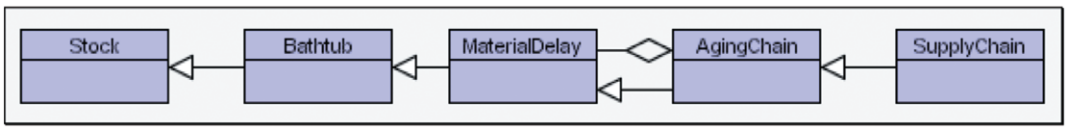

Fig. 4. A small "chain" of the specialization hierarchy

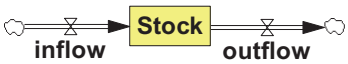

Fig. 5. The bathtub molecule

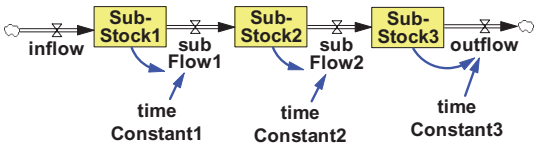

Fig. 6. The aging chain molecule

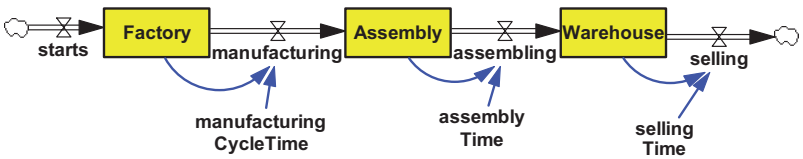

Fig. 7. A simple sequential supply chain

Modeling by replacement

Readers familiar with the notions of inheritance in object-oriented programming will immediately see the similarities between these concepts and the concept of specialization as we are using it here. Our use of specialization, however, differs in two important ways from the way that inheritance is typically used in most object-oriented programming languages. First, the items that are specialized in typical object-oriented programming are often "objects", not "actions". In many simulation models, however, the items of primary interest are the actions ("verbs") whose effects are being simulated, not the objects ("nouns") upon which those actions operate. Our specialization hierarchies inherit down a hierarchy of "verbs" as well as "nouns", and this provides substantial power and flexibility for creating simulation models. (See Malone *et al.* (1999) and Lee and Wyner (2003) for extensive discussions of this issue.)

In addition, our approach requires that the specialization hierarchy of molecules be arranged in such a way that replacing a molecule with any of its specializations still results in a mathematically and conceptually valid simulation model.[1] For example, in a system dynamics model, you can always replace a *bathtub* with an *aging chain* and

DOI: 10.1002/sdr

still have a valid model. In other words, the specialization hierarchy must have the following formal property (see, for example, Liskov and Wing, 1994; Lalond and Pugh, 1991):

> *Substitution property*: If S is a mathematically valid molecule whose parts are the molecules M_i (for $i = 1, \ldots, n$), and M_i' is a specialization of M_i, then replacing M_i with M_i' in S results in a molecule S' which is also a mathematically valid molecule.

When this property holds, users can refine subparts of a model and have the new subparts automatically substituted into the overall model. Naturally, the user must still choose a specialization that is an appropriate representation of the environment being modeled. While our approach cannot guarantee that users will choose appropriately, the list of specializations provides a framework which allows users to compare the resulting models with the environment. The challenge for inexperienced modelers shifts from "How do I model this decision?" to "Which specialization is closest to this decision?"

The specialization hierarchy makes molecule refinement straightforward. Users can select any element of a model (e.g., by clicking on it), and immediately see all the possible specializations of this element. Then, if the user selects one of these specializations and invokes the "replace with specialization" action, the system automatically substitutes the specialized version of the element in place of the original version. For example, a financial model might represent accounts payable as a material delay. A click would replace the material delay with an aging chain, in order to separate payable accounts into "buckets" of different ages.

In many cases, the system can automatically make all the necessary connections so that the new model is a completely valid simulation model. In the example immediately above, the inflow to the old material delay would be the inflow to the new aging chain. Any component that used the outflow from the original material delay would receive instead the outflow from the new aging chain. Components that depended on information about the single stock of receivables in the old formulation would automatically get information on the sum of the buckets in the aging chain. When the system cannot itself make the necessary connections, however, it can at least automatically call the user's attention to the places where further actions are needed to make the model a mathematically valid one. For example, when replacing a policy with a goal-gap, there may be more than one candidate for the goal. In this case, as discussed below, the system will create a "reaction object" (a sort of place-holder), which the user can easily connect to one of the valid alternatives in the model. Once again, the user must choose the appropriate goal from the set of valid alternatives. While there is little doubt that modeling experience will help in the choice, the list of alternatives allows users to easily compare the choices.

This process of replacing parts of a model with more specialized versions of the same parts can, of course, be repeated many times in different parts of the same model. It is often desirable, for instance, to make a substitution in one part of an overall model and then make further substitutions inside the subparts (i.e., "sub-molecules") of the molecule that has just been added. In this way, users can create arbitrarily complex models simply by making repeated substitutions in a single starting model. At each step along the way, they have a valid model, and all they ever have to do is select from the

DOI: 10.1002/sdr

alternatives that are automatically presented to them by the system. They never have to write a single line of textual specification ("programming") as they would in almost all other simulation environments today.

Library expansion as a byproduct of modeling

Replacement hierarchies can be formed so that modeling by replacement creates an expanding library of molecules available for future modeling efforts. All that is required is a natural extension of the *substitution property*. If we view the tree of all specializations of a molecule as a set, then the extension is simply to ensure that the set of specializations is closed under the operation of replacement:

> *Substitution property with closure*: If S is a mathematically valid molecule whose parts are the molecules M_i (for $i = 1, \ldots, n$), and M_i' is a specialization of M_i, then replacing M_i with M_i' in S results in a molecule S' which is also a mathematically valid molecule *and which is a specialization of S*.

In other words, the new molecule S' can be immediately "shelved" under its generalization, the old molecule S. This cataloguing and storage function is easily automated. For example, say we change the sequential supply chain model of Figure 7 by substituting a specialization of the outflow. We immediately create a specialization of the original supply chain model. The specialization is located right "beneath" the original supply chain model. If we have a larger industry model of which the original supply chain model was a component, we can now replace the original supply chain model with its new specialization. This will create a new specialization of our industry model, which, in turn, could replace the old industry model in a yet larger model of an economy.

The use of deep components with the closed substitution property creates a rapid and less error-prone process of model creation that continually produces new, properly catalogued specializations of prior molecules, and which ultimately results in the model itself also becoming part of the specialization hierarchy, properly catalogued and available for future use. For many common simulation approaches (including system dynamics), we believe it is possible to construct "complete" taxonomies from which any possible mathematically valid model can be constructed by making successive replacements in the way just described. The approach we have described can still speed and simplify some (often, most) of the model creation task, even when such a complete taxonomy cannot be constructed.

Implementing the approach

To develop and test this approach, we applied it to system dynamics simulation models of supply chains. We also used a new generation of *tangible* user interfaces (Patten *et al.*, 2001), which we describe briefly below. The supply chain domain was an obvious choice. First, the manufacturing company with which we worked closely in this project is known for its supply chain expertise, and our closest associates at this company included people with significant knowledge of the supply chain. Second, the focus on supply chains permitted us to apply our ideas in an area where the need for better alignment and integration is widely recognized by both managers and academics.

DOI: 10.1002/sdr

An important goal of our project was to develop simulation environments that could be easily used, not just with traditional graphical user interfaces but also with a new generation of "tangible user interfaces" (Ishii and Ullmer, 1999). In general, tangible user interfaces move beyond pointing to words and pictures on computer screens and, instead, let users see and manipulate three-dimensional physical objects in the real world. Such interfaces can have several of these three-dimensional physical objects. Consequently, tangible user interfaces seem particularly attractive for the kind of collaborative model building that may prove useful in domains such as supply chains.

We chose to use system dynamics models in this project for four reasons. First, system dynamics possesses few primitive components (e.g., stocks, flows, and policies) so it seemed likely that the process of replacement would usually work in a system dynamics model. Second, a number of common modeling structures in system dynamics are already recognized within the field. These common formulations make a good start on a comprehensive set of molecules. Further, system dynamics is particularly well suited to the central challenge facing people who manage and study supply chains: understanding and improving the performance of a system considered as a whole. Finally, the notion of collaborative model building is already well established within the system dynamics field (e.g., Vennix *et al.*, 1997).

The particular tangible user interface developed in this project, known as the Sensetable, allows users to move special physical objects (called "pucks") around on a special table that senses the location of the pucks, while computer-generated colors, words, and pictures are projected from above onto the pucks and table (Figure 8). While our approach to developing simulation models does not depend on using such a tangible user interface, we believe our approach is especially well suited to taking advantage of this new generation of computer user interface.

In order to apply our approach here, we needed to develop the three key elements of our approach described in the previous section: *a systematically organized library of molecules*, *a way of replacing parts of molecules*, *and a way of automatically cataloguing new molecules into the library.*

A systematically organized library of predefined molecules

To develop the library of molecules for system dynamics models, we started with an earlier hierarchy of 50 common components of system dynamics models (Eberlein and Hines, 1996; Hines, 1996).

Because this earlier hierarchy did not strictly enforce the substitution properties described above, our first step was to reorganize the molecules into a specialization hierarchy with the property of substitution with closure. At the "top" of our new replacement hierarchy we put three basic *types* of molecules: *Stocks*, *Flows*, and *Policies* (see Figure 3). Stocks are accumulations of physical things or information; Flows carry physical things or information into and out of stocks; and Policies are the decision rules which control the flows.

A new hierarchy reveals new molecules

With these three fundamental categories, we turned to categorizing the 50 original molecules and discovered that the pre-existing structure had some significant leaps in degree of abstraction. For instance, the *Stock Protected By Stock* molecule differed from

DOI: 10.1002/sdr

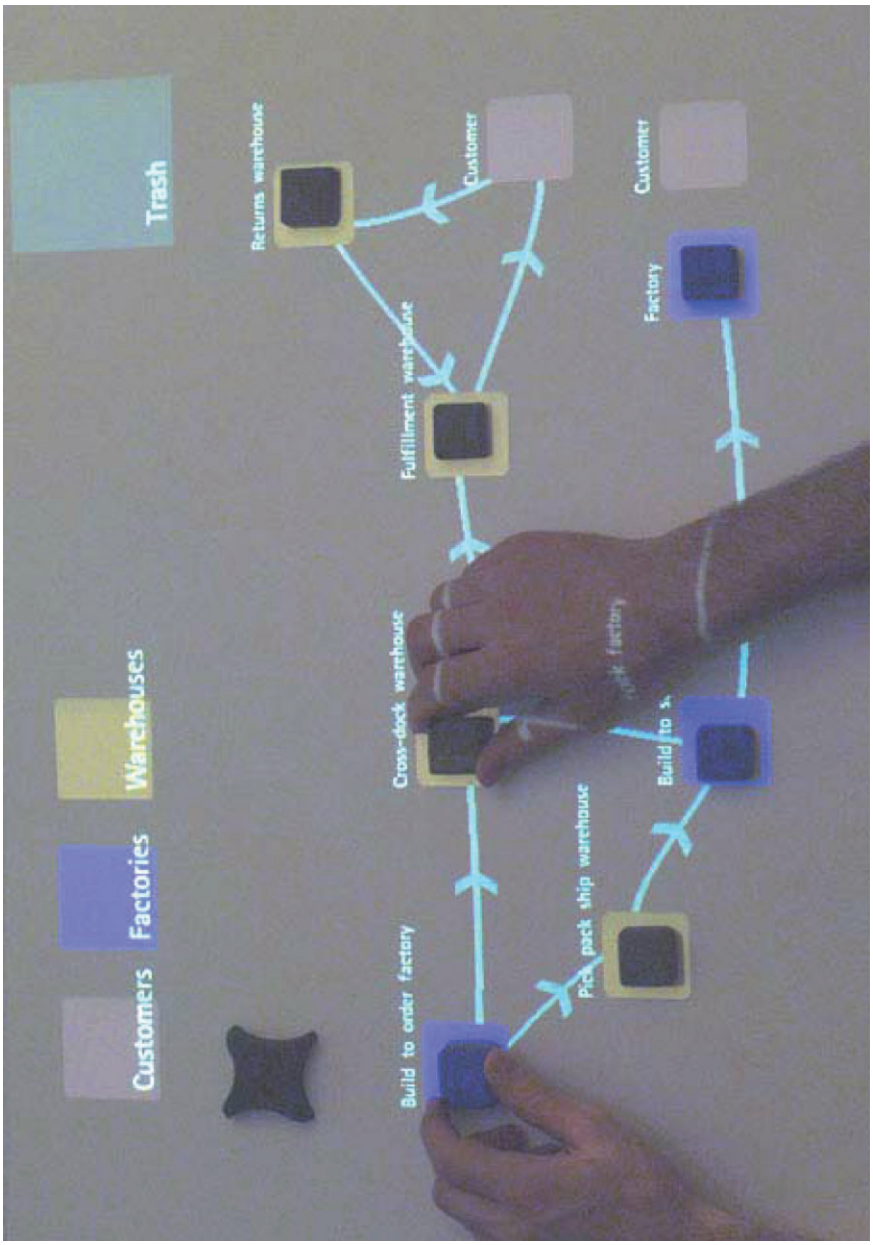

Fig. 8. The tangible user interface

DOI: 10.1002/sdr

a *stock* molecule by defining an outflow that was kept below a maximum value in order to ensure that the flow would not take the stock below zero. The maximum value for the flow was calculated as a user-defined function of the stock: $Outflow_t = f(Stock_t) \times IndicatedOutflow_t$. The function $f(\cdot)$ equals 1 as long as the stock is above a critical value (the "desired stock") so that, when unconstrained, the outflow is equal to the indicated outflow. Below the critical value, the function goes to zero as the stock goes to zero. In the new hierarchy, the direct connection between *stock* and its child would have meant that this particular outflow type would descend directly from an undifferentiated flow. But a number of other kinds of flows—less general than an undifferentiated flow, but more general than this particular flow—could be conceptualized as intervening: *Flow outflow* → *Outflow Below Maximum* → *Outflow Protected By Stock*.

As illustrated in Figure 9, the new structure "opens up" the hierarchy so that other molecules can be inserted in their rightful place by asking the question: How else do system dynamics modelers represent outflows that are below a maximum? A pre-existing molecule, *Outflow Protected By Flow* easily fits, and it was moved from its prior parent, *Decay*. We realized another formulation—*DrainToZero*,[2] which drains a stock until it is zero and then stops draining—was also widely used, even though it had escaped notice during the earlier attempt at hierarchy building. This was one way in which the new hierarchy fostered the identification of new molecules.

Another way our replacement hierarchy helped us discover missing molecules involved the idea of collectively exhaustive specialization (CES). Because replacement (or subtype) hierarchies are based on meaning or concept, one can ask the question whether a set of specializations covers the entire concept represented by their common parent. For example, having the molecule *Outflow Below Maximum* raises the question whether there should be a molecule for *Outflow Above Minimum*. We believe such a molecule would not be found in most practitioners' mental warehouses of tried-and-true structures. Nonetheless, a formulation can be easily created and is actually useful in representing, say, a container (e.g., a warehouse) of fixed volume. When the container is full, the outflow has to be at least equal to the inflow. CES, in this case, led to the creation of a "new" molecule—one that was not widely recognized before this work.

As we applied these processes of filling in the chain and looking for collectively exhaustive specializations, the original set of 50 molecules grew to over 200. By

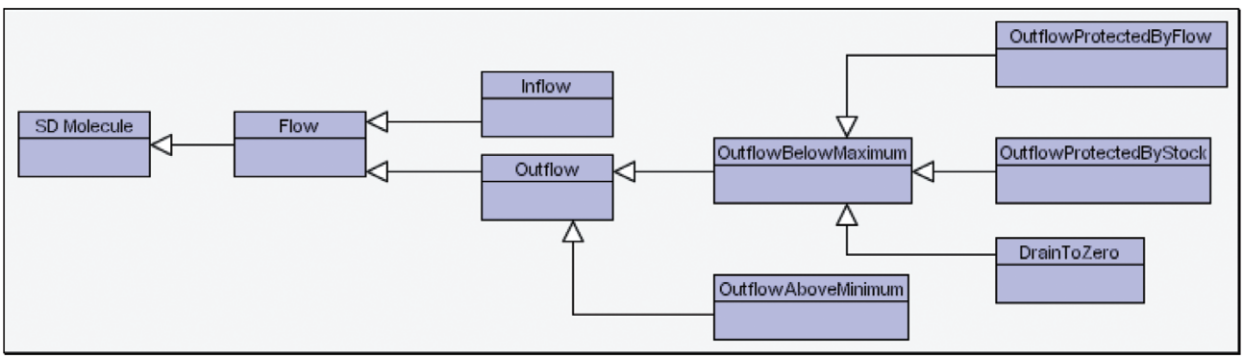

Fig. 9. *Outflow Protected By Stock* and a portion of the new taxonomy

DOI: 10.1002/sdr

systematically organizing molecule types into a replacement hierarchy, we created knowledge about the range of possible elements in a simulation model. In this sense, our approach is similar to the periodic table of the elements in chemistry, which highlighted the potential existence of new elements even before they were discovered.

Obviously, 200 molecules would overwhelm a flat-component, icon-based architecture. Interestingly, the same taxonomic system that allowed us to expand the number of molecules also keeps them ordered and available to anyone familiar with the taxonomy. In this use, the taxonomy is something like the Dewey decimal system. If you know the kind of molecule you need, you can go to the proper shelf to find it; and, if the molecule is missing, you know that specifying it will be a contribution to the field.

Modeling by replacement

The hierarchy of system dynamics molecules was stored in a systematically organized online knowledge base called the *Process Handbook* (see Malone *et al.*, 1999, 2003), which already included extensive facilities for manipulating and viewing textual and graphical descriptions of processes arranged in specialization hierarchies and a preexisting library of over 5000 business activities and processes. The *Process Handbook* also already included capabilities for replacing an element in a business process with one of its specializations by simply selecting from a menu of the possible alternatives. As part of this project, we augmented these existing capabilities of the *Process Handbook* with additional capabilities to store and manipulate mathematical equations and to display system dynamics models using stock-and-flow symbols.

In addition, the *Process Handbook* can store substantial information about each of the alternative specializations of an item. Thus the handbook can prompt users who do not immediately know which choice they want to make for a given replacement. We extended this capability to store molecule-relevant information, such as units as well as information about *how* a molecule can replace a parent. For example, when replacing an original bathtub (Figure 5) with an aging chain (Figure 6), the handbook "knows" that the inflow to the original bathtub should become the inflow to the aging chain and that the aging chain's outflow should replace the outflow of the original bathtub. The handbook also knows to propagate the physical units (e.g., 'barrels of wine') as well as the time unit (e.g., 'months') from the original bathtub to the new aging chain.

In many cases, as soon as a user selects a replacement, the system automatically makes all the necessary connections so that the resulting model is conceptually valid and completely functional. In other cases, the system makes most of the necessary connections and substitutions, but additional user action is required to make a completely functional model. In these cases, the system creates and displays one or more of what we call "REAction objects" (short for "Required Editorial Action objects"). For example, when a modeler replaces a *Stock*, representing an inventory, with a *Monitored Stock* (i.e., one with a goal attached), information concerning the gap between the stock and its goal could go to the downstream supplier, the upstream pricer, or both. The REAction object focuses the modeler's attention on that choice.

As part of this project, we implemented all the capabilities we have just described for tangible user interfaces (TUIs) (Patten *et al.*, 2001) as well as graphical user interfaces (GUIs). In TUIs, for instance, instead of showing menus of alternative replacements on

DOI: 10.1002/sdr

a screen and letting users make selections with a mouse, the menus are projected onto the special table, and users make selections by moving a special "puck" on the table.

Library expansion as a byproduct of modeling

As one molecule is replaced by another, a series of specializations are made. Each new specialization is itself a molecule. Because the molecule resulted from a specialization, the system has the information of where to place the molecule in the hierarchy at the time the molecule is created. Our system automatically places every new molecule into its proper place in the hierarchy.

Scenario of use

To help visualize the usefulness of this approach, imagine the following scenario, based upon our analyses of actual supply chain issues in the large manufacturing company we studied. Since our system is not yet robust enough for daily use in remote sites, the scenario described here is a hypothetical description of how a system like ours could, in the future, be used in practice. The specific characters and events are fictional.

The scenario involves three people from a semiconductor manufacturing company: Manny, the manufacturing manager; Polly, the planner; and Warren, the warehouse manager. Their collaboration is central to performance, but they seldom find a time or a setting conducive to that collaboration. Earlier Manny had confronted Polly with a disturbing pattern of dramatic oscillations in plant utilization: back and forth from very heavy to very light. Polly responded by saying she was reacting to erratic requests from Warren in the warehouse. Warren, reached by telephone, reported that he frequently had to scramble because of the unpredictable, stop-and-go nature of deliveries from Assembly.

The three managers decide to meet in the "war room", a converted conference room that is the home of a system similar to the one we have developed. The most visible part of the system is the equipment for the TUI: a medium-sized table with built-in sensor technology, an LCD projector mounted from the ceiling projecting onto the table, and a box of small disks (about 1.5 inches in diameter), called "pucks".

Polly begins by putting a puck down on the table to represent the beginning of a model of the company's supply chain. The system projects onto the table several possible specializations of this generic element, and Polly picks a bathtub (see Figure 5, above). The symbol for a bathtub is then projected on the puck. Next Polly says that the stock represents all of the stock in the company from manufacturing through assembly and including the warehouse. She specifies the units by typing "chips" on a keyboard. The system then asks her what units she wants to use to measure time. She chooses "weeks", and the system automatically sets the units on the inflow and outflow to be "chips per week". Polly then turns the knob on top of the puck to set the initial value of the chips in the system. She guesses that there are 20 million chips in the system. Then she takes a new puck from the box, puts it on the inflow valve, and turns the knob to represent an inflow of 900,000 units per week, saying "That's about what I'm starting right now." Next Warren takes another puck from the box, sets it on the outflow, and dials in 1 million chips per week, explaining that that was about the current rate of shipments.

DOI: 10.1002/sdr

Now, Manny says he would prefer to see his own manufacturing plant separated from the assembly plant and from the warehouse, so he *replaces* the bathtub with an aging chain, one of the specializations of bathtub, though not a direct one (see Figure 6). To do this, he takes a special puck— used for doing replacements—and puts it on the bathtub. In response, the system projects on to the table a list of potential specializations of bathtub and Manny moves the special puck to the one called Aging Chain and types in new labels: Factory, Assembly and Warehouse (see Figure 7). Manny also specifies that factory currently has around 10 million chips, whereas Assembly and Warehouse have 5 million each. He also sets manufacturing Cycle Time to 10 weeks, and Assembly Time and Selling Time to 5 weeks each.

Behind the scenes, the system automatically generates a new model at each step, simulates it, and projects the results on to the table. After the last step above, the Sensetable shows a diagram like the one in Figure 10 and the simulation engine has the following model:

$$\frac{\mathrm{d}Factory}{\mathrm{d}T} = starts - manufacturing$$

$$starts = 900{,}000$$

$$manufacturing_t = \frac{Factory_t}{manufacturingCycleTime} = \frac{Factory_t}{10}$$

$$\frac{\mathrm{d}Assembly}{\mathrm{d}T} = manufacturing_t - assembling_t$$

$$assembling_t = \frac{Assembly_t}{assemblyTime} = \frac{Assembly_t}{5}$$

$$\frac{\mathrm{d}Warehouse}{\mathrm{d}T} = assembling_t - selling_t$$

$$selling_t = \frac{Warehouse_t}{sellingTime} = \frac{Warehouse_t}{5}$$

$$Factory_o = 10{,}000{,}000;\ Assembly_o = 5{,}000{,}000;\ Warehouse_o = 5{,}000{,}000$$

Note that the system automatically sets the value of the inflow to be the same as the inflow to the original bathtub. In addition, the system automatically propagates the units through the more complicated structure. With the model equations, inflow, and initial conditions specified, the system generates the dynamic behavior shown in Figure 11.

Polly says that she does not actually keep starts constant at 900,000, but instead is continually smoothing production requests coming from upstream. Using the pucks, the smooth molecule and a few quick gestures, she alters the diagram accordingly (Figure 12).

DOI: 10.1002/sdr

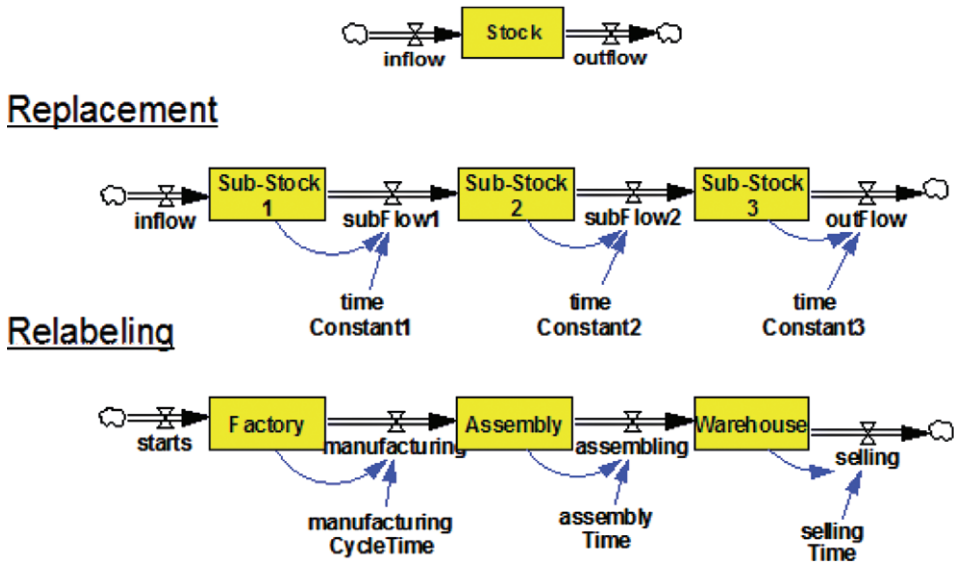

Fig. 10. First three actions in scenario

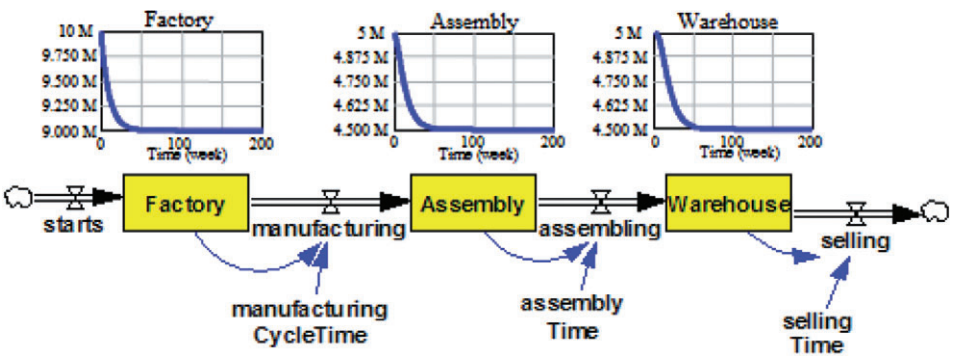

Fig. 11. Simulating model after first three replacements

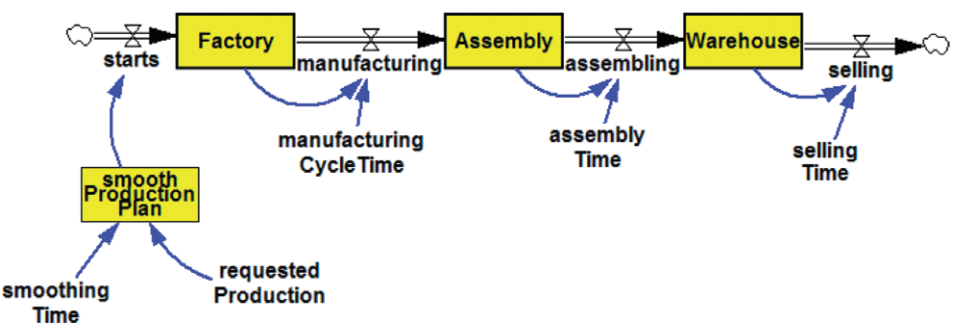

Fig. 12. The production manager smoothes production requests

DOI: 10.1002/sdr

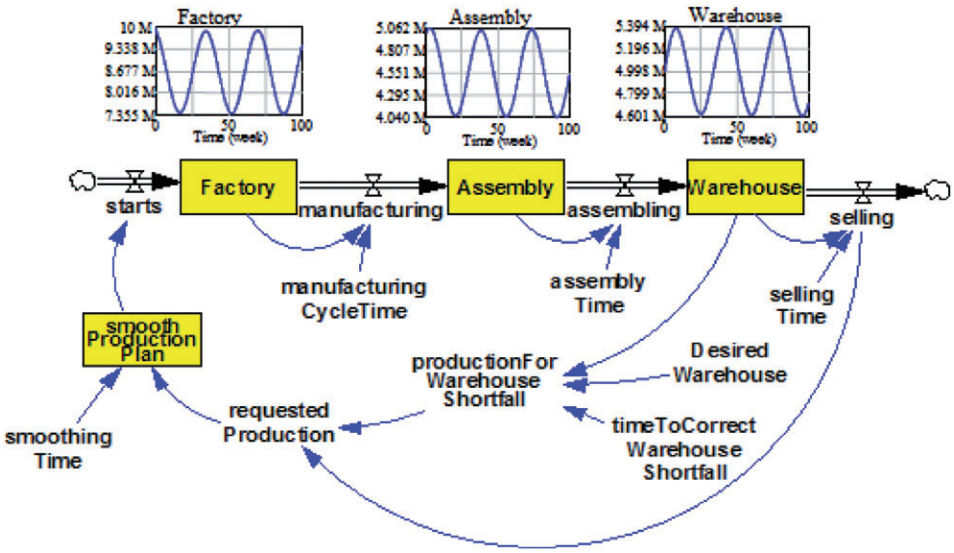

Fig. 13. Additional production is requested to correct a warehouse shortfall

The warehouse manager says that he would not allow the warehouse inventory to fall as it does in Figure 11. Instead, he would request that production be increased to eliminate the shortfall between the desired and the actual position of the warehouse. While the warehouse manager, using a goal-gap molecule (see Figure 3), makes the required substitutions, Polly observes that the warehouse request is one component of total requested production, explaining that the other component, information on shipments, represents production required to replace what is being sold. As the warehouse manager finishes his modification, Polly put in hers, leading to the model diagram in Figure 13.

Eventually the three managers arrive at a model that oscillates (Figure 13) and, in so doing, realize that their well-meaning policies for factory starts and warehouse control, although quite reasonable in themselves, combine with the factory cycle time to produce cyclical ups and downs in all inventories as well as in all flows (e.g., starts, manufacturing, assembling, and selling). Interestingly, Polly's well-meaning attempt to smooth production actually increases system-wide instability.

Based on this new shared understanding, the three managers continue using the technology to design policies that not only work well in isolation but also work well *together.* When the new policies are implemented, the supply chain operates more smoothly with less waste, less disruption and, not incidentally, less wear and tear on the managers involved. The managers know that the future will bring changes and, eventually, the need for further redesign. But, because their model was automatically stored as a molecule in its proper place within the hierarchy, any subsequent redesign will pick up where the three managers previously left off.

Discussion

Firm conclusions concerning the benefits and drawbacks of this approach must wait more extensive testing. Nonetheless, our anecdotal experience of using this approach

DOI: 10.1002/sdr

with the company we studied is highly suggestive of the benefits, limitations and open issues of this approach.

Benefits

Speed of model construction

The most obvious benefit of this approach is that models can be created more rapidly even for a professional modeler. With predefined components available (and locatable), modelers do not need to create from scratch the standard formulations they want to use. Our rough estimates are that 80–90 percent of most professionally built system dynamics models are composed of standard formulations. Currently, modelers have to create each formulation anew every time they use it.

Being relieved of this mindless repetition can itself be a time saver. In addition, typographical errors often complicate the process of recreating a standard formulation from scratch. Using guaranteed typo-free pre-built molecules eliminates the considerable time even very good modelers spend tracking down the sources of odd behavior generated by such errors. Further, because the system suggests replacements, a modeler who previously was unaware of the existence of a useful molecule will have it automatically suggested to him—saving the time that would otherwise be spent needlessly "reinventing the wheel".

Conversation-oriented modeling

Conversation normally proceeds much faster than traditional modeling. The increase in modeling speed from using our system appears to be about the same order of magnitude as that by which conversation normally outpaces modeling; and, in fact, in demonstrations with our research sponsors, the modeling seemed to easily keep pace with the conversation around the system. The approach described here promises to allow *modeling* to be used within a group conversation. We suspect that this combination may alter the nature of managerial conversations, by adding the equivalent of a flipchart that can "talk back" via the magic of simulation.

Engaging people

We have found that the TUI seems to have a remarkable effect on many people to whom we show the system. They are engaged—drawn into it. This engagement effect enhances the probability that this system can change the nature of conversations and collaboration in organizations.

Confidence

Managers who have seen our system rarely ask about the validity of the model being constructed, perhaps because they are *there* while the model is built. Since they know what is in the model, they do not wonder if the simulated behavior is due to some hidden formulation. As importantly, building models from pre-existing (and previously vetted) molecules reduces the fear that model behavior arises from idiosyncratic (or erroneous) formulations of a particular modeler.

Speed of learning

It currently takes years for a would-be system dynamics modeler to become truly proficient. One reason for this is that, until now, modelers have had to construct their own

DOI: 10.1002/sdr

mental warehouse of robust molecules. Our system externalizes these molecules and makes them available through easy navigation. It seems likely that a result may be that beginners will find themselves becoming better modelers sooner. Indeed, one of the authors taught an MBA course in which the early, primitive hierarchy of molecules was introduced to students. The midterm exam required students to create a model of a causal diagram within an hour and a half—a task that even advanced doctoral students might find difficult. In this case, however, every student completed the task successfully.

Relaxing the skill requirements

Quite apart from the possibility of shortening the time to become an expert modeler is the possibility that this approach will enable non-experts to create models without hiring a professional. With predefined molecules, an intelligent interface providing options for the modeler, and a process (modeling by replacement) that guarantees a well-formed model, it is possible that building good models will require less skill. Of course, our approach only helps with some of the skills required of a modeler. For example, a system dynamics modeler also needs to be able to conceptualize the growing model in terms of feedback loops. Nonetheless, it seems almost certain that the approach described here will lower, to some extent, the hurdle to modeling.

The cost of modeling

In a prior section we noted that creating even simple system dynamics models can cost from $25,000 to $100,000, and large calibrated models can cost an order of magnitude more. Speeding the modeling process promises to reduce these costs (though the time spent modeling may amount to only 25 percent of the total consultant time).

A more significant saving may come from the automatic storage and cataloguing of new molecules. When previous modeling efforts are, in effect, "cannibalized for parts" via the automatic generation and cataloguing of molecules, the economics of simulation modeling can change radically. As a company continues to use the approach described here, the repository of molecules grows, making it more likely that the right molecule will be available for the next modeling effort, and thus the cost of modeling continues to fall.

Testing levels of aggregation

The approach makes it easier for users to test the level of aggregation of useful SD models. Since users start from single stocks and through replacement and specialization arrive at the resulting models, they could potentially evaluate the usefulness of further disaggregation once a first aggregated model has been completed. Hence users could quickly disaggregate models to verify whether the resulting ones would behave differently and yield more insightful representations of reality.

Limitations

Many of the limitations we have glimpsed through our early experience with this approach are the flipside of its benefits. For example, the fact that molecules accumulate means that in the early stages of using our process a company will have access to only the generic molecules that come with the system. It is only by using the hierarchy that

DOI: 10.1002/sdr

the hierarchy takes on a richness and specificity to the organization using it. Since our approach is most difficult to use at the beginning, it may face a hurdle to initial adoption.

Clutter

Although accumulating molecules adds to the hierarchy's richness, it can also create clutter—a problem that we have experienced ourselves in our testing and demonstrations of the system. Currently, our system automatically adds to the hierarchy a new molecule every time a replacement is made. Some of these molecules are not useful and need to be pruned. A solution to this problem is to make cataloguing new molecules less automatic. Perhaps the user should be asked if the molecule merits storage.

Need for facilitation

A further concern involves the potential need for facilitation. Although we have noted that our approach makes modeling easier, it is not clear how much easier things will be.

While we hope that a user will not need to be a professional modeler, our use of the tool has always involved people with significant experience in all three underlying components: system dynamics, TUI and *Process Handbook*. We simply do not know what minimum level of skill is needed to make productive use of this approach. At least initially, it is very likely that use of the system will require someone familiar with simulation modeling and with the system itself.

Moleculitis

When we first began advocating the use of molecules, one leader in the system dynamics field worried that it permitted naïve modelers to string together molecule after molecule with no real justification. The resulting models would grow ever larger, while never delivering any benefit to anyone. He termed this condition "moleculitis". As mentioned earlier, we have observed a little of this in classes in which we have taught the early, primitive set of molecules. However, most students experienced an unprecedented jump in modeling capabilities.

The increased ease of building models that this tool provides has raised another similar concern among some of our colleagues. Some have suggested that uninformed modelers will be able to quickly produce poor, misleading models. In fact, similar fears were raised about spreadsheets and iconographic modeling in system dynamics. People worried that spreadsheets would let inexperienced programmers create lots of incomplete, inaccurate, inconsistent, and otherwise flawed models. They also worried that iconographic modeling made system dynamics models "too accessible to non-experts", leading to poorly conceptualized, feedback-poor, and flawed models. To some degree this has turned out to be true, but most people would agree that the overall benefits of spreadsheets and iconographic modeling have far outweighed the harm done by the flawed models that people sometimes create.

Early model buy-in

While the ability to develop models quickly allows users to build and test more models, it is possible that users may get stuck in the first model that provides a solution to their

DOI: 10.1002/sdr

problem. However, since our approach allows for rapid replacement and specialization of different structures, the possibility of being stuck with an initial model is perhaps less problematic than in the traditional SD method. Ideally, the system could be used to systematically explore and assess a variety of specific modeling choices and different levels of aggregation.

Need for model simplification
It is possible that large complicated models may result from the rapid replacement and specialization of different molecules. While "moleculitis" also leads to large, complicated models, here the resulting models may still be useful but very difficult to understand. To improve the quality of the resulting models and understanding of model builders (and decision makers), model simplification would be fruitful after the creation of models using our approach, as suggested by Saysel and Barlas (2006) for the traditional system dynamics method.

Analysis
In some modeling disciplines, the real benefit of modeling comes not from the specific numerical results of a simulation but from deeper analyses of the models. In system dynamics, for example, getting a benefit usually entails understanding which feedback loops generate which patterns of behavior. Traditionally, analysis proceeds at least as slowly as creating the model itself. If there were no technology to allow analysis to proceed as quickly as a conversation, many of the collaborative and conversational advantages of speeding the modeling would be lost. Although we have not yet combined this capability with the other components of the system, our simulation engine today incorporates a new method of model analysis that automatically suggests which feedback loops are most important to a particular pattern of model behavior (Perez-Arriaga, 1981; Forrester, 1982; Gonçalves *et al.*, 2000). Our hope is that when combined with the other components this new method will provide a sizeable speedup in model analysis as well.

Open issues

An open issue to be explored in future research deals with the applicability of our approach to different knowledge domains. We chose to do our initial exploration of "construction by replacement" in the domain of supply chain management because we wanted to understand how the approach would work in a relatively bounded and well-understood setting. In this sense, we chose an "easy" domain for our initial explorations, but we believe that there are a number of other areas that have been extensively studied in system dynamics (e.g., project management) and that would be equally easy domains in which to apply this approach. Moreover, several classic system dynamics models (such as the capital growth model) could be readily constructed by replacement from the taxonomy we presented here. Hybrid models including agents, genetic algorithms, and other methodologies may also be modeled using our approach, but further research is required to show examples of its applicability.

While we cannot guarantee that our approach will be easily transferable to different knowledge domains, counter-examples (if they exist) would be extremely helpful in understanding the limitations of the approach. We have tried ourselves to find such

DOI: 10.1002/sdr

counter-examples, but we were not successful. Based on this experience, our intuition suggests that finding counter-examples would be difficult, but it would be desirable in future work (ours or others') to be able to specify this intuition and if possible formulate it as a proof.

Conclusion

We have seen in this paper how the approach of constructing simulation models by successively replacing parts of predefined molecules with more specialized molecules has the potential to substantially improve the cost, quality, and usefulness of simulation modeling. In our work to date with applying this approach, we have developed a hierarchy of increasingly specialized simulation molecules for system dynamics models of supply chains. We believe that this same general approach can also be used with other modeling disciplines besides system dynamics, but further work is needed to demonstrate this concretely. In general, we hope that the work reported here will stimulate others to further develop this approach and apply it more broadly to many kinds of simulation models.

Notes

1. The primary organizing principle for the class hierarchy in most object-oriented programs is implementation inheritance, an efficient strategy for programmers. In contrast, our molecule hierarchy strictly enforces replacement, an efficient strategy for component users.
2. A *DrainToZero* is a stock whose outflow is equal to some *desiredOutflow* as long as that outflow will not cause the stock to fall below zero in the next solution interval. *Shipping* might be defined as the minimum of *demand* or the outflow that will empty the stock in a single simulated instant (i.e., a single "dt"): $shipping = \min\left(\frac{finishedChips}{\mathrm{d}t}, demand\right)$.

Biographies

Jim Hines is a system dynamics consultant with Ventana Systems and creator of the System Dynamics Distance program at Worcester Polytechnic Institute, where he currently teaches. Jim's research has focused on organizational evolution, "modeling at conversation speed," and automated model analysis. A past president of the System Dynamics Society, Jim has consulted all over the world and holds a Ph.D. in system dynamics from MIT, and an MBA from the University of Chicago.

Thomas W. Malone is the Patrick J. McGovern Professor of Management at the MIT Sloan School of Management and the founding director of the MIT Center for Collective Intelligence. He was also the founder and director of the MIT Center for Coordination Science and one of the two founding co-directors of the MIT Initiative on "Inventing the Organizations of the 21st Century". His most recent book is The Future of Work: How the New Order of Business Will Shape Your Organization, Your Management Style, and Your Life. Professor Malone has also published over 75 articles, research papers, and book chapters, been an inventor on 11 patents, and co-edited three books.

DOI: 10.1002/sdr

Paulo Gonçalves is Associate Professor of Management at the University of Lugano, Research Affiliate at the MIT Sloan School of Management and Academic Director of the Advanced Master in Humanitarian Logistics and Management. He obtained his Ph.D. in Management Science and System Dynamics from MIT Sloan School of Management. His work focuses on understanding behavioral aspects of common operational decisions. Current research interests include the development of supply chain experiments for understanding and improving managerial decision making. His work combines a number of techniques such as simulation, optimization, econometrics and non-linear dynamics.

George Herman is a research scientist retired from MIT whose areas of expertise are in process improvement and organizing business knowledge. He also spent many years improving logistics information systems at a major computer manufacturer.

John Quimby has been a research scientist at MIT for the last fifteen years. His current projects are developing knowledge management and simulation software in the Biology Department for biofuels and bioplastics research. While working in the Sloan School's CCS and CCI research centers his focus was the software development of the MIT Process Handbook and ELM. For fifteen years prior to MIT, John developed simulators, collaboration tools, and workflow products at Digital Equipment Corporation.

Mary Murphy-Hoye is a Senior Principal Engineer at Intel Corporation. An innovator in Information Technology and Supply Chain solutions, Ms Murphy-Hoye integrates emerging technologies to create and implement large-scale experiments in high volume production environments. Her most recent focus has been the creation of Intel's RFID/Wireless Sensor Networks Lab for industry-scale proactive computing experimentation across businesses.

Jim Rice is the Deputy Director of the MIT Center for Transportation. He also serves as the Director of the MIT Integrated Supply Chain Management (ISCM) Program and MIT Supply Chain Exchange. Jim's research is focused on supply chain design, currently focused on design for security and resilience, and recently he directed the productive Supply Chain Response project at CTL. Currently, Jim is working with National Center for Secure and Resilient Maritime Commerce and Coastal Environments (CSR) to develop secure and resilient maritime transportation systems.

James Patten Ph.D. is the founder and principal of the design and technology firm Patten Studio LLC, based in New York. Patten designs and develops interactive tabletop systems that make complex datasets easier to understand and manipulate. He earned his doctorate at the MIT Media Lab where he developed the Sensetable tabletop interface platform.

Hiroshi Ishii is a tenured Professor of Media Arts and Sciences, at the MIT Media Lab. He was named Associate Director at the Media Lab in May 2008. He co-directs Things That Think (TTT) consortium and directs Tangible Media Group. Hiroshi Ishii's research focuses upon the design of seamless interfaces between humans, digital information, and the physical environment.

References

Alexander C. 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press: New York.

Alexander C. 1979. *The Timeless Way of Building*. Oxford University Press: New York.

DOI: 10.1002/sdr

Davis, PK, Anderson, RH. 2004. Improving the Composability of DoD Models and Simulations. *The Journal of Defense Modeling and Simulation Applications, Methodology, Technology* **1**(1): 5–17.

Eberlein RL, Hines JH. 1996. Molecules for modelers. In *1996 International System Dynamics Conference*, Cambridge, MA.

Fleishman E, Hemple W. 1994. Design of object oriented simulations in Smalltalk. *Simulation* **49**: 239–252.

Forrester JW. 1958. Industrial dynamics: a major breakthrough for decision makers. *Harvard Business Review* **26**(4): 37–66.

Forrester JW. 1961. *Industrial Dynamics*. MIT Press: Cambridge, MA. (now available from Pegasus Communications, Waltham, MA).

Forrester JW. 1968. Market growth as influenced by capital investment. *Industrial Management Review* **9**(2): 83–105.

Forrester NB. 1982. *A dynamic synthesis of basic macroeconomic theory: implications for stabilization policy analysis.* Doctoral dissertation, MIT, Cambridge, MA.

Gamma E, Helm R, Johnson R, Vlissides J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley: Reading, MA.

Goldberg A, Robson D. 1989. *Smalltalk-80: The Language*. Addison-Wesley: Reading, MA.

Gonçalves P. 2003. *Demand bubbles and phantom orders in supply chains*. PhD dissertation. MIT Sloan School of Management, Cambridge, MA.

Gonçalves P, Lertpattarapong C, Hines J. 2000. Implementing formal model analysis. In *18th International Conference of the System Dynamics Society*, Bergen, Norway, July 2000.

Gonçalves P, Hines J, Sterman J. 2005. The impact of endogenous demand on push–pull production systems. *System Dynamics Review* **21**(3): 187–216.

Hines JH. 1996. *Molecules of Structure version: Building Blocks for System Dynamics Models 1.4*. Ventana Systems and LeapTec: Cambridge, MA.

Ishii H, Ullmer B. 1999. Tangible bits: towards seamless interfaces between people, bits and atoms. In *Proceedings of Conference on Human Factors in Computing Systems (CHI '97)*, Atlanta, GA, March 1997. ACM Press: New York; 234–241.

Lalond WR, Pugh JR. 1991. Subclassing ≠ subtyping ≠ Is-a. *Journal of Object- Oriented Programming* **3**(5): 57–62.

Lee J, Wyner G. 2003. Defining specialization for data flow diagrams. *Information Systems* **28**(6): 651–671.

Liskov B, Wing J. 1994. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems* **16**(6); 1811–1841.

Lyneis JM. 1980. *Corporate Planning and Policy Design*. MIT Press: Cambridge, MA. (now available from Pegasus Communications, Waltham, MA).

Malone TW, Crowston KG, Lee J, Pentland B, Dellarocas C, Wyner G, Quimby J, Osborn CS, Bernstein A, Herman G, Klein M, O'Donnell E. 1999. Tools for inventing organizations: toward a handbook of organizational processes. *Management Science* **45**(3): 425–443.

Malone TW, Crowston KG, Herman G (eds). 2003. *Organizing Business Knowledge: The MIT Process Handbook*. MIT Press: Cambridge, MA.

Meyer B. 1992. *Eiffel: The Language*. Prentice-Hall: Upper Saddle River, NJ.

Panko R, Halverson R. 1996. Spreadsheets on trial: a survey of research on spreadsheet risks. In *Proceedings of the 29th Annual Hawaii International Conference on System Sciences*; 326–335.

Patten J, Ishii H, Pangaro G, Hines J. 2001. Sensetable: a wireless object tracking platform for tangible user interfaces. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI '01)*, Seattle, WA, 31 March–5 April 2001. ACM Press: New York; 253–260.

Perez-Arriaga IJ. 1981. *Selective modal analysis with applications to electric power systems*. Doctoral dissertation, MIT, Cambridge, MA.

DOI: 10.1002/sdr

Richardson G, Pugh J. 1981. *Introduction to System Dynamics with Dynamo.* MIT Press: Cambridge, MA. (now available from Pegasus Communications, Waltham, MA).
Richmond B. 1985. STELLA: software for bringing system dynamics to the other 98%. In *Proceedings of the 1985 International Conference of the System Dynamics Society*, Keystone, CO; 706–718.
Saysel K, Barlas Y. 2006. Model simplification and validation with indirect structure validity tests. *System Dynamics Review* **22**(3): 241–262.
Sterman J. 2000. *Business Dynamics*. McGraw-Hill: New York.
Vennix JAM, Andersen DF, Richardson GP. 1997. Foreword: group model building—art and science. *System Dynamics Review* **13**(2): 103–106.

DOI: 10.1002/sdr