CONSTRUCTION BY REPLACEMENT: A NEW APPROACH TO SIMULATION MODELING

Authors:

- James H Hines (MIT Sloan School of Management)
- Thomas W. Malone (MIT Sloan School of Management)
- George Herman (MIT Sloan School of Management)
- John Quimby(MIT Sloan School of Management)
- Mary Murphy Hoye
- James Rice
- Paulo Goncalves
- James Patten
- Hiroshi ishii

Sloan School of Management

Massachusetts Institute of Technology

MIT Sloan School of Management Working Paper No.

MIT Center for Coordination Science Working Paper No. 225

Abstract:

Simulation modeling can be valuable in many areas of management science, but is often costly, time-consuming and difficult to do. This paper describes a new approach to simulation that promises to be much cheaper, faster and easier to use in many situations. In this approach, users start with a very simple generic model and then progressively replace parts of the model with more specialized "molecules" from a systematically organized library of predefined components. At each point, the system lets the user select from lists of possible substitutions, and then either automatically creates a new running model or shows the user where further manual changes are needed.

The paper describes an extensive experiment with using this approach to construct system dynamics models of supply chain processes in a large manufacturing company. The experiment included developing a comprehensive catalog of system dynamics molecules analogous to the periodic table in chemistry. The experiment also included developing an innovative "tangible user interface" with which users can create simulation models by moving actual physical objects around on a special table called a Sensetable. The paper concludes with a discussion of the benefits and limitations of this approach and how it could be used in other situations.

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.

This paper is about a new approach to simulation modeling that has the potential to substantially reduce the time, cost, and effort of creating simulation models. The approach starts with a systematically organized library of predefined model components (or “molecules”) and a set of software tools for replacing one molecule with another molecule. Using this tool kit, users can rapidly construct new simulation models by replacing parts (molecules) of models with more specialized versions of the same parts.

While related ideas have been used in some previous simulation systems for years (e.g., Goldberg & Robson 1989, Myer 1992, Fleischman and Hemple 1994), we believe that the approach we describe in this paper takes the basic ideas significantly further than any previous efforts of which we are aware. In particular, our approach 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 1997) where users manipulate actual physical objects on a special table.

We don’t believe that this (or any other current) approach is a “magic bullet” that makes the creation of simulation models instantaneous and effortless, but we do believe that our approach 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 and validate our approach, we conducted a substantial multi-year investigation using the approach 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 kind of "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 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 (Gonçalves 2003, Gonçalves et al. 2004). 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 in the 20th century 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.

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 arranged in a *specialization hierarchy* with increasingly specialized versions of a component automatically "inheriting" properties of their more generic "parents".

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 who don't have special training as programmers or modelers can create for themselves only the simplest and least flexible form of simulations -- those using spreadsheets. 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).

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 $25,000 to $100,000; large efforts can range into the millions (Dalton 2003, Eberlein 2003). As for error rates, Panko concludes in his survey of studies of spreadsheet models that "every study that has looked for errors has found them in significant numbers" (Panko & Halvorson, 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 – even very costly models – 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 born by the potential benefit of solving only the *current* problem.

One of the important goals of this project is to develop a tool and an approach to creating simulation models that allow expert and non-experts users alike to create simulation models faster, more accurately, and with more reusability than ever before.

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 cataloging 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 planning production, 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 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 non-linear 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) is represented as a double arrow, a stylized pipe. A policy (decision rule) controlling a flow is represented as a stylized "valve" (often depicted as a simple hour-glass 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 – rather than a rectangle – is used. Figure 1 illustrates the symbols.

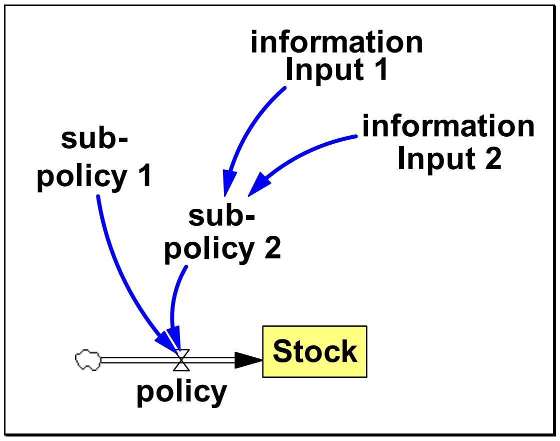

Figure 1: Symbols Used in System Dynamics Stock-and-Flow Diagrams

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 that is a delayed version of another flow.

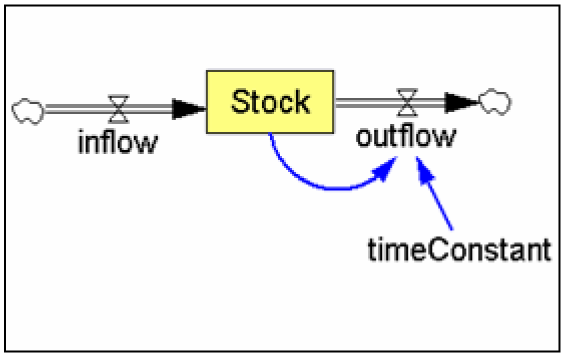

Figure 2: System Dynamics *Material Delay* Molecule

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

$$\frac{d}{dt}Stock_t = inflow_t - outflow_t$$
$$outflow_t = \frac{Stock_t}{timeConstant}$$

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).*

The idea of molecules is not limited to system dynamics simulations. For example, an important molecule for discrete event modeling might be the common structure that represents a resource (e.g. a forklift) moving an entity (e.g. an engine) from one queue (e.g. parts inventory) to another (assembly line).

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 UML class diagram of a small subset of the specialization hierarchy around the material delay molecule described above[1].

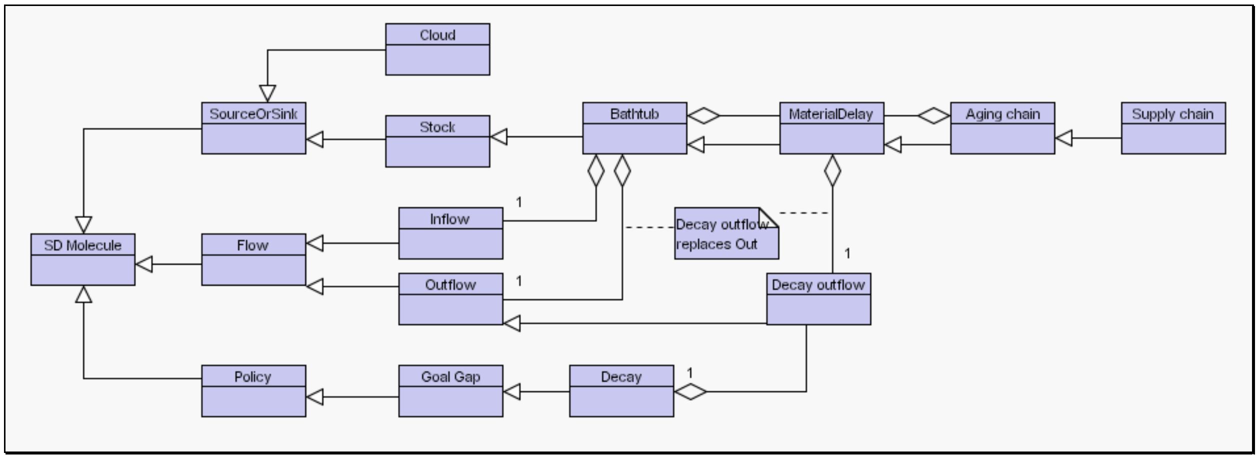

Figure 3: Specialization Hierarchy Surrounding the Material Delay

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's useful to focus on a single chain. For example:

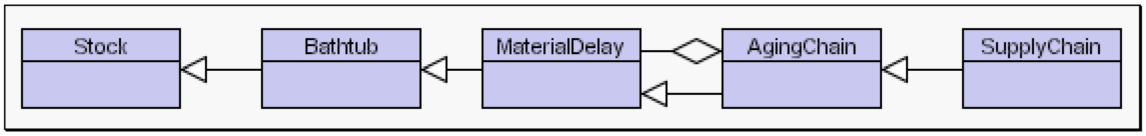

Figure 4: A Small "Chain" of the Specialization Hierarchy

A Stock can have any number of inflows and outflows. Figure 4 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.

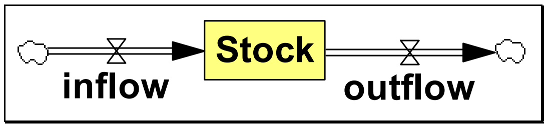

Figure 5: The Bathtub Molecule

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, above) and defined mathematically as the expression

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

An aging chain is a disaggregation of a (first-order) material delay into an nth-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 below. Each of the disaggregated time constants would normally be set to one-third the original timeConstant.

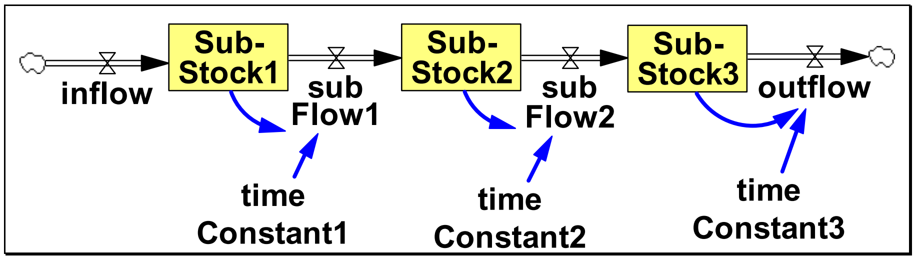

Figure 6: The Aging Chain Molecule

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.

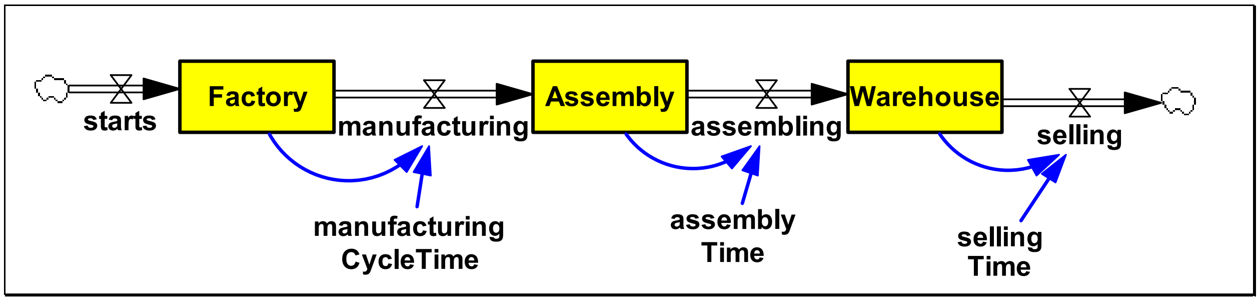

Figure 7: A Simple Sequential Supply Chain

This specialization created by a modeler, is a running model and has its own place in the specialization hierarchy (see bottom of 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—even more 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.

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"[2]. 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 & 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[3]. For example, in a system dynamics model, you can always replace a *bathtub* with an *aging chain* and 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 valid molecule whose parts are the molecules M_i (for i = 1…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 valid molecule.

When this property holds users can refine subparts of a model and have the new subparts automatically substituted into the overall model.

Once the molecules have been arranged in a systematic specialization hierarchy as just described, this automatic refinement is 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. And any component that used the outflow from the original material delay would receive instead the outflow from the new aging chain, while components that depended on information about the single stock of receivables in the old formulation, would automatically now 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 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.

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 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, as a byproduct, 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 valid molecule whose parts are the molecules M_i (for $i = 1…n$), 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 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 cataloging and storage function is easily automated.

Say we change the inventory model of Figure 7 by substituting a specialization of the outflow. We immediately create a specialization of the original simple inventory model. The specialization is located right "beneath" the original inventory model. If we have a larger industry model of which the original inventory model was a component, we can now replace the original inventory 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 valid model can be constructed by making successive replacements in the way just described. Even when such a complete taxonomy cannot be constructed, the approach we've described can still speed and simplify some (often, most) of the model creation task.

IMPLEMENTING THE APPROACH

To develop and test this approach, we applied it to the kind of simulation common in the field of system dynamics and to the application domain of supply chains. We also used a new generation of *tangible* user interface (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 and wide ranging contacts within 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.

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 a number – sometimes a large number -- 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 just two primitive components – stocks and flows – so it seemed likely that the process of replacement would usually work in a system dynamics model. Second, the system dynamics approach to simulation is as mature as any simulation approach and more mature than most, and consequently a number of common structures 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, that of 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 9). 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.

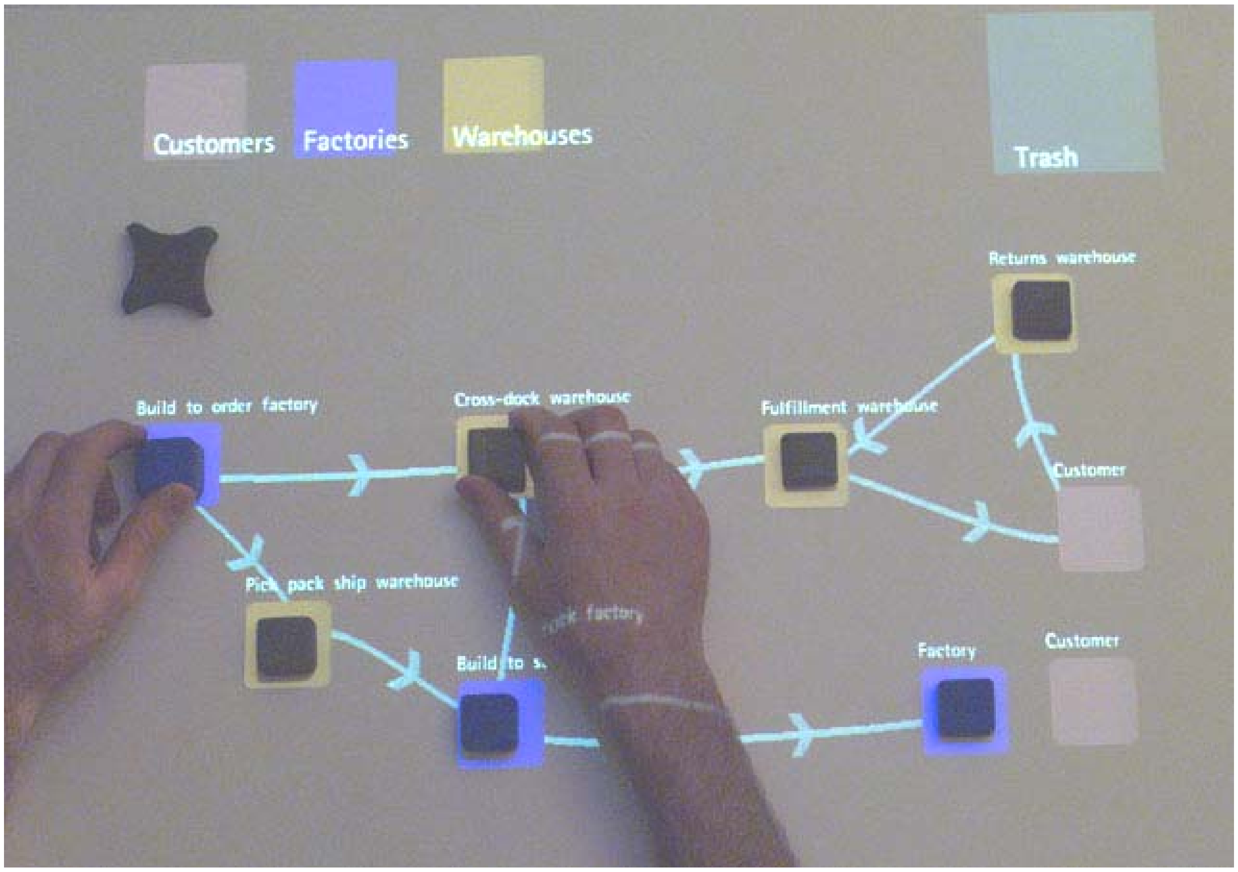

Figure 8: The tangible user interface

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 developed by Hines (Eberlein, Hines (1996), Hines (1996)). This earlier hierarchy, however, did not strictly enforce the substitution properties described above, so 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 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 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: $Outflow_t = f(Stock_t) * IndicatedOutflow_t$. The function *f( )* 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*$\rightarrow$ *outflow* $\rightarrow$ *Outflow Below Maximum*$\rightarrow$ *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 fit, and it was moved from its prior parent, *Decay*. We realized another formulation – *DrainToZero*[4] 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 cover the entire concept represented by their common parent. For example, having the molecule *OutflowBelowMaximum* raises the question whether there should be a molecule for *OutflowAboveMinimum*. 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.

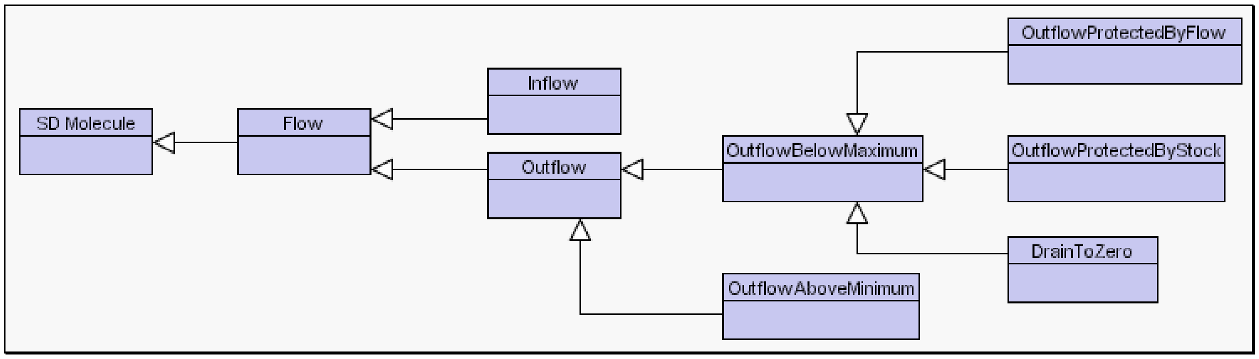

Figure 9: *Outflow Protected By Stock* and a portion of the new taxonomy

As we applied these processes of filling-in-the-chain and looking for CES, the original set of 50 molecules, grew to over 200. By systematically organizing molecule types into a replacement hierarchy we created knowledge about the range of possible elements in a simulation model. In this sense, therefore, 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 on-line knowledge base called the "Process Handbook" (see Malone et al, 1999; Malone, Crowston, Herman, 2003) which already included extensive facilities for manipulating and viewing textual and graphical descriptions of processes arranged in specialization hierarchies and a pre-existing 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 don't 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've just described for tangible user interfaces (TUI's) (Patten et al 2001) as well as graphical user interfaces (GUI's). In TUI's, for instance, instead of showing menus of alternative replacements on 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 – three people whose collaboration is central to performance, but who seldom find a time or 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 Tangible User Interface (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" into 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 to 15 million --her guess about the total product 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.

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 and Figure 4). To do this, he takes a special puck—used for doing replacements—and puts it on the Bathtub. In response, the system projects on 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, Manufacturing, Assembly and Warehouse (see also Figure 7).

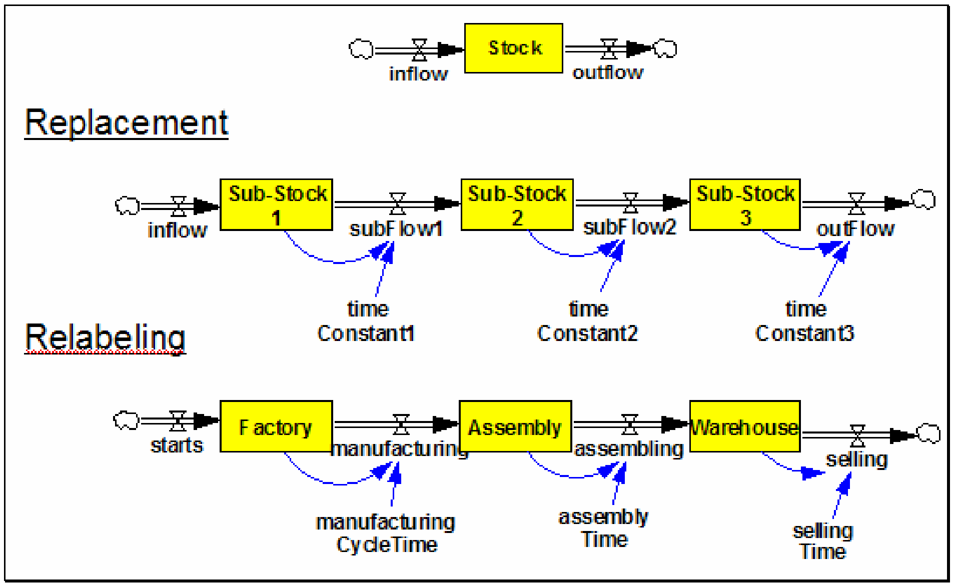

Figure 10: First three actions in scenario

Behind the scenes, the system automatically generates a new model at each step, simulates it, and projects the results on the table. After the last step above, for instance, the simulation engine has the following model, and the table shows a diagram something like the one in Figure 11:

$$\frac{dFactory}{dt} = starts - manufacturing_t$$
$$starts = 900000$$
$$manufacturing_t = \frac{Factory_t}{manufacturingCycleTime}$$
$$\frac{dAssembly}{dt} = manufacturing_t - assembling_t$$
$$assembling_t = \frac{Assembly_t}{assemblyTime}$$
$$\frac{dWarehouse}{dt} = assembling_t - selling_t$$
$$selling_t = \frac{Warehouse_t}{sellingTime}$$

Note that the System automatically calculates the initial values of the new Levels. It sets the value of the inflow to be the same as the inflow to the original Bathtub. Finally, the system also automatically propagates the units through the more complicated structure.

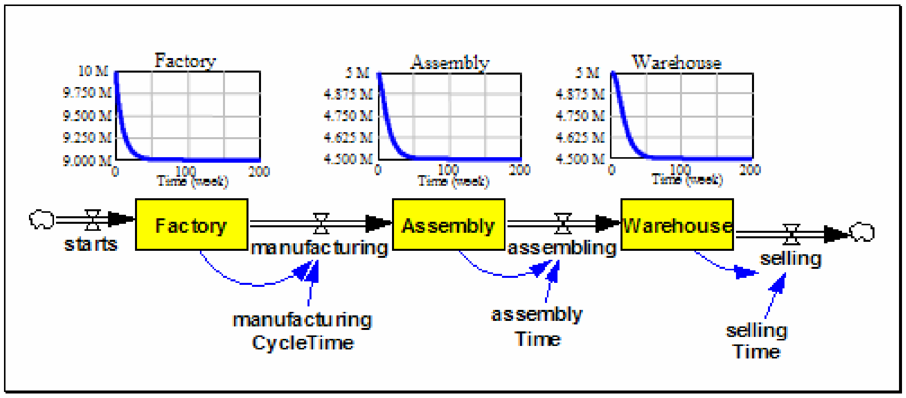

Figure 11: Simulating model after first three replacements

Polly says that she doesn't 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.

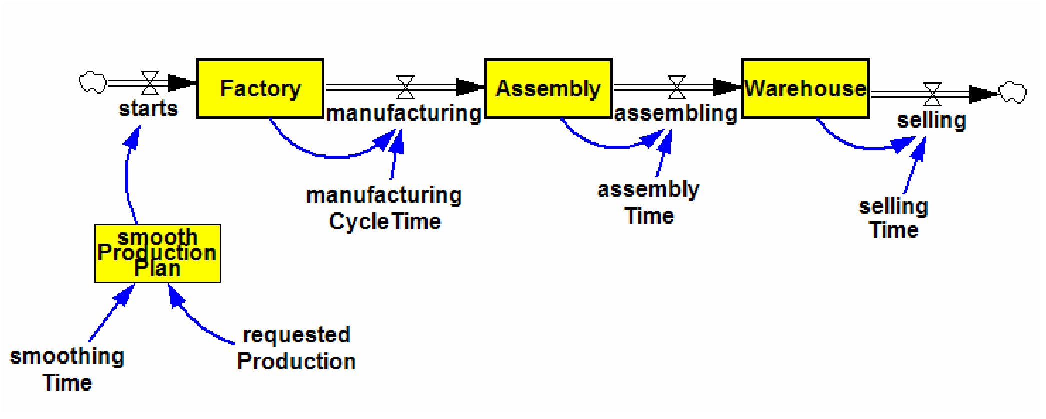

Figure 12: The production manager smooths production requests

The warehouse manager says that he wouldn't 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 managers, 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.

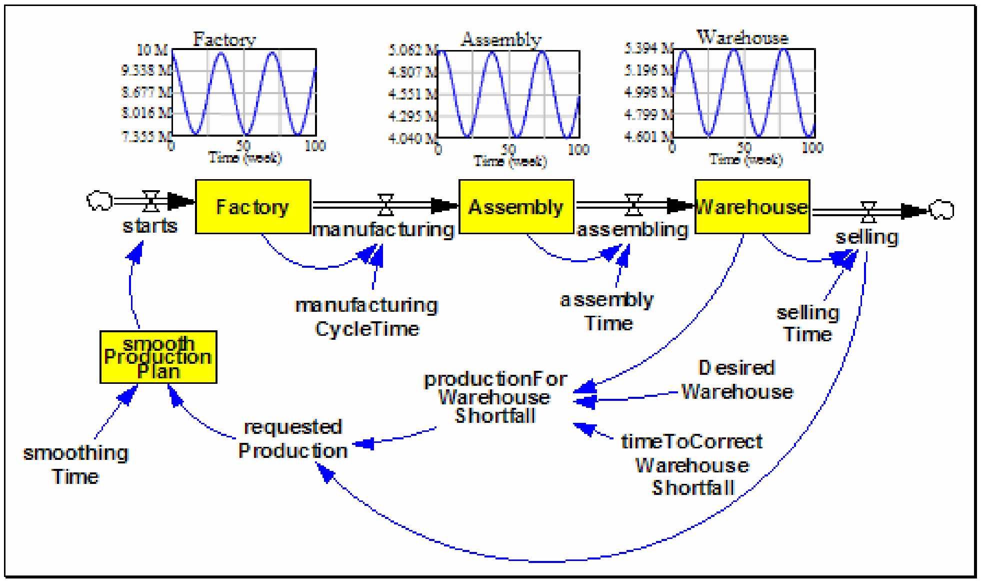

Figure 13: Additional production is requested to correct a warehouse shortfall.

Eventually the three managers arrive at a model that oscillates 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 – starts, , manufacturing, assembling, and sales. 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 await more extensive testing. Nonetheless, our anecdotal experience of using this approach with the company we studied is highly suggestive of both the benefits and limitations of this approach.

Benefits

Speed of model construction. The most obvious benefit of this approach is that models can be created more rapidly—perhaps by a factor of 10 or 100--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% 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 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 flip chart 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 don't 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 up until now, modelers have had to construct their own 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 us (Hines) 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's 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% of the total consultant time).

A more significant savings may come from the automatic storage and cataloging of new molecules. When previous modeling efforts are in effect, "cannibalized for parts" via the automatic generation and cataloging 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.

Limitations.

Many of the limitations we have glimpsed through our early experience with this approach are the flip side 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 the hierarchy takes on a richness and specificity to the organization using it. Such an approach, that is most difficult to use at the beginning, may face a hurdle to acceptance.

Clutter. Although accumulating molecules adds to the hierarchy's richness it can also create clutter, a problem that we've 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've noted that our approach makes modeling easier, it is not clear how much easier things will be. While we hope a user won't 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 don't know what minimum level of skill is needed to make productive use of this approach. It is very likely that -- at least initially -- 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". We have in fact observed a little of this in classes in which we have taught the early, primitive set of molecules. However, the incidence of moleculitis was small, and as noted above, 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. People worried that spreadsheets would let inexperienced programmers create lots of incomplete, inaccurate, inconsistent, and otherwise flawed models. To some degree this has turned out to be true, but most people would agree that the overall benefits of spreadsheets have far outweighed the harm done by the flawed models that people sometimes create.

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 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 (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.

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 be used with other modeling disciplines besides system dynamics. The key to using the approach with other simulation disciplines is finding molecules and arranging them into a replacement hierarchy.

For example, we are quite optimistic about the prospects of applying this approach to Discrete Event modeling, probably the most widely used simulation technique outside of spreadsheets. Like system dynamics, discrete event simulation has been around long enough for commonly used candidate "molecules" to have evolved (e.g. Goldberg and Robson, 1989). Of course, establishing the true degree of applicability of our approach to discrete event modeling or any other simulation technique will, of course, have to await the attempt.

References

Eberlein, R. L., & Hines, J. H. (1996). Molecules for Modelers. Paper presented at the 1996 International System Dynamics Conference, Cambridge, Massachusetts.

Eberlein, Robert. 2003. Ventana Systems. Personal Communication, May 16, 2003.

Arriaga, Jose Ignacio Perez. 1981. *Selective Modal Analysis with Applications to Electric Power Systems.* Unpublished doctoral dissertation. Cambridge, MA: MIT.

Dalton, W.J. 2003. PA Consulting. Personal Communication.

Forrester, Jay W. 1961. *Industrial Dynamics.* Waltham, MA: Pegasus Communications.

Forrester, Nathan B. 1982. *A Dynamic Synthesis of Basic Macroeconomic Theory: Implications for Stabilization Policy Analysis.* Unpublished doctoral dissertation. Cambridge, MA: MIT.

Fleishman, E. and W. Hemple (1994). "Design of Object Oriented Simulations in Smalltalk." *Simulation* **49**: 239-252.

Gamma, Erick, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley Publishing Company.

Goldberg, Adele and David Robson. 1989. Smalltalk-80: The Language. Reading, MA: Addison-Wesley Publishing Company.

Gonçalves, Paulo, C. Lertpattarapong, and J. Hines. (2000). "Implementing Formal Model Analysis," *The 18th International Conference of the System Dynamics Society*, Bergen, Norway, July.

Gonçalves, P. (2003). "Demand Bubbles and Phantom Orders in Supply Chains." Unpublished Ph.D. Dissertation. MIT Sloan School of Management, Cambridge, MA.

Gonçalves, P., J. Hines, J. Sterman, and C. Lertpattarapong. (2004). "The Impact of Endogenous Demand on Push-Pull Production Systems." MIT Sloan School of Management Working Paper.

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

Ishii, H. and Ullmer, B., Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms, in Proceedings of Conference on Human Factors in Computing Systems (CHI '97), (Atlanta, March 1997), ACM Press, pp. 234-241.

Lalond, Wilf R. and John R. Pugh. 1991. "Subclassing ≠ subtyping ≠ Is-a". Journal of Object-Oriented Programming. 3(5):57-62.

Lee, Jintae, & Wyner, G. Defining Specialization for Data Flow Diagrams, Information Systems 28 (6) (2003): 651-671

Liskov, Barbara and Jeanette Wing, "A Behavioral Notion of Subtyping", ACM Transactions on Programming Languages and Systems, Vol 16, No 6, November, 1994, pages 1811-1841.

Malone, T. W., Crowston, K. G., & Herman, G. (Eds.) Organizing Business Knowledge: The MIT Process Handbook. Cambridge, MA: MIT Press, 2003.

Malone, T. W., Crowston, K. G., Lee, J., Pentland, B., Dellarocas, C., Wyner, G., Quimby, J., Osborn, C. S., Bernstein, A., Herman, G., Klein, M., & O'Donnell, E. 1999. "Tools for inventing organizations: Toward a handbook of organizational processes." *Management Science*, 45, 3 (March), 425-443.

Meyer, B. 1992. *Eiffel: The Language*. Prentice Hall, Upper Saddle River, N.J.

Panko, Raymond and Richard Halverson, "Spreadsheets on Trial: A Survey of Research on Spreadsheet Risks. 1996. *Proceedings of the 29th Annual Hawaii International Conference on System Sciences*. 326-335

Patten, J, H. Ishii, G. Pangaro, and J. Hines. 2001. "Sensetable: A Wireless Object Tracking Platform for Tangible User Interfaces". *Proceedings of Conference on Human Factors in Computing Systems (CHI '01)*. Seattle, Washington, USA, March 31 - April 5, 2001. ACM Press. pp.253-260.

Sterman, John. 2000. *Business Dynamics*. New York: McGraw-Hill

Vennix, J.A.M, D.F, Andersen, and G.P. Richardson. 1997. Introduction: Group Model Building -- Art and Science. *System Dynamics Review* 13,2 (summer 1997).

FOOTNOTES

---

[1] 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).

[2] Among the most noteworthy exceptions to this general rule is Gamma et al 1995 (e.g. pp. 315-325).

[3] 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.

[4] 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”): $shipping = \min\left(\frac{finishedChips}{dt}, demand\right)$