Proceedings of the 2001 Winter Simulation Conference
B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds.

SISCO: A SUPPLY CHAIN SIMULATION TOOL UTILIZING SILK™ AND XML

Dean C. Chatfield

Department of Management Science
and Information Technology
1007 Pamplin Hall (0235)
Virginia Tech
Blacksburg, VA 24061, U.S.A.

Terry P. Harrison
Jack C. Hayya

Department of Management Science
and Information Systems
303 Beam Building
Penn State University
University Park, PA 16802, U.S.A.

ABSTRACT

We discuss SISCO, the Simulator for Integrated Supply Chain Operations, a Java-based tool that simplifies supply chain simulation model development. SISCO maps supply chain descriptions stored in the XML-based Supply Chain Modeling Language (SCML) format to a set of supply chain "building blocks" developed with ThreadTec's Silk™ simulation classes. The resulting system combines the ease of a visual supply chain simulator, the power and flexibility of a full object-oriented programming language, and the unparalleled supply chain model detail of a new, open, information standard.

1 INTRODUCTION

Simulation modeling is a powerful methodology that allows supply chain modelers to capture the dynamic nature of supply chains. Two issues that hinder the more widespread use of simulation modeling for supply chain analysis are an inability to share supply chain problem information and the relative difficulty in developing supply chain simulation models.

One of the greatest roadblocks to effective supply chain analysis is the inability of supply chain researchers and practitioners to exchange information. This inability to effectively share supply chain information is due in part to the absence of a universal method for supply chain description. Individual researchers and practitioners have independently decided how the supply chains they are analyzing will be represented. Structural and managerial information is encoded differently for each modeling effort, erecting a de facto barrier to effective information sharing.

We develop the Supply Chain Modeling Language (SCML) as a standard method for describing the structure of a supply chain. SCML is an XML-based, platform-independent, methodology-independent, industry-independent means of storing the structural and managerial information that describes a supply chain's layout and operational characteristics. SCML can provide a universal language with which users of various analytical tools, including simulation tools, can share or exchange supply chain descriptions.

To address the inherent difficulty of creating simulation models of supply chains, we develop the Simulator for Integrated Supply Chain Operations (SISCO). SISCO is SCML-compliant to allow for easier interchange of problem information. SISCO allows users to graphically describe the supply chain structure they wish to model, greatly simplifying the process for the modeler, and save it as an SCML file. The SCML definition file is used by a model generation module that reads the contents of the file and creates a simulation model by performing an automated mapping of the SCML file to a library of reusable supply chain constructs, developed with Java and ThreadTec's Java-based Silk™ simulation classes.

SISCO takes the idea of a supply chain "simulator," as seen in recent developments, such as Simulation Dynamics' Supply Chain Builder and IBM's Supply Chain Analyzer, and moves it one step further. The open storage format, SCML, allows users to share supply chain descriptions not only with other SISCO users, but users of any SCML-compliant modeling tool. SISCO provides a true multi-threaded, object-oriented simulation system with full access to the underlying general programming language. The depth of model detail possible is state-of-the-art, including aspects such as explicit definition of arcs and global trade issues. Internet "friendliness" is ensured by the Java-based execution system.

SISCO's simplification of the process of developing supply chain simulation models will boost their attractiveness to researchers and practitioners alike and result in a greater amount of supply chain research and analysis that takes the stochastic nature of the processes into account.

2 OVERVIEW OF SISCO SYSTEM

2.1 Basic Structure of SISCO

SISCO is designed to provide a powerful supply chain simulation tool that simplifies the modeling process and does this as part of an open extensible framework. As such, SISCO allows a user to graphically "build" a supply chain through a graphic user interface and then specify relevant logical and descriptive information in order to fully define the supply chain. This supply chain description is stored in the open, XML-based SCML format.

SISCO allows the user to create a simulation model by mapping the SCML file contents to the SISCO Library. The SISCO Library is a set of Silk™-derived classes that represent the objects and actions occurring in a supply chain. An object-oriented approach was taken because it provides a natural mapping from reality to simulation model. The user will then be able to simulate the operation of the supply chain under various conditions.

The basic structure of the Simulator involves the following three modules, each of which will perform an important role in the final system:

- The Visual Supply Chain Editor
- The Experiment Designer
- The Model Builder.

The Visual Supply Chain Editor is used for defining a supply chain's structural and managerial aspects. The Experiment Designer is used for defining simulation specific information. The SISCO Model Builder performs the automated model generation and initiates the model execution. A conscious decision was made to make the VSCE, the SISCO Model Builder, and the Experiment Designer individual stand-alone applications, as opposed to creating a single, monolithic SISCO application. As stand-alone applications, the pieces of SISCO are not only useful as part of the SISCO system, but can also be useful in conjunction with other supply chain modeling software as well.

2.1.1 Visual Supply Chain Editor

The Visual Supply Chain Editor (VSCE) is a graphic tool for defining the supply chain to be simulated. The basic supply chain layout is built in a drag and drop fashion, resulting in a set of nodes and arcs (locations and pathways) that define the general topology of the supply chain. These nodes and arcs are then further defined, as are the resources that utilize these nodes and arcs. The Visual Supply Chain Editor is essentially a graphic SCML editor, saving the supply chain information as an SCML file. Once the user has stored the supply chain information in an SCML format file, it can be used by any SCML-compliant application.

2.1.2 Experiment Designer

The Experiment Designer gathers simulation-specific information from the user. This includes information regarding environment variables such as the number of replications, animation settings, output statistics, and similar data. The collected information is saved to an experiment file.

2.1.3 SISCO Model Builder

The Model Builder performs the most crucial function; it takes the SCML file created by the VSCE and the experiment file created by the Experiment Designer and creates a corresponding simulation model. Silk™ is the Java-based simulation engine created by ThreadTec, Inc. that is employed by SISCO for creating and executing the simulation model. The model builder maps each SCML element to a Java object that Silk™ can process, essentially performing an automated simulation model generation.

2.2 Development Technologies and Platforms

The implementation of SISCO involves merging three relatively new technologies: Java, XML processing, and ThreadTec's Silk™. Some of the technologies, mainly the XML-related ones, were not fully specified at the time of development. The SISCO Model Builder was built with the Java programming language. All Java programming was done using Symantec's Visual Cafe Professional Edition 3.0c integrated development environment utilizing release 1.1.7a of Java and release 1.0.3 of "Swing" (Java Foundation Classes 1.1), which are used for user-interface programming. Java's object orientation, platform independence, and Internet "friendliness" were major factors in its choice as the development language. Of additional import (because XML parsing is an important task of the SISCO Model Builder) is that Java currently has the best XML processing capabilities and support. Thus, the Java language is a natural fit for the development of the SISCO Model Builder.

In addition to Java, another major technology employed was XML. XML is the basis of the SCML For purposes of XML file utilization within Java, the Sun Project-X, Technology Release 2 parser was utilized. Sun's parser provides the SAX parsing capabilities necessary to allow the Model Builder to read SCML formatted files.

The third technology, the Silk™ simulation engine from ThreadTec Incorporated, is used to provide core simulation capabilities. Silk™ is a set of Java classes created by ThreadTec to provide the discrete-event simulation features needed for simulation modeling, including basic implementations of entities, queues, random number generators, and a coordinated simulation clock. Developers create Silk™-based simulation models by working directly in Java, utilizing the relatively small set of pre-built Java

classes that provide process-oriented simulation capabilities, to develop powerful Java-based simulation models (Healy and Kilgore 1998). Silk™ is easily integrated into Symantec's Visual Café 3.0a, a major commercial Java development tool. Visual Café was utilized for all development work on the SISCO Model Builder.

The "user-interaction" modules of SISCO (VSCE and Experiment Designer) were developed with Microsoft's Visual Basic 6 Professional Edition (VB6) development environment. This integrated development environment (IDE) was chosen because it is a mature graphic application development system that allows robust, visually appealing applications to be developed in a straightforward manner.

3 SUPPLY CHAIN DEFINITION – VSCE

3.1 Visual Supply Chain Editor Information Requirements

Input required for the successful design and execution of a supply chain simulation model is extensive and varied. In order to adequately describe a supply chain, both the physical (design) aspects of the supply chain and the logical (informational) aspects of the supply chain must be defined.

The physical aspects include nodes, or locations, within the supply chain, and arcs, or pathways, that connect the nodes, and components. The node types that the VSCE allows a user to create include suppliers, production facilities, warehouses, distributors, retailers, and customers. Arcs utilizing various modes may be defined, including land, rail, air, and telecommunications linkages between nodes. In addition to nodes and arcs, a supply chain's components, which include materials, finished goods, labor, currency, and other items that are consumed, transported, created, or otherwise utilized by the supply chain, must be defined.

The logical aspects of the supply chain include actions, which describe a transformation process, such as producing an item, placing an order, or shipping a delivery. Also included in a description of supply chain logic are policies that define conditions under which actions occur by describing circumstances that "trigger" the actions or by defining goals that are to be met because of performing the actions. Examples include inventory, transportation, and production.

Beyond the basic topology defined by the nodes and connecting arcs that are included in the supply chain, it is necessary to define the characteristics of each node and arc within the supply chain. The node characteristics that must be defined include the inputs and outputs, the order and shipment routings, the independent demands (for customer nodes), the storage capacities and costs, the overhead costs, the actions that can occur at a node, and the policies, such as those of inventory, that control various actions. The arc characteristics that must be defined include the distance, the travel time, the travel costs, the basic "container" employed (such as a trailer or cargo container), the maintenance costs, and the expansion costs.

3.2 VSCE - Generating SCML Files

The VSCE must be able to create the SCML equivalent of what the user has described. SCML is an XML-based markup language for defining supply chain structure and policies. Thus, XML processing capabilities are necessary. XML processing is handled by an XML parser that implements the SAX (Simple API for XML) API, the DOM (Document Object Model) API, or both. SAX parsers are faster, less memory-intensive, and provide a simple, efficient manner for processing an XML file in a sequential, one-pass, fashion. In contrast, the DOM is a tree-based API, which maps an XML document into a set of objects in a tree-like structure determined from the document's DTD (Mosenhi 1999). The DOM API allows the programmer to read objects (XML nodes on the tree), as well as XML-node modification and addition of new XML nodes to the tree. The tree-structure (the DOM) can then be navigated and information searched for, extracted, or changed.

Using a DOM parser, we can build an XML document from the ground up by specifying a single root node (the element that serves as the base of the tree) as the starting point and then adding branches to the tree. This technique meshes well with what the VSCE must do when writing an SCML, so a DOM parsing-based routine was developed for the "Save" function of the VSCE. The Editor employs the Microsoft XML parser (MSXML) to provide a DOM parser for writing the SCML formatted description of a supply chain when users choose to "Save" their work.

3.3 VSCE - Reading SCML Files

In order to allow a user to read an SCML file back into the VSCE for updating, a method for reading the file and preparing it for editing is needed. The task of reading an SCML file back into the VSCE is a natural fit for a SAX parser. The SAX parser reads through the XML files sequentially and fires events, which perform the desired information processing, as the structures of interest, such as the start of an element, are found. Unfortunately, Microsoft's MSXML does not provide SAX parsing capabilities. Vivid Creation's ActiveSAX ActiveX object is used to add SAX parsing capabilities to the Visual Supply Chain Editor.

4 EXPERIMENT DEFINITION - ED

4.1 The Experiment Designer (ED)

Simulation-specific information, generally referred to as experimental control information, is also needed to create an executable simulation. This information is gathered by running the Experiment Designer (ED); it includes infor-

mation regarding the simulation environment, animation support, empirical distributions, and output data and statistics desired.

The ED is a Visual Basic application similar in structure to the Visual Supply Chain Editor, though much simpler. The Experiment Designer presents users with a single form in which they specify the stopping criteria for each run, the number of replications, and associated information. Using MSXML's DOM parser, the information is stored in an XML-based format called the Silk Experiment Markup Language (SExML). Silk refers to the simulation engine that is employed by SISCO, ThreadTec Incorporated's Silk™. The experiment files output by the ED are used by the SISCO Model Builder, in conjunction with the SCML files generated by the VSCE, for creating and executing the simulation as the user desires.

5 MODEL GENERATION – MODEL BUILDER

The heart of the SISCO system is the SISCO Model Builder. This module translates the supply chain information provided by the user into an equivalent Silk™-based supply chain simulation model. To accomplish this, a mapping scheme and a set of specially-designed, supply chain-oriented Silk™-compatible Java classes are used to create a Java representation of the supply chain.

5.1 Modeling Approach

Our modeling approach creates autonomous units, or entities, representing each node or arc within the supply chain. These entities interact with each other, performing the basic actions an order undergoes during its life, in processing orders from inception to disposal.

5.1.1 Life Cycle of an Order

We approach the modeling of a supply chain from the perspective of an order's life cycle, from inception through delivery. The basic life-cycle of an order, including where the actions occur, is as follows:

- order creation (origin node)
- order placement (origin node)
- order transport (information arc)
- order processing (target node)
- order shipping (target node)
- order transport (shipment arc)
- order receiving (origin node).

5.1.2 Modeling Nodes

The most basic pieces of a supply chain are the nodes and arcs that define the topology of the supply chain. Nodes can be of six types: suppliers, production points, warehouses, distributors, retailers, or customers. A node in a supply chain performs five basic actions with regard to the life cycle of an order:

- order creation
- order placement
- order processing
- order shipping
- order receiving.

Order creation involves the creation and initialization of an order. Orders are generally initially created by one of two sources, an inventory policy signaling that a replenishment order is necessary, or an external demand generated by a customer node. At the time of creation, an order is essentially a desire for an item. In order to act upon that desire the order must be placed.

Order placement is the process that makes the order known to the supply chain, and is not the same as the initial creation of the order. Order placement is the process of preparing an order for transport to its target, the node that will fulfill it. The placement process is the first action that an order undergoes and may include processing delays and costs. The placement process is also where the routing of the order (where to send it for fulfillment) may be determined.

Order processing is the action that attempts to meet the needs or demands of the order. Orders arrive at a node and attempts to fill those orders are made. Fulfillment of an order is made from the finished goods inventory of the node the order has arrived at. If finished goods inventory cannot meet the order's needs, the order must wait until goods arrive in the finished goods inventory. Finished goods arrive as a result of either a processing delay (for suppliers), a production process (for production nodes), or the placement of an order (stocking point nodes). A finished goods inventory is utilized with all types of nodes, except the customer node, though its interpretation changes somewhat with non-production nodes. The finished goods inventory has its most traditional meaning when used with a production node, but can also be interpreted as "regular inventory" when used with stocking point nodes. Or it can be looked at as a "hand-off" point, when used with supplier nodes whose internal operations are considered a "black box." The completion of the order processing action is the point at which the order begins the return trip to its origin. The delays and costs involved with order processing are entirely dependent on the type and characteristics of the node in question.

Order shipping is the process of preparing the fulfilled order for transport to its origination node. Order shipping may involve grouping of certain orders together, prioritizing orders, or the handling and processing of goods to be shipped. There may be costs and delays involved with the shipping process.

Order receiving is the process of accepting an order that has been filled. Orders received are orders that were placed at some point in the past and have traversed their life cycle, being transported, filled, and transported again to return to their origin. The receiving process serves to organize the receipt of goods and ensures they are accounted for in inventory figures or customers-served tallies. After receipt of an order, it is essentially disposed of.

5.1.3 Modeling Arcs

Arcs can be of two fundamental types: information (order) arcs, or delivery (product) arcs. The arcs in a supply chain perform one action in the life cycle of an order, order transport, though it may be performed multiple times during the life of a single order. An arc represents the movement of an order (information or goods), in a single direction, between two nodes. Order transport may be the movement of information, such as the case with demand and replenish orders being sent to their destinations for processing, or the movement of goods, such as the shipment of filled orders, back to their origin. Costs and delays are individual in nature.

5.1.4 Modeling Other Supply Chain Processes

We also include other operations besides those, described above, that define the life cycle of an order. Inventory management, including both materials and finished goods, is one such action. We include such actions by making them processes owned by any node that stocks items. At each node, a separate, policy-driven inventory management system is used to control each component (item) stored. This design provides the greatest flexibility in modeling of component stocking throughout the supply chain. Inventory management systems are contained within individual nodes and operate autonomously based on the needs and conditions at that particular node.

In addition, the external, independent demands that drive the supply chain's overall actions must be included. These demands are distinguished from the dependent demands generated by inventory management systems because they are external to the system and in many cases beyond the direct control of supply chain managers. We represent these demands by making them processes owned by the Customer nodes. Demand characteristics for each demanded item can be individually specified for each customer. This allows for the greatest flexibility in the demand-generated order sequence that drives the supply chain.

5.2 SISCO Supply Chain Library

The implementation of the modeling approach described in the above section is based around a specialized set of Java classes. The SISCO Model Builder is a Java application that incorporates the Silk™ simulation engine. By utilizing the Silk™ primitive classes as a basis, we develop a library of specialized, Silk™-compatible Java classes that represent the various pieces of a supply chain. The SISCO Model Builder then uses the various pieces of this library to create a simulation model from an SCML file that has the same structure and characteristics as the supply chain described (Chatfield 2001).

5.2.1 "X-Classes"

The first part of the supply chain library is a set of approximately 50 Java classes, known collectively as the "x-classes." They are known as the "x-classes" because all begin with the letter "x", standing for XML, which hints at their purpose. The x-classes are data-only classes that provide a means of representing the supply chain data contained in an SCML file as a set of Java classes with the same hierarchical structure. This is needed because the information must be in a Java-accessible format before we can make use of it for model development. This set of classes mimics the structure of the SCML file format with each of the elements defined in the SCML specification having an equivalent x-class, except the root element (*supplyChain*). It allows a straightforward transfer of information from SCML (XML) files to Java compatible data structures. The top-level x-classes are xNode, xArc, xComponent, xAction, xPolicy, xCountry, and xTradingGroup. The xComponent, xAction, xPolicy, xCountry, and xTradingGroup, classes all extend (inherit the data and capabilities of) a parent class called "Entity." The Entity class is a Silk™-provided class that implements (has access to) the Silk™ simulation capabilities and enables an Entity-derived object to run in its own Java thread of execution. The xNode and xArc classes extend the Location class, which contains extra information needed by these types and extends the Entity class itself. In addition to initialization procedures, the xNode and xArc classes are used as parent classes for Node and Arc classes, which are operational classes of the SISCO system. These operational classes are described next.

5.2.2 Operational Classes

Whereas the x-classes are data storage classes, the operational classes represent the object types that will actually perform operations and interact with each other in a manner that simulates the operation of a supply chain. For any object to function properly within the Silk™ simulation framework, the object must be of a type (class) that is derived from the Silk™ Entity class. According to the Silk™ documentation, "the Entity class provides the basis for defining classes that employ the process-oriented simulation extensions to Java that constitute the Silk™ language." Instances (objects) of Entity-derived classes run in their own

Java thread of execution. By running in its own thread of execution, the object executes independently from other simulation objects running in their own threads, much like if each thread of execution were a separate computer. Each thread has its own life span and, most importantly, operates according to its own time-line without being interfered with by other objects' operations. Objects running in separate threads can interact with other objects in the system, a capability which is obviously needed in a simulation environment. The Entity class also provides a process() procedure that contains code for the tasks an object will perform during its life. We "start" the process() procedure when we wish the life of the object to begin. To build the SISCO Library, we create a set of classes that extends (inherit from) Entity and represent the node, arcs, and orders of the supply chain and the tasks that are performed within these elements of the supply chain.

5.2.3 Owner-Manager-Actor Structure

The operational classes follow a fundamental structure that we refer to as the "owner-manager-actor" structure. This approach computationally separates the various management, monitoring, and task oriented operations of the supply chain by creating separate objects to perform each. For example, a node has a number of tasks that occur at that location in parallel, such as order placement, order processing, and inventory management, to name but a few. If each of these processes were implemented as procedures within the node object, they would preempt each other -- inventory management tasks would be performed, while order processing tasks that should be performed at the same time wait. The "owner-manager-actor" structure addresses this problem by creating an object for each task. Each object runs in a separate thread of execution, which allows each to perform its procedures simultaneously, without preempting each other.

The "owner" is the object representing the place in the supply chain at which the tasks are occurring, such as a node. "Manager" objects generally implement tasks, usually policy-related, that the owner must continuously perform. An example of a manager would be an object that handles order processing by constantly checking a queue for orders, or one that handles inventory management by checking the inventory level of an item and comparing it to the policy parameters. When a manager determines that an action needs to be performed, such as production to fulfill an order or placement of an inventory replenish order, an "actor" object is created to perform the action. The reason for this is the same reason that managers are created to perform monitoring duties for their owner (node or arc); to allow simultaneous operations. For example, if an inventory manager object also performed the replenishment ordering action then inventory monitoring would not occur while the ordering is taking place. Thus, to prevent preempting a manager's monitoring activities, "actor" objects are created as needed to handle the actions. In most cases, actions are temporary objects that are disposed once the action has been completed. The "owner-manager-actor" structure, coupled with the threading capabilities of Silk™, allows us to create simulation models that operate like they do in reality, with processing occurring simultaneously.

5.2.4 The Node Class

The most basic need is to represent the nodes and arcs that define the basic structure of the supply chain, and this is where the operational classes begin. The Node and Arc classes are templates for the creation of objects based on Silk™ Entities that represent the nodes and arcs. The most important part of the Node class involves the coordination of the basic actions that occur at a node: order creation, order placement, order processing, shipping, and receiving. The Node class defines Silk™ Resources, Silk™ Queues, and Managers for the basic actions.

The Silk™ Resources represent capabilities (such as equipment) of a node that must be available for an action to be performed. Over the course of a node's life, the various Resources are "seized" when they are required to perform an action and "released" when the action is complete. A seized resource indicates that resource is "in-use" and unavailable unless the current use has been completed. The interim period is the performance time of the action, generally referred to as the "delay."

The Silk™ Queues implemented by the Node precede each of the basic actions, except order creation. These Queues are where orders waiting to have a certain action performed wait.

The Managers defined by the Node make use of the Resources and Queues to control the basic action. An Order Placement Manager that utilizes the Order Placement Queue (where orders wait for placement to occur) and the Order Placement Resource to control the process of Order Placement is created. Likewise, an Order Processing Manager, a Shipping Manager, and a Receiving Manager are created to control those operations. When the Managers determine that it is necessary to perform a basic action, they remove an order from the queue, seize the appropriate resource, and create an instance of an actor class to perform the action. These Actor classes (Order Placement Actor, Order Processing Actor, Shipping Actor, and Receiving Actor) update the Order, delay the appropriate amount of time, and release the resource. When the Actor finishes performing the action, it is disposed of.

Order creation is controlled by one of two types of managers. If the node is a customer node, then the orders are being created from external customer demands. A separate Demand Generator object is created for each item that is demanded by the customer node. The Demand Generator object operates according to the demand charac-

teristics provided by the modeler, creating Order instances at specified intervals, either constant or determined from a distribution, for specified sizes (again, either constant or determined from a distribution). For more complex situations, a policy procedure can be employed to determine demand. If the node is of another type (excepting suppliers, who do not generate orders), the Order instances will be generated by the inventory system in the form of an inventory replenish order. When a Node is created, three Silk™ State Variables are created for each item or component that enters the node (node inputs) and for each item that leaves the node (node outputs). Silk™ State Variables can be continuously monitored by the Silk™ simulation system, triggering procedures that determine the level of the State Variable any time it changes. The Silk™ State Variables are used to store the current inventory level, the amount on order, and the inventory position of each input and output component. In addition, an Inventory Manager is created for each node input and node output as well. The Inventory Managers monitor the inventory levels and position of each component and, based on an inventory policy specified, create and instance of Inventory Actor. The Inventory Actor creates the replenish order, updates the inventory levels stored in the State Variables, and is then disposed of.

Besides creating the necessary Queues, Resources, and Managers to enable the basic actions to be performed, a Node also creates variables used to track the performance of the node. Silk™ Time Dependent variables are used to track the time-dependent statistics such as the utilization rate of the various Resources and the length of each of the Queues. In addition, Silk™ Observational variables are used to track the performance aspects of the Node that are observed occurrence by occurrence, such as the time spent by the Orders in the various Queues.

5.2.5 The Arc Class

The Arc class is structured like the Node class except that it is much simpler, since the Arc performs only one basic action, order transport. A Silk™ Resource, Silk™ Queue, and Transportation Manager are created to control the access to the transportation action. When the Manager indicates a transportation action needs to be performed, a Transportation Actor is created to perform the action and then disposed of.

5.2.6 The Order Class

The method of supply chain modeling we employ is based on the life cycle of an order, so a well-designed representation of an order is important. The Order class extends the Silk™ Entity class so that Order instances can be utilized within the Silk™ simulation system. Each order has a reference to origin node (creator) stored in an "originNode" field and a reference to the destination node, when determined, is stored in a "targetNode" field. References to the order and shipment arcs are also stored in "orderArc" and "shipmentArc" fields. Other characteristics that are included in the Order class include the component the order is for, the amount ordered, and the units of measurement used. Timestamps before and after every action in the order's life cycle are stored in a set of data fields to aid in system and sub-system performance reporting.

The most important part of the Order class is its guidance of the order through the basic life cycle. An order's destination node, as well as the order and shipping arcs, are determined by the Order Placement and Shipping Managers of the node that creates the order. The order's process() method coordinates the life cycle. It starts by placing the order in the Order Placement Queue and then temporarily stalls the process() procedure with a halt() command. When the placement process is complete the order's process() procedure is activated which sends the order to the next point in the life cycle by placing it in the order arc's Transport Queue and halting the process() procedure. This continues for each stage in the life cycle. Thus, the order controls its own sequence of actions, but the nodes determine the details of where those actions will occur.

5.3 Model Generation and Execution

The model generation process involves taking the user input, mainly the SCML file, and generating an equivalent Silk™ simulation model by creating instances of the appropriate SISCO Library classes. The process involves two main actions: parsing the SCML file to generate the appropriate supply chain objects as they are encountered; and invoking the initialize() procedure of a newly created object to create appropriate Resources, Managers, statistical tracking variables, and related structures owned by the object.

To enable the parsing of the SCML file, we utilize Sun's Project-X Technology Release 2 (TR2). Project-X TR2 provides a Java implementation of both SAX and DOM –based parsers. The process of reading the SCML file and creating instances of the appropriate SISCO Library classes is well-suited for a SAX parsing procedure. The potential size and complexity of the SCML document, compared with other XML documents, favors the one-pass, event-based parsing method of SAX over the tree-based parsing of DOM parsers. DOM parsers store a representation of the hierarchy of the entire document in memory before processing can be performed, which is slower and more memory-intensive than the one-pass, immediate action approach of a SAX parser.

The elements representing supply chain "basic constructs" (*node, arc, component, action, policy, country, and tradingGroup*) are level-2 items in the SCML hierarchy and are the heart of a supply chain description. The x-

classes of the SISCO Library previously described are designed to replicate the data structures of these elements and the information they contain in sub-elements and attributes. When a *component, action, policy, country, or tradingGroup* element is found in the SCML file, an object of the corresponding x-class is created and the descriptive information (attributes and sub-elements) from the SCML file is transferred to the new object. The new object is then placed in an array with other objects of the same type.

When the parser encounters the start of a *node* or *arc* element, the processing follows the same structure as the other basic elements, but is a bit more complex. As with elements for the other basic constructs, the start_element() event for a *node* or *arc* element (indicating the beginning of a *node* or *arc* has been encountered) will create a representative class. In this case however, the class created is not an x-class (xNode or xArc). Instead, extensions of the xNode and xArc classes, named Node and Arc are used. The reason for this is that the x-classes are data-only classes that mimic the structure of the SCML language. Representing the operation of nodes and arcs is central to the simulation modeling of a supply chain and, as a result, the needed objects must be more complicated than Java class-based implementations of the SCML *node* and *arc* elements. The start_element() event begins by creating a Node or Arc–derived object. Since Node and Arc extend xNode and xArc, they include all the data fields of those classes. Thus, the transfer of descriptive information and addition to an array occurs in the same manner as with the other basic constructs.

When the end_element() event is executed (i.e., the end of a *node* or *arc* element has been found) the process() procedure of the Node or Arc is invoked, essentially beginning the "life" of the Node or Arc as an entity in the Silk™ simulation system. The first action within the process() procedure is to call the initialize() method, which creates arrays that allow easy access to the node or arc information. Next, this routine performs actions that create Silk™ structures needed to represent the operation of the node or arc properly.

For a node, the Silk™ Queues and Resources for basic supply chain actions (order placement, order processing, shipping, and receiving) are created. The associated Manager objects (order placement, order processing, shipping and receiving Managers) are created as well. In addition, Silk™ State Variables are generated for each input and output of the node, creating materials and finished goods inventory tracking variables. In addition to the State Variables, an Inventory Manager object is generated for each node input and output to perform the actual inventory monitoring. Finally, if the node is a customer node, a Demand Generator object for each component demanded is created. All of the Managers, plus the Demand Generators are separate Silk™ entities running in their own thread of execution and each has its process() procedure "started" by the Node object (entity) after it is created and initialized. The Node entity itself does not perform any processing, it serves as the "owner" of these other Silk™ entities that perform processing for it. Thus, the simulated actions of a node can all occur in parallel because they are being executed in separate threads by separate entities. The process is the same for creating an Arc, but simpler because its only action is transporting orders.

Execution of the simulation model occurs immediately, because the process of "starting" various entities occurs as the model is generated. The parsing procedure ensures that the various simulation entities are initialized and started in the correct order, because it follows the hierarchy as it is laid out in the SCML file. Thus, an element is never initialized before any of its sub-elements, and an entity is never started before all the elements it contains are initialized. Essentially, the process of model generation and model execution are combined.

6 APPLICATIONS

SISCO was developed because simulation modeling can provide valuable insight into the operational characteristics of supply chains. Variation is a reality in all systems, supply chains not excluded. Variation in demands, production yields, transportation times, and cost of goods over time, as well as many other factors, are common in the actual operation of a supply chain. Yet these operational factors are often modeled deterministically.

The most extensive use of SISCO so far has been to investigate the phenomenon of demand variability amplification, often referred to as the Bullwhip Effect. A set of supply chain structures was created using the VSCE and simulated under various conditions to examine the impact of lead time variability and forecasting methods on the severity of the Bullwhip Effect. SISCO was invaluable in the creation, execution, and coordination of the simulations.

The following are some additional aspects of supply chain operations modeling that would benefit from a rigorous simulation-based analysis:

- realistic-sized supply chains are rarely modeled stochastically
- delivery times and costs are assumed linear or static in most supply chain models
- very few supply chain modeling endeavors have taken exchange rate and other global conditions into account, though these can be significant drivers of profitability
- service-oriented supply chains have been only sparingly modeled.

7 CONCLUSIONS

The Simulator for Integrated Supply Chain Operations (SISCO) provides an improved approach to performing

simulation analysis of supply chains. As a fully object-oriented supply chain simulator, SISCO goes beyond current simulators by enabling GUI-based "off-the-shelf" modeling of common supply chain operations, while still allowing access to a full general-purpose programming language for customization if desired. Silk™, the Java-based core of SISCO, provides a robust, flexible, internet-friendly, multi-threaded environment for simulation model development and execution. SISCO addresses the supply chain information sharing problem by integrating SCML file compatibility, allowing sharing of supply chain problem information with users of any SCML compliant modeling tool and utilizing the Visual Supply Chain Editor as the main method for describing the supply chain to be modeled.

In addition to SISCO itself, the SCML parsing routines and data structures in Java and Visual Basic 6, developed as part of the SISCO project, are generic and can be used to make other applications and tools SCML compliant.

In all, SISCO greatly lowers the "barriers to entry" for simulation modeling of supply chains, extends the current state of the art in terms of modeling features, and provides information sharing capabilities in a robust, object-oriented simulation modeling tool.

ACKNOWLEDGMENTS

We would like to thank the Center for Supply Chain Research at Penn State for partial funding of this project. We would also like to thank Rich Kilgore of ThreadTec for all of his help in utilizing Silk™.

REFERENCES

Chatfield, D. 2001. SISCO and SCML- Software Tools for Supply Chain Simulation Modeling and Information Sharing. Unpublished Ph.D dissertation. Department of Management Science and Information Systems, Penn State University, University Park, PA.

Healy, K. and R. Kilgore 1998. Introduction to Silk™. ThreadTec Incorporated. Available online `http://www.threadtec.com`

Mosenhi, P. 1999. An introduction to XML for Java programmers. *Java Pro*, 3(3):48-52. Fawcette Technical Publications, Palo Alto, CA.

AUTHOR BIOGRAPHIES

DEAN C. CHATFIELD is Assistant Professor of Management Science and Information Technology at Virginia Polytechnic Institute. He received his Ph.D. from Penn State University in 2001. His research interests include manufacturing and service supply chain analysis and design, simulation modeling of production and supply chain systems, and the application of meta-heuristics. His email address is <deanc@vt.edu>.

TERRY P. HARRISON is Professor of Management Science at Penn State University. He has teaching and research interests in the areas of supply chain management, large scale production and distribution systems, decision support systems, applied optimization and the management of renewable natural resources. His email address is <tharrison@psu.edu>.

JACK C. HAYYA is Professor Emeritus of Management Science at Penn State University. His research interests lie in the areas of production and inventory management, applied statistics, supply chain management, military systems analysis, and food safety. His email address is <jch@psu.edu>.