Proceedings of the 2004 Winter Simulation Conference
R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

XML-BASED SUPPLY CHAIN SIMULATION MODELING

Dean C. Chatfield

Business Information Technology Dept. (0235)
Pamplin College of Business Administration
Virginia Tech
Blacksburg, VA 24061, U.S.A.

Terry P. Harrison
Jack C. Hayya

Supply Chain and Information Systems Dept.
Smeal College of Business Administration
Penn State University
University Park, PA 16802, U.S.A.

ABSTRACT

We describe a different approach to using XML to support the simulation modeling of supply chains. Instead of using XML to specify the simulation constructs, as most previous approaches do, we utilize XML to describe the supply chain itself. The Supply Chain Modeling Language (SCML) is a general, reusable, platform and methodology independent standard for describing a supply chain's structure and logic. SCML is usable by analysts using many methodologies, including simulation. We describe a sample simulation system (SISCO) that uses SCML files as input. This system uses an algorithm to "map" the SCML file contents to simulation classes contained in a supply chain simulation class library, resulting in an object-oriented simulation model of the supply chain.

1 INTRODUCTION

Markup languages, especially XML, have received significant attention in recent years. Though much focus has been placed on the use of markup languages for authoring purposes, they are also applicable as methods of information structuring, delivery, and transfer. XML has been successfully used in a number of fields (chemistry, e-commerce, etc.) for defining powerful, flexible data standards. We believe the benefits of an open, standard data format for supply chain modeling can make supply chain analysis, especially via simulation modeling, more robust, yet more accessible.

2 MARKUP LANGUAGES

A markup language describes the structure and/or appearance of information contained in a document by the addition of descriptive symbols (markup characters, or "tags") within the document itself and by imposing a certain structure which the document must follow. (Tittel et al., 1998).

A markup language can be one of two basic types, a *meta-language* or a *task-specific markup language*. A meta-language is a markup language that is used for creating task-specific languages. Meta-languages provide a means for defining the markup symbols, such as elements and attributes, as well as the overall structure of a document. Essentially, a meta-language allow one to create a task-specific markup language by defining all the "pieces" that may exist in a document as well as the "rules" for putting those pieces together to form a valid document. Examples of meta-languages are the Standard Generalized Markup Language (SGML) and XML.

A task-specific markup language is one that is used for describing a certain type of document (for example, a hypertext document) or information about a certain subject (for example a supply chain). Examples are the HyperText Markup Language (HTML), which is a markup language that describes how a hypertext document should be displayed by a web browser, the Chemical Markup Language (CML), which provides a standard means for describing the components, structure, and characteristics of a chemical compound, and the Mathematical Markup Language (MathML) used for equation representation. HTML is considered an application of SGML while CML and MathML are XML-based.

For example, HTML is a task-specific language designed to describe how a hypertext document should be appear. The HTML specification is composed of a specific set of elements (sometimes referred to as "tags"), attributes, and other constructs that define whether part of a document should appear bold, centered, displayed as a hyperlink, etc. The HTML specification also defines a basic structure that must be employed when creating an HTML document. Therefore, the HTML specification defines the pieces than may be used and the rules governing how these pieces may be used when constructing an HTML document. The HTML specification was defined by the World Wide Web Consortium using the SGML meta-language to create the HTML Document Type Definition (DTD), which describes the pieces and the rules in a standardized format.

2.1 SGML and XML

SGML is the precursor to XML and is derived from GML, which was developed in the 1960s as part of a legal information processing project by IBM (VanHerwijnen, 1994). SGML was standardized in 1986 by the International Standards Organization as a universal means for describing a document's structure and appearance in separate, logical statements. This desire to separate the content (data), structure, and appearance of a document into separate units was championed initially by the publishing industry; thus, markup languages are still closely associated with publishing (VanHerwijnen, 1994).

XML can be likened to SGML "light" because XML takes much of the important functionality of SGML, such as document type definitions (DTDs), and implements it in a streamlined package (Tittel et al., 1998). As a result, XML parsers, processors, and programming tools are being developed and more widely implemented. Examples of this include Microsoft Internet Explorer and Netscape Navigator being XML capable, and the recent flow of Java and ActiveX XML parsers made available from Microsoft and other vendors. An active XML coordination group exists as part of the World Wide Web Consortium (W3C) to ensure XML continuity and to foster further development of XML and complementary technologies.

2.2 Document Type Definitions

Much of XML's power comes from the ability to define a Document Type Definition (DTD), which is the foundation of a task-specific language. A DTD defines the "pieces" and "rules" that make up a class of documents, essentially creating a custom type of document for a particular use. The DTD, written with a special DTD-specific syntax, is the "hub" of the XML system. The basic parts of a DTD are the definition of elements (classes such as "node" or "paragraph" or "atom"), definition of attributes (descriptive data, generally about elements, such as a node's name), and definition of entities (nonstandard or external symbols, files, and images). This definition of elements, attributes, entities, etc. is basically a specification of the "pieces" that may be used. The DTD defines which of the above can occur, with what names, in what order, how often, and so on (i.e. the "rules").

2.3 Parsing XML-Based Documents

To utilize XML-based documents, one must have software that understands and utilizes the format, much like a word processing program must understand and utilize the RTF (Rich Text Format) file format if one is to create, read, or edit a document stored in that format (VanHerwijnen, 1994). A parser is the "engine" that performs reading, writing, processing, and validation (DTD agreement checking) tasks for XML documents and enables developers to incorporate XML capabilities into their software.

There are two basic types of parsers, categorized by how they process a document. The first type are called SAX (Simple API for XML) parsers. These are event-based parsers, meaning that a document is read serially and when certain markup characters are found (such as a "start document" tag, a "start element" tag, or an "end element" tag) an event is fired. The application developer writes the code that will be executed when the various events are fired to customize the parser. The parser reads the document a single time from start to finish, with the resulting execution of event procedures providing the desired results (such as document display, processing, or other related tasks).

The second type of parser is called a DOM (Document Object Model) parser. A DOM parser reads the XML document and creates a temporary in-memory model of the document's element hierarchy. The in-memory model contains a DOM "node" to represent each element in the XML document. One can then navigate through the tree structure of the in-memory model to locate, extract, or edit information. The structure of the in-memory model itself can be altered (addition or removal branches, for example) as well. The model resides in memory indefinitely (until removed) and changes to the in-memory model can be saved, resulting in changes to the XML document used to create it.

2.4 Previous Uses of XML in Simulation Modeling

XML has a number of uses in simulation modeling and simulation software. The most basic use is to create standardized data formats for basic simulation input and output files. Open, standardized representations for data such as user-defined and/or empirical distributions, time-series output, basic statistical measures of performance, etc. are valuable for cross-platform computing, analysis, and results sharing.

A more proprietary use of XML would be as a basis for a simulation tool's model storage file format. This would be similar to the way Microsoft has migrated to XML-based files for storing Microsoft Office documents (as of Office 2003). The user is not aware of the difference from previous file formats and does not change the way they enter information. The use of an XML-based file format "internally" within a simulation software package would provide the benefit of standardized data access, making it easier to add features and plug-ins, enable cross-product integration, and allow 3rd party add-ons (provided the DTD is available to developers).

XML has been utilized for communication between a simulation model and other software or simulation models. Qiao, Riddick, and McLean (2003) and Lu, Qiao, and McLean (2003) describe the use of an NIST (National Institute for Standards and Technology) developed XML-based markup language for standardized exchange of

manufacturing information between applications, including simulation applications. Additionally, Fishwick (2003) describes two XML-derived languages (MXL and DXL) designed to support simulation multi-modeling of all types.

A grander vision of the use of XML in simulation is take by those attempting to create a universal simulation language and syntax via an XML-defined markup language. Kilgore (2001) and Kilgore (2002) discuss the open source SML (Simulation Modeling Language) initiative to create a platform-independent, open source simulation language. Wiedemann (2002) extends this idea by proposing an XML-based, language-independent standard for representing SML statements. Code converters are used to generate language specific (ex. Java, C++) statements from the generic XML-based version when the model is to be executed. An alternate XML-based simulation project is described by Reichenthal (2002). This project involved the creation of the Simulation Reference Markup Language (SRML) to allow standardized descriptions of simulation models, and the Simulation Reference Simulator (SR Simulator) to process and execute SRML models.

We are not aware of any applications of XML, other than ours, to specifically support supply chain simulation modeling.

3 XML-BASED SUPPLY CHAIN SIMULATION

We approach the XML-aided simulation of supply chains in a different manner than those projects and initiatives described in the above section. We use an XML-based markup language, the Supply Chain Modeling Language (SCML) that is specific to supply chain modeling, but not to the simulation methodology. SCML is a general markup language for storing the basic structural and logical information needed to quantitatively model a supply chain. This format is designed to be a universal supply chain description language that analysts can use for supply chain decision support tools utilizing various methodologies.

We take the description in an SCML file and generate an equivalent object-oriented simulation model by mapping the contents of the SCML file to a class library of fundamental supply chain constructs. The appropriate simulation objects (class instantiations) are created and the simulation is executed. Thus, our approach requires:

- A way to create an SCML document
- A library of fundamental supply chain simulation constructs (classes) that can operate in an object-oriented simulation environment.
- A mapping routine to generate a set of objects from the library that correspond to the SCML file contents.

By using this approach the user is primarily concerned with creating an accurate description of the supply chain, not simulation modeling issues. The model generating software that converts the SCML supply chain description to the object-oriented simulation model handles the modeling issues. Also, by creating a domain-specific simulator, this approach can offer modeling depth and robustness, while still allowing extensibility, depending on the platform used to create the supply chain class library and associated conversion ("mapping") software. The fact that the basic supply chain description is platform and methodology independent means that modeling tools, such as a simulation tool, can be built in a modular fashion, essentially as a set of supply chain decision support methodology plug-ins that center around the SCML format as a "hub."

4 SCML - A GENERALIZED SUPPLY CHAIN MODEL DESCRIPTION LANGUAGE

The Supply Chain Modeling Language (SCML) consists of an XML Document Type Definition that provides a flexible, object-oriented model description language for storage and use of supply-chain information. The language is composed of a set of basic elements corresponding to broad groups of supply-chain components. The SCML concept has undergone several refinements and extensions in the five years since its initial presentation (Chatfield, Harrison, and Hayya, 1999).

4.1 Rationale for Using XML

The XML language provides a flexible tool for development of special purpose description languages. XML's inherent hierarchy provides for straightforward implementation of the elements and attributes that form the basis of SCML. XML implements those object-oriented capabilities most important to the design of a language like SCML, namely the object and attribute paradigm for data representation. Although XML is not completely object-oriented, its DTD syntax does not include inheritance as a core capability, the limitations will not interfere with development of SCML and can be circumvented with judicious DTD design if necessary (by the creative use of entities). Due to its object-oriented characteristics, increasing acceptance, inherent support for custom data formatting (through DTDs), and the successful implementations in other fields, XML is the logical choice as the meta-language "building block" upon which SCML is formed.

4.2 Basic Structure

The first step in designing the SCML DTD involved determining the overall strategy for representing a supply-chain in a hierarchical, object-oriented manner. This set of the most primitive supply-chain information classes must be complex enough so as to not place undue restrictions on what supply-chain features can be described, but must also

be simple enough to provide an elegant and scalable representation of supply-chains. In addition, since an object-oriented style of representation is desired, the basic categories should also be determined so that each can be represented as a relatively self-contained unit that interacts with other self-contained units of the same or other types. The elements in an SCML file should be easy to map to a set of equivalent objects, attributes, and methods in an object-oriented modeling environment. As a result, five basic elements were chosen:

- Node
- Arc
- Component
- Action
- Policy.

This set provides a simple, logical framework for describing a supply-chain that is easily scalable and allows a very detailed supply-chain description to be generated. Each of these elements can be roughly interpreted as a "class" in the object-oriented parlance, and individual occurrences of these elements can be interpreted as objects (instances of a class).

The SCML basic elements each utilize a set of attributes and sub-elements to describe a specific instance of the basic element. For example, by setting the attributes and sub-elements appropriately, we can designate a particular instance of the node class as a production facility and another instance as a different type of production facility or even as a warehouse. Thus, given proper element and attribute assignments, any supply-chain can be described.

4.3 The SCML Specification

As necessitated by XML syntax restrictions, any supply-chain described must have the information formatted into a tree-like structure, with a single root node. The root element of any SCML file is designated by the name supplyChain. The supplyChain element contains five container elements, each holding zero or more elements of that category (except "nodes", which contains one or more node elements). The following is the DTD section (content model) describing the root and container elements.

```
<!ELEMENT supplyChain (nodes, arcs, components, actions, policies)>
<!ELEMENT nodes (node+)>
<!ELEMENT arcs (arc*)>
<!ELEMENT components (component*)>
<!ELEMENT actions (action*)>
<!ELEMENT policies (policy*)>
```

Each of the basic elements has a content model that specifies the sub-elements and attributes that further define the element. As an example, the DTD markup below is used to define an "arc" element's sub-elements and attributes.

```
<!ELEMENT arc (arcContainer, arcCosts, arcTravelTime, arcPolicies, initialLevels)>
<!ATTLIST arc
name ID #REQUIRED
sourceNode IDREF #REQUIRED
destinationNode IDREF #REQUIRED
mode (land | rail | sea | air | telecomm | other) "land"
distance CDATA #IMPLIED
unitOfDistance CDATA "miles"
capacity CDATA #REQUIRED
unitOfCapacity CDATA "unit"
containersOnHand CDATA #IMPLIED>
```

The sub-elements will have content models of their own, possibly specifying further sub-elements and attributes to hold the information. The SCML specification stores much of the supply chain information in XML attributes because content restriction and information extraction are much simpler when using attributes.

4.4 SCML Document Creation

For creating and editing SCML files, a custom editor that enforces the DTD restrictions is the best choice since it is easiest for the end user. Such an editor, the Visual Supply Chain Editor (VSCE) was created just for this purpose. The VSCE is a Windows application that allows a user to graphically depict the supply-chain structure (nodes and arcs) and then access specialized forms that step the user through the process of further describing the various parts of a supply-chain. The result is a graphic application that allows the user with little or no knowledge of the SCML DTD to create an SCML file describing a supply-chain.

4.5 SCML Document Processing

Creating software that can utilize and understand the SCML file format necessitates the inclusion of SCML file processing (e.g.. extraction of information) capabilities. This requires the SCML DTD, which provides the specifications necessary to understand the SCML file structure, and an XML parser to extract the information from the file, since SCML is based on XML. In addition to the DTD and the parser, a software developer will generally create a set of data structures to hold the information, for example a set of C++ or Java classes that mimic SCML's hierarchy of elements and attributes. To simplify the use of SCML files we have created a set of parsing routines and data structures in several languages. We have created a set of Java classes, a set of .NET classes, and a set of Visual Basic 6.0 user-defined data types that correspond the SCML DTD's hierarchy. SCML parsing routines have been created for the Java, Visual Basic 6.0, and Visual Basic .NET languages. These routines are designed to transfer the SCML file contents to a corresponding set of data structures to enable easy access to and manipulation of the information by software.

5 GENERATING A SIMULATION MODEL FROM AN SCML FILE

Section 4 discussed the SCML format, including document creation and the basics of document processing. This leaves two remaining pieces necessary for our supply chain simulator to operate.

First, a library of fundamental supply chain simulation constructs (classes) that can operate in an object-oriented simulation environment is needed. Since SCML is hierarchical in nature and represents a supply chain in a roughly object-oriented manner, utilizing an object-oriented simulation environment is a logical choice. The simulation environment may be a commercial product, open-source, academic, or custom designed if so desired. The environment should allow interaction with external data sources and/or code routines to allow the mapping routine to generate the appropriate model. The library should include simulation classes for nodes, arcs, orders, and other pieces of the supply chain although the actual set of classes will depend on the modeling approach chosen.

Second, a mapping routine to generate a set of simulation objects, from the class library, that corresponds to the SCML file contents is needed. This mapping routine will have an SCML parsing routine at its core. However, the mapping routine must go further than just reading the SCML file contents and placing the information in a data structure that mimics the file. The mapping routine must instantiate the correct simulation classes and then, depending on the simulation environment being utilized, coordinate the initialization process and other pre-execution activities.

6 SISCO - A SAMPLE IMPLEMENTATION

We have developed an implementation of the components discussed above, which we have named the Simulator for Integrated Supply Chain Operations (SISCO). In the following sections we discuss the details of this sample implementation (Chatfield, Harrison, and Hayya, 2001).

The centerpiece of the SISCO system is the SISCO Engine. This module translates the supply chain information provided by the user into an equivalent Silk™-based supply chain simulation model. The SISCO Engine is comprised of

- The SISCO Library,
- The SISCO Automated Model Mapper (SAMM),
- Simulation Management Routines.

6.1 Simulation Environment

For the development of SISCO we chose to utilize the Silk™ simulation environment. Silk™ is a Java-based, multi-threaded, discrete-event simulation framework created by ThreadTec Inc. Silk™ consists of a set of Java classes that provide low-level, core simulation constructs from which users create simulation models. Examples of these constructs include entities, resources, queues, statistical tracking variables, the system clock, random number generators, and other fundamental pieces of discrete event simulation. Users build a Silk™ simulation model by developing a Java program that utilizes the Silk™ simulation classes, which offers flexibility because the users have access to all the capabilities of Java, a general-purpose programming language, while creating a simulation model. Users can extend the Silk™ classes, as well as include any custom logic desired by creating additional Java routines. We felt the flexibility of Silk™ constructs coupled with power of having access to the entire Java programming language made Silk™ the best choice for development of SISCO.

6.2 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.

6.2.1 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).

6.2.2 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 all of the basic actions in the life cycle except order transport.

Order creation involves the creation and initialization of an order. *Order placement* is the process that makes the order known to the supply chain, and is the process of preparing an order for transport to its target, the node that will fulfill it. *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. *Order shipping* is the process of preparing the fulfilled order for

transport to its origination node. *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.

6.2.3 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.

6.2.4 Modeling Other Supply Chain Processes

We also include other operations besides those, described above, that define the life cycle of an order. For example, 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.

6.2.5 Owner-Manager-Actor Structure

The operational classes follow a fundamental structure that we refer to as "owner-manager-actor." This approach computationally separates the 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. 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 where 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 replenishment order, an "actor" object is created to perform the action. In most cases, actors are temporary objects that are disposed off once the action has been completed. The "owner-manager-actor" structure, coupled with the multi-threading capabilities of Silk™, allows us to create simulation models that operate as in reality, with processing occurring independently and simultaneously.

6.3 SISCO Supply Chain Library

The implementation of the modeling approach described in the above section is based around a specialized set of Java classes. By utilizing the Silk™ primitive classes as a basis, we develop a library of specialized, Silk™-compatible Java classes, the SISCO Library, that represent the various pieces of a supply chain. The SISCO Automated Model Mapper 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). The SISCO Library consists of two parts, the x-classes and the operational classes.

6.3.1 "X-Classes"

The first part of the supply chain library is a set of approximately 50 Java classes, known collectively as the "x-classes" ("x" for XML). 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, and xPolicy.

6.3.2 Operational Classes

Whereas the x-classes are data storage classes, the operational classes represent object types that will 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. 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. The operational classes of the SISCO Library are a set of classes that extend the Silk™ Entity class and represent the nodes, arcs, and orders of the supply chain, as well as the managers and actors that control and perform tasks within these elements of the supply chain.

The operational classes of the SISCO Library include the Order, Node, Arc, and multiple Manager and Actor classes. Our modeling paradigm focuses on the life-cycle of an order, so a well-designed representation of an order is important. The most important part of the Order class is its guidance of the order through the life cycle. With the origin and target nodes determined, the order controls its own sequence of actions, but the nodes determine the details of where and how those actions will occur. The Order class provides a representation of both demand (customer) and replenishment orders, and includes a basic structure for storing individual order information, plus the logic (code) needed for guiding the order through its lifecycle and recording statistics along the way.

Besides accurately representing an order, the most basic modeling need is to represent the nodes and arcs that define the basic structure of the supply chain. The Node and Arc classes are templates for the creation of objects, based on Silk™ Entities, that represent the nodes and arcs. These classes contain all the data structures of the x-classes they extend (xNode and xArc), but also include methods (coded routines) to enable the creation and control of the Managers, variables, arrays, queues, resources, statistical tracking variables, and other structures necessary to simulate the operation of the node or arc. 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 Arc class is similar to the Node class except it is much simpler because arcs only perform one basic action, order transport.

The Manager classes include the OrderPlacementManager, OrderProcessingManager, ReceivingManager, ShippingManager, TransportationManager, InventoryManager, and DemandGenerator classes that form the basis for objects which control the basic actions that occur at the nodes and arcs of the supply chain. The Manager classes contain the queue monitoring, logic, and actor creation routines necessary for a Manager object to independently monitor and control one of the basic actions and to create an appropriate actor object when that action needs to occur.

For example, the Managers created by a Node make use of the Resources and Queues to control the basic actions. An Order Placement Manager utilizes the Order Placement Queue (where orders wait for placement to occur) and the Order Placement Resource to control the process of Order Placement. Likewise, an Order Processing Manager, a Shipping Manager, and a Receiving Manager are created to control those operations. Order creation is handled by DemandGenerator Manager objects if the demand is from a customer, or by InventoryManager objects that generate replenishment orders based on an inventory policy if the demand is internal to the supply chain.

The Actor classes define templates for finite life-span objects that are created to allow multiple independent, simultaneous actions to occur in the model. The actor objects are created to perform a specific action one time after which they are destroyed. The Actor classes include the OrderPlacementActor, OrderProcessingActor, ReceivingActor, ShippingActor, and TransportationActor. The relationships between the Nodes and Arcs ("owners"), the Managers, and the Actors is fundamental to the way SISCO operates.

6.4 SISCO Automated Model Mapper (SAMM)

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.

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 be-fore 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, and policy) are the heart of a supply chain description. When a component, action, or policy 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 that 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.

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 be-

ginning 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.

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. Once SAMM is through creating the appropriate objects and initialization is finished, the model is ready to execute.

7 CONCLUSIONS AND FURTHER WORK

We have shown that an XML-based, platform and methodology independent language for describing a supply chain's structure and logic (SCML) can be effectively used in a simulation system. The SISCO system is able to convert these general supply chain descriptions into object-oriented simulation models. We believe that tools that support XML-based standards are desirable in this age of cross-platform and cross-methodology modeling and analysis endeavors.

To extend this work we would like to create supply chain class libraries and model mappers for other platforms, such as SimKit,(Java), SML (.NET), or an object-oriented commercial simulation package.

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 for all of his help.

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.

Chatfield, D.C., T.P. Harrison, and J.C. Hayya. 2001. SISCO: A Supply Chain Simulation Tool Utilizing SILK and XML. In *Proceedings of the 2001 Winter Simulation Conference*, ed. B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds., 614-622. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Chatfield, D.C., T.P. Harrison, and J.C. Hayya. 1999. SCML: A Generalized Supply Chain Modeling Language. In *Program of the 1999 INFORMS Conference*. November 7-10, 1999, Philadelphia, PA. At <http://www.informs.org/Conf/Philadelphia99/TALKS/WB09.html> [Accessed July 10, 2004]

Fishwick, P. A. 2002. Using XML for Simulation Modeling. In *Proceedings of the 2003 Winter Simulation Conference,* ed. S. Chick, P. J. Sanchez, D. Ferrin, and D. J. Morrice, 616-622. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Kilgore, R. A. 2001. Open Source Simulation Modeling Language (SML). In *Proceedings of the 2001 Winter Simulation Conference*, ed. B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds., 607-613. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Kilgore, R. A. 2002. Simulation Web Services With .NET Technologies. In *Proceedings of the 2002 Winter Simulation Conference,* ed. E. Yucesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, 841-846. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Lu, R. F., G. Qiao, and C. McLean. 2003. NIST XML Simulation Interface Specification at Boeing: A Case Study. In *Proceedings of the 2003 Winter Simulation Conference,* ed. S. Chick, P. J. Sanchez, D. Ferrin, and D. J. Morrice, 1230-1237. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Qiao, S., F. Riddick, and C. McLean. 2003. Data Driven Design and Simulation System Based on XML. In *Proceedings of the 2003 Winter Simulation Conference,* ed. S. Chick, P. J. Sanchez, D. Ferrin, and D. J.

Morrice, 1143-1148. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Reichenthal, S.W. 2002. Re-Introducing Web-Based Simulation. In *Proceedings of the 2002 Winter Simulation Conference,* ed. E. Yucesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, 847-852. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Tittel, E., N. Mikula, R. Chandak. 1998. *XML For Dummies*. Foster City, CA: IDG Worldwide Inc.

VanHerwijnen, E. 1994. *Practical SGML, 2nd edition.* Boston, MA: Kluwer Academic Publishers.

Weidemann, T. 2002. Next Generation Simulation Environments Founded on Open Source Software and XML-Based Standard Interfaces. In *Proceedings of the 2002 Winter Simulation Conference,* ed. E. Yucesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, 623-628. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

AUTHOR BIOGRAPHIES

DEAN C. CHATFIELD is Assistant Professor of Business Information Technology at Virginia Polytechnic Institute. He received his Ph.D. in Management Science from Penn State University. 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 Supply Chain and Information Systems 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>.