SISCO: THE SIMULATOR FOR INTEGRATED SUPPLY CHAIN OPERATIONS

Dean C. Chatfield
Department of Business Information Technology
Pamplin College of Business Administration
1007 Pamplin Hall (0235)
Virginia Tech
Blacksburg, VA 24061
540-231-4593
deanc@vt.edu

Terry P. Harrison
Professor, Department of Supply Chain and Information Systems
The Smeal College of Business Administration
Pennsylvania State University
509 Business Administration Building
University Park, PA 16802
814-863-3357
tharrison@psu.edu

Jack C. Hayya
Professor Emeritus, Department of Supply Chain and Information Systems
The Smeal College of Business Administration
Pennsylvania State University
303 Beam Bldg
University Park, PA 16802
814-865-1461
jch@psu.edu

* Partially supported by the Center for Supply Chain Research, Smeal College of Business, Penn State University

ABSTRACT

We discuss SISCO, the Simulator for Integrated Supply Chain Operations, a Java-based decision support tool that simplifies supply chain simulation model development. SISCO relieves the user of both simulation modeling and programming tasks, making this methodology available to a wider audience. It allows a user to specify the structure and policies of a supply chain with a GUI-based application and then store that specification in the open, XML-based Supply Chain Modeling Language (SCML) format. SISCO maps the contents of the SCML file 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 with the power and flexibility of a full object-oriented programming language that provides unparalleled supply chain simulation model detail and flexibility.

1 INTRODUCTION AND MOTIVATION

We developed the Simulator for Integrated Supply Chain Operations (SISCO) to provide researchers and practitioners with a robust yet flexible simulation tool designed specifically to address the complexities in modeling supply chains. SISCO takes the idea of a supply chain "simulator," examples being Simulation Dynamics' Supply Chain Builder and IBM's Supply Chain Analyzer, and develops it further [1].

1.1 Major Design Goals

We believe simulation modeling of supply chains is an important decision support technique and that the current state of the art needs refinement in a number of areas, including accessibility, flexibility, model depth, and others. The development of SISCO was driven by a number of design goals, described in the following sub-sections.

1.1.1 Accessibility and Extensibility

Currently, the development of complex supply chain simulation models requires the modeler to possess a number of skills, including programming and simulation modeling capabilities. SISCO relieves the user from modeling and programming tasks, and allows one to easily create simulation models of complex systems with a straightforward, GUI-based interface. This capability gives a large population of users, including many that would not normally be able to create supply chain simulation models, access to the advantages of simulation. Additionally, SISCO frees the user to focus on more important aspects, such as gathering information and using simulation to analyze various scenarios. While SISCO allows novice users easy access to simulation modeling, the extensibility and customization capabilities needed by "power users" are not compromised. For technically capable users who wish to customize SISCO, the addition of user-defined code is straightforward given that

SISCO is Java-based. The user may augment SISCO via additional routines developed in Java. This extensibility is an important advantage of SISCO.

1.1.2 Powerful, Detailed Representation

As a simulation modeling tool, our objective was to allow users of SISCO to describe a supply chain's operations with a degree of depth and complexity that is unavailable with current simulation tools, commercial or non-commercial. SISCO's depth of model detail is state-of-the-art, including aspects such as explicit definition of arcs and a multi-step order life-cycle.

This desire for supply chain models with depth and complexity lead to our decision to employ an object-oriented, agent-type approach. A simulation application, especially one focused on supply-chain simulation, is a natural fit for object-oriented development, since each "node" in the supply-chain can be represented by an object, providing a natural mapping from reality to simulation model. Using object-oriented development techniques also assists in the implementation of the accessibility goal of SISCO by enabling the development of set (library) of reusable supply-chain constructs. This library of constructs, the SISCO Library, is fundamental to the automated simulation model generation that makes SISCO so easy to use for novice model builders, while the ability to extend these constructs is what makes SISCO extensible. Pre-built object-oriented constructs make SISCO a very efficient supply-chain analysis tool.

To accurately model the interactions *between* supply chain participants, we chose to take an agent-style approach, where each active participant in the supply chain is represented by an autonomous object and the set of objects that comprise the supply chain simulation model interact and operate by sending messages among themselves. SISCO generates supply chain models that employ this "agents" approach. Such an approach requires an object-oriented, multi-threaded platform, which we satisfy in the combination of ThreadTec's Silk simulation framework [7] and the Java programming language---both key building blocks of SISCO.

1.1.3 Open Storage Format

The absence of an open, standardized method of storing a representation about a supply chain's structure and operations deters supply chain research and modeling. Each modeling effort includes creating a new method for representing a supply chain's structure and policies, which inhibits sharing of information and generally results in continual duplication of effort. We believe it is important to use a general, structured method of supply chain information storage in the SISCO system.

We employ the Supply Chain Modeling Language (Chatfield [1]), as a standard method for describing the structure and policies of a supply chain. SCML is an XML-based, platform-independent, methodology-independent, means of storing the structural and managerial information that describes a supply chain's layout and operational characteristics. It is a set of elements, attributes, and document formatting rules that create a systematic format for supply chain information storage. SCML is defined by a document type definition (DTD), written in XML. Much as HTML is a set of elements, attributes, and rules for describing how hypertext is to be displayed by a browser, SCML is a set of elements, attributes, and rules for storing supply chain information. SCML provides a universal language with which users of various analytical tools, including simulation tools, can store and exchange supply chain information. The decision to use this open storage format, the Supply Chain Modeling Language (SCML), allows users of SISCO to share supply chain descriptions not only with other SISCO users, but with users of any SCML-compliant modeling tool.

1.1.4 Java-Based Development

Utilizing a Java-based core (the SISCO Engine) not only enables our other design goals, such as accessibility, object-orientation, and extensibility, but also provides a number of other advantages over other possible development languages. In addition to being a fully object-oriented language, Java is a multi-platform development tool that enables the SISCO Engine's use on the widest possible user base by supporting all three major platforms, Windows, Unix, and MacOS. Also of importance is availability of well-developed XML parsers written for Java. The threading capabilities of Java enable agent-style modeling and even distributed processing possibilities. Finally, internet "friendliness" is ensured by the Java-based execution system, providing flexibility in the ways that SISCO is delivered, such as in application form, applet form, or served from a remote server.

1.2 Benefits and Applications of SISCO

We developed SISCO because simulation modeling can provide valuable insights into the operational characteristics of supply chains. Variability is endemic in all systems, and certainly so in supply chains. Variation in demands, production yields, transportation times, and cost of goods over time are common in the actual operation of a supply chain, yet these operational factors are often modeled deterministically.

The following are issues in supply chain decision-making that would support the use of simulation analysis in supply chain management:

- Realistic-sized supply chains are rarely modeled stochastically.
- Delivery times and costs are assumed linear or static in most supply chain models.
- Few supply-chain modeling efforts have taken exchange rates and other global conditions into account, though these can be significant.
- Service-oriented supply chains have been only sparingly modeled.

We have already used SISCO to investigate the phenomenon of demand variability amplification, often referred to as the Bullwhip Effect (BWE). By examining the impact of lead-time variability and forecasting methods on the severity of the BWE [4].

1.3 Organization of Paper

This paper is organized into seven sections. Section 2 is an overview. The three stages of SISCO are described in Sections 3-5. Section 6 discusses issues related to implementation of the SISCO modeling platform. Section 7 summarizes our work to date and offers further development directions. To conserve space, we have placed technical details about SCML in a working paper [3], also available on the web.

2 OVERVIEW OF THE SISCO SYSTEM

Operation of SISCO involves three stages: *definition*, *translation*, and *execution.* The structure of the SISCO system, dictated by these stages, includes the following three modules, each of which will perform an important role in the final system:

- The Visual Supply Chain Editor (VSCE)
- The Experiment Designer (ED)
- The SISCO Engine.

A conscious decision was made to make the VSCE, the ED, and the SISCO Engine individual stand-alone applications, as opposed to creating a single, monolithic SISCO application. As stand-alone applications, the components 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. Our idea is to begin the creation of a supply-chain modeling and analysis "workbench" which allows modular tools to be "plugged-in," as needed. SISCO's heart, the SISCO Engine, is a simulation module created with this goal in mind. Additionally, by separating the SISCO system into modules, we allow distributed usage where the user executes the VSCE and ED modules locally and submits the model information to a remotely served SISCO Engine. By separating the Java-based Engine from the rest of the system, this architecture admits the possibility of Web-based distributed processing.

At the definition stage, the users describe the supply chain, as well as the experimental conditions. The Visual Supply Chain Editor allows a user to "build" a supply chain through a graphic user interface and then specify relevant logical and descriptive information. The nodes and arcs are subsequently defined with greater detail, as are the processes, materials, products, and managerial policies that interact with the nodes and arcs. The VSCE is used to define a supply chain's structural and managerial aspects. Please see figure A1 for a screenshot of the VSCE and [3] for further visuals and details of the VSCE. This supply chain description is stored in the open, XML-based SCML file format. The reader is referred to [3] for the complete details of the SCML format. The Experiment Designer is used for defining simulation-specific and experimental information, which is stored in an XML-based experiment (EXP) file.

At the translation stage, the SISCO Engine performs automated model generation, resulting in a Silk™ simulation model. Silk™ is a Java-based object-oriented simulation framework that provides core simulation capabilities in the form of Java classes [4].   The SISCO Library is a set of Silk™-derived classes that represent the participants, materials, and actions occurring in a supply chain. It consists of data classes that mimic the information structure of the SCML file format, along with supply-chain-oriented simulation building blocks created with the Silk™ framework. The SISCO Engine maps the SCML file contents to the SISCO Library, creating the appropriate Java objects, which form a Silk™-based simulation model.

The SISCO Engine, in conjunction with a Java compiler, is then used to execute the simulation model. The output of the SISCO system is extensive and useful for various types of analysis. First, the user is presented with a simulation summary detailing the performance characteristics for each node and arc's activities over all replications. Second, the user is provided with a data file containing the performance characteristics for each individual replication, and another data file containing time-series data. Please see Figures A5, A6, and A7 for screen representations of the three output files. The SISCO Engine uses the SCML and EXP files as input, processes the information contained in those files to automatically build a Java-based supply chain simulation model, and outputs simulation summary, individual replication, and time series data on the performance, the supply chain participants, and the supply chain as a whole. See Figure 1 for a graphic overview of the SISCO system.

3 DEFINITION: DESCRIBING THE SUPPLY CHAIN AND EXPERIMENT

3.1 Visual Supply Chain Editor

The Visual Supply Chain Editor (VSCE) is a graphic tool for defining the supply chain to be simulated. In keeping with our desire to create stand-alone applications that have utility outside of the SISCO system, we use the Supply Chain Modeling Language (SCML) as a standard method for describing the structure and logic of a supply chain. The Visual Supply Chain Editor is a graphic SCML editor, allowing the user to define a supply chain's structure and characteristics and then save this information as an SCML file. Once the user has stored the supply chain information as an SCML file, that information can be used by any SCML-compliant application.

Information 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 and logical design aspects of the supply chain must be defined. We divide the supply chain into five basic types of constructs: *nodes, arcs, components, actions, and policies.*

The *nodes, arcs*, and *components* constitute the physical aspects of the supply chain. *Nodes* represent locations within the supply chain, such as a factory. We allow a user to create suppliers, production facilities, warehouses, distributors, retailers, and customers. 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 supply-chain topology. Beyond the basic topology defined by the nodes and connecting arcs, 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.

Arcs represent the connections between nodes and are defined by the nodes at their endpoints. Additionally, we define the arc's mode (land, rail, air, water, or telecommunications), the capacity, the container size, the transportation rates and costs, maintenance and expansion costs, as well as the policies that may control the arc's actions. In addition to nodes and arcs, a supply chain's components must be defined. *Components* include materials, finished goods, labor, currency, and other items that are consumed, transported, created, or otherwise utilized in the supply chain. The value, physical characteristics (if applicable), and the methods of creation must also be defined.

Beyond the physical aspects of the supply-chain, the actions and controlling logic of the supply-chain must be defined. *Actions*, which include demand or replenishment order placement, order processing (production), transportation, receiving, order placement, or shipping a delivery. Also included 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. Finally, the *policies* of the supply chain must be defined. Policies define relationships between supply chain participants, the control or conditions under which actions occur, or otherwise specify the controlling logic of the supply chain. The VSCE provides the user with a friendly, structured, GUI-based method of defining the supply-chain information and saving it in the non-proprietary, XML-based SCML format.

3.2 The Experiment Designer (ED)

Simulation-specific information, generally referred to as experimental control information, is also needed to create an executable simulation. This information includes information regarding environment variables, such as the number of replications, animation settings, output statistics, and similar data. and is specified by running the Experiment Designer (ED). For example, one can see in Figure A2 the ED form used for specifying basic run parameters: the duration of each replication, the number of replications, and the status of the simulation traces (on/off). The collected information is saved to an XML-based experiment (EXP) file for use by the SISCO Engine.

4 TRANSLATION: MODEL GENERATION WITH THE SISCO ENGINE

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 assists the user of both simulation modeling and programming by taking the description of the supply chain and experimental conditions residing in the SCML and EXP files and generating a Silk™ simulation model based on that information. 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. The SISCO Engine is comprised of

- the SISCO Library,
- the SISCO Automated Model Mapper (SAMM),
- simulation management routines.

The SISCO Library is a set of supply-chain-oriented Java classes that represent supply chain locations, resources, actions, and logic. The Library classes are created by extending the core classes contained in the Silk™ simulation framework. The SAMM is a set of routines that perform a real-time mapping of the supply chain to the proper members of the SISCO Library. For each piece of the supply chain, the correct classes in the Library are identified and corresponding objects are created, resulting in a Silk™ simulation model of the supply chain. The simulation management routines handle the basic Java and Silk™ "housekeeping," provide the GUI for the SISCO Engine, compile the output statistics, create the various output files, and provide an overall "container" for the SISCO Engine routines.

4.1 SILK

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. This does, however, require the users to have specialized programming skills, in addition to simulation modeling skills.

For any object to function properly within the Silk™ simulation framework, that object must be of a type (class) that is derived, directly or indirectly, 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, as if each thread of execution were a separate computer. Each thread has its own life span and, most importantly, operates according to its own timeline 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.

4.2 Modeling Paradigm

Our modeling approach creates autonomous units representing each node or arc within the supply chain. These objects interact with each other, performing the basic actions an order undergoes during its processing, from inception to disposal. This object-oriented, agent-based approach is used because it provides flexibility, extensibility, and a natural mapping from reality to the simulation model.

4.2.1 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 life cycle of an order, where the actions occur in parenthesis, 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).

4.3 Modeling Fundamental Pieces of the Supply Chain

4.3.1 Modeling Nodes

The 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, and order receiving.

Order creation involves the creation and initialization of an order. Orders are 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. 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 prepares an order for transport to its target, which is 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 attempts to meet the needs or demands of the order. Fulfillment of an order is made from the finished goods inventory of the node. If finished goods inventory cannot meet the order's needs, the order waits 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 the interpretation changes 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 complete.

4.3.2 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 replenishment orders being sent to their target node for processing, or the movement of goods, such as the shipment of filled orders, back to their origin.

4.3.3 Modeling Other Supply Chain Processes

We also include other operations that define the life cycle of an order. Inventory management, both for materials and finished goods, is one such action. We include these 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 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 with processes, called Demand Generators, 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.

4.4 SISCO Library

The implementation of the modeling approach described in the above section is based on a specialized set of Java classes known as the SISCO Library. The SISCO Engine is a Java application that incorporates ThreadTecs's Silk™ as a simulation foundation. 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 Engine uses the various pieces of the SISCO Library to create a simulation model from an SCML file that has the same structure and characteristics of the supply chain described. The following sections discuss the parts, architecture, and implementation of the SISCO Library.

4.4.1 X-Classes

The first part of the SISCO Library is a set of approximately 50 Java classes, known collectively as the "x-classes," so-called because all begin with the letter "x", standing for XML focus. The x-classes are data-only classes that provide a means of representing the supply chain 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. The x-classes mimic 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. Thus, for example, an *Action* element in the SCML file will be represented by an equivalently structured Java object based on the xAction class., and a NodeCosts SCML element will be represented by an xNodeCosts object. The top-level x-classes are xNode, xArc, xComponent, xAction, and xPolicy, which represent the node, arc, component, action, and policy elements of the SCML format. The xNode and xArc classes are also used as parent classes for Node and Arc classes, which are operational classes of the SISCO system.

4.4.2 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. 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. Each order has an origin node, which is the node that creates the order, and a target node, which is the node that will fill the order and then send it back to the origin. The arc used to transport the order from origin to the target is referred to as the information arc, while the arc used to move the filled order from the target back to the origin is the shipment arc. The most important part of the *Order* class is its guidance of the order through the life cycle. An order's target node and the information arc are determined by the Order Placement Manager of the origin node, while the shipping arc is determined by the Shipping Manager of the target node. 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 Arc*s* (“owners”)*,* the Managers*,* and the Actors is fundamental to the way SISCO operates.

4.4.3 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, anode 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. 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, 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. See Figure 2 for a representation of the owner and managers of the owner-manager-actor architecture.

4.5 Model Generation with SAMM

The SISCO Automated Model Mapper (SAMM) is the part of the SISCO Engine responsible for turning the user’s description of the supply chain and experimental conditions into a working simulation model. The model generation process involves taking the user input, primarily contained in the SCML file, and generating an equivalent Silk™ simulation model by creating instances of the appropriate SISCO Library classes. The process involves parsing the SCML file to identify the various supply chain constructs, mapping these constructs to the appropriate classes in the SISCO Library; and initializing these constructs if necessary.

The SCML file stores the supply chain information in elements and attributes according to a specific structure defined by the SCML document type definition (DTD) and by XML conventions.

By knowing the elements, attributes, and the rules of their structure, we can identify the supply chain construct descriptions in the SCML file and then create an equivalent set of Java objects. To process the SCML file and identify the various structures within it, we utilize a Java-based XML parser to read the SCML document in a serial fashion from the top down. The XML parser executes "event procedures" when it encounters certain general structures within an XML file. For example, when the parser encounters the start of an element, a start Element procedure is executed, after which the parser continues reading the document. By adding custom task-specific code to the parser's event procedures, we are able to customize the parser for a specific application of XML, such as SCML. The customized parser will then perform specific actions, such as creating an object based on a specific template (instantiating a class) or copying the contents of an SCML attribute to a Java object field, based on what structures were encountered by the parser.

When the parser identifies the beginning of a basic supply chain construct (*node, arc, component, action, or policy*) it creates an object based on the corresponding SISCO Library class and begins to complete the details with the information that follows in the SCML file. When the parser identifies the end of a construct's description, the object is placed in an array with similar objects for future access, it is initialized if necessary, and the parser continues on to the next supply chain construct in the SCML file.

The initialization process varies, based on the type of object. Non-operational objects, such as components, actions, and policies, have a simple initialization that includes creation of arrays or other organizational aids to make access to the information they contain easier. On the other hand, the initialization of nodes and arcs is extensive. The initialization of a node or arc creates the appropriate Manager and Demand Generator objects, as well as creating all the necessary Silk™ resources, queues, state variables, and statistical tracking variables. For a node, the Silk™ Queues and Resources for supply chain actions (order placement, order processing, shipping, and receiving) are created. Silk™ State Variables are generated for each input and output of the node, creating materials and finished goods inventory tracking variables. The associated basic action Manager objects (order placement, order processing, shipping and receiving Managers) are created, as are Inventory Manager objects for each node input and output. If the node is a customer node, a Demand Generator object for each component demanded is created. Additionally, a full complement of statistical tracking variables are created. The Manager objects will then go through an initialization process of their own, creating the appropriate variables, arrays, and related constructs. The process is the same for creating an Arc, but simpler because its only action is transporting orders.

Thus, the initialization of a node or an arc object begins an hierarchical initialization process, which may be several level deep, that results in the creation of all necessary Silk™ components, Java objects, and their associated routines, to simulate the operation of that node or arc. The parsing procedure contained in SAMM 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.

When the SCML file is finished being processed the entire supply chain it describes is now represented as a set of Java and Silk™ simulation objects that are ready to execute by starting the simulation run.

5 MODEL EXECUTION AND OUTPUT

5.1 Executing the Model

The SISCO Engine is executed as a Java application, which can be seen in Figure A3. The user specifies the SCML and EXP files that contain the model and experimental information, as well the names for the summary, replication, and time-series data files. After specifying the files to use, SAMM parses the files and generates a Silk simulation model. As soon as the Model Mapper is finished generating the set of Silk objects that will represent the supply chain, the model is ready to execute. Essentially, the model is ready to execute since all objects are created, started, and initialized. The model sits in a state of suspended animation because the simulation clock is not advancing, but held at time 0.0 until the user starts the simulation run. The user starts the simulation by clicking on the "run" button in the control console dialog that pops up when model generation is finished, which can be seen in Figure A4.

5.2 Model Output

The results of a SISCO simulation run are contained in three files: a summary text file, a replication data file, and a time-series data file. The summary file contains performance characteristics over all replications for each node, arc, and the system. In addition to total supply chain costs, for each node we are presented with characteristics of the following information:

- time orders spent in each of the "action" queues,
- length of the queues,
- time spent performing each action,
- utilization of the action resources,
- costs incurred for each of the actions,
- inventory and shortage levels,
- inventory and shortage costs,
- inter-order times,
- order lead times,
- order sizes and variances,
- lead-time demand,
- demand and lead time forecasts,
- service levels

and other related information.

The following information is presented for each arc in the supply chain:

- the time that orders spend in the transport queue,
- the length of the queue,
- the time spent in transit,
- the utilization of the transportation resources,

and other related information.

The replication data file contains the mean performance value for each node (or arc) over all replications for a set of chosen metrics. Thus, if three metrics are chosen and we simulate a five- node supply chain for 30 replications, then the replication file will contain thirty values for each metric for each of the five nodes (or arcs). The time series data file contains a complete set of values for one replication of the simulation. The time series file is used to capture information for further analysis, and may include order amounts, inventory levels, forecasts and other characteristics.

6 IMPLEMENTATION AND APPLICATION

6.1 Development Technologies

The implementation of SISCO involves merging three relatively new, complementary technologies: Java, ThreadTec's Silk™, and XML processing (called "parsing").

6.1.1 Java

Java's object-orientation, platform independence, and Internet "friendliness" were major factors in choosing it as the SISCO Engine’s development language. Java’s object-orientation fits with our modeling goal of creating simulation models consisting of autonomous supply chain participants. Of additional import is that Java currently has the best XML processing capability and support among general-purpose programming languages. Thus, the Java language is a natural fit for the development of the SISCO Engine.

6.1.2 SILK

Silk™, by ThreadTec Inc., is a set of Java classes that provide discrete-event simulation features needed for simulation modeling, including basic implementation 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 [4]. Silk™ allows the user to develop complex, customized simulation models, while still allowing access to the full capabilities of the Java language. Since we utilize a large amount of custom programming, both as an external “controller” that actually builds the simulation model, as well as within the simulation models themselves, the capabilities and flexibility of the Silk™ framework are a great fit for the development needs of SISCO. Additionally, the multi-threaded, object-oriented nature of Silk™ simulation models fully coincides with the aim to have SISCO generate object-oriented simulation models where supply chain participants are represented by relatively autonomous units.

6.1.3 XML Parsing

XML forms the basis of the SCML supply chain description language so, for SISCO to utilize SCML files, parsing (both read and write) capabilities are necessary. At the time of SISCO's development some XML-related technologies were not yet fully specified.

In order to operate as a cohesive system, each of SISCO's three modules, the VSCE, the ED, and the SISCO Engine, must be able to utilize XML-based file formats, namely SCML files containing the "model" information and the EXP files containing the "experimental" information. The ability to read and write XML-based file formats is developed by customizing the procedures of a general XML parser so that it recognizes the structures of a specific XML-based file format, also called a document type. In our case we have two document types, SCML and EXP, each of which require separate custom parsing procedures. The general XML parsers may implement the SAX (Simple API for XML) API (application programming interface) for parsing XML, the DOM (Document Object Model) API for parsing XML, or both APIs.

SAX parsers are faster and less memory-intensive than DOM parsers and provide a simple, efficient manner for processing an XML file in a sequential, one-pass fashion. The SAX parser reads through the XML files sequentially and "fires" events when certain structures, such as the start of an element, are identified. Event procedures contain code, executed whenever the event occurs, that performs the desired information processing as the structures of interest are identified. The event procedure code is customizable.

In contrast, the DOM is a tree-based API that maps an XML document into a set of objects in a tree-like structure determined from the document's DTD [5]. The DOM API allows the programmer to read objects (XML nodes on the tree), as well as add, subtract, or modify XML nodes to the tree. The tree-structure (the DOM) can then be navigated to find, extract, and change information, if desired. 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.

We chose to utilize the SAX API for file reading routines in all SISCO modules. In the SISCO Engine, we must read through an SCML file, identify structures of interest, perform tasks (executing appropriate code), and then move on to the next piece of the supply chain. This is a natural fit for a SAX parser. Likewise, the task of reading the information contained in an SCML or EXP file back into the Visual Basic data structures of the VSCE or ED is a natural fit for a SAX parser. The DOM API allows one to build an XML file from scratch, which is what we wish to do when saving the contents of the VB data structures in the VSCE to an SCML file, or the ED's data structures to an EXP file. Therefore we chose to utilize the DOM API for the file writing routines included in the "Save" function of the VSCE and ED.

6.2 SISCO Engine

The SISCO Engine needs to be able to read supply chain structure and logic information from SCML files and experimental information from EXP files. Additionally, in the case of the SCML file, procedures must be executed on-the-fly, based on what is identified in the supply chain description file, in order to generate the appropriate set of simulation objects.

6.2.1 Parsing SCML Files

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.

Of the two document types the SISCO Engine must read and process, the processing of SCML files is significantly more complex and is described in the following sections. In essence, the parser transfers, or maps, the structure and content of the SCML file to an equivalent structure comprised of Java objects. These Java objects are based on the x-classes of the SISCO Library, which mimic the hierarchy and information content of SCML elements and attributes, as defined in the SCML DTD. Some objects based on the Operational classes of the SISCO Library, such as Nodes, Arcs, and InventoryManagers, are also created by the parsing process or the initialization of objects created by the parsing process. These operational objects control or perform the actual simulation activities of the model, as opposed to the essentially information storage duties of objects based on the x-classes.

The most important SAX parser event procedures used by the SISCO Automated Model Mapper, the part of the SISCO Engine that processes SCML files, are the start_Element and end_Element events. The start_Element procedure is executed whenever the parser encounters the start "tag" of an XML element and contains the majority of the custom code that defines the SCML parser. The start_Element procedure identifies the element the parser has encountered (such as *node* or *actionOutput*) and creates a corresponding object, from the appropriate SISCO Library class, placing it in the proper location within the object hierarchy. The element's information, generally contained in XML attributes and sub-element, is parsed and transferred to the appropriate fields of the new Java object, keeping the hierarchy of information intact. The end_Element event procedure fires when the end "tag" of an XML element is found and identifies the element whose description is "ending," using that information to ensure that the Java data structures faithfully reflect the hierarchical structure of the SCML file. If the element ending is identified as a basic construct (*node, arc, component, action, or policy*), the object used to represent that construct is placed in an array with objects of the same type and the object is initialized.

For example, if an *actionOutput* Element is "found" by the parser an xActionOutput object is created (based on the xActionOutput class in the SISCO Library), the appropriate information is transferred to its fields, and the object assigned to appropriate field of the xAction object representing the *action* element currently being parsed, and then initialized, if necessary. The parser will continue extracting information about the current *action* element until the end "tag" for that element is encountered and the end_Element procedure executed. The xAction object will then be placed in an array with other actions, initialized, and the parsing of the SCML document will continue.

Initialization may include the creation of Managers, Resources, Queues, arrays, state variables, statistical tracking variables, and related structures. Some of the created structures may have their own initialization procedures. The result for more complex supply chain constructs, such as a *node*, is that initializing the object begins an hierarchical initialization process that includes the creation and initialization of subordinate structures, which may occur several levels deep.

Thus, when the parser finds the end of a basic construct's description, the Java object that will represent that construct in the simulation model is initialized and put in an array with like objects. The parser then moves on to the next element in the SCML file and the process continues until the entire SCML file is processed. When the SCML file is finished being processed the entire supply chain it describes is now represented as a set of Java and Silk simulation objects that are ready to execute by starting the simulation run.

6.2.2 Parsing EXP Files

The procedure the SISCO Engine uses for parsing EXP files is fundamentally similar, but significantly less complex than the one for SCML files. The customized SAX parser identifies the various elements of the EXP file as is reads through the file and extracts simple pieces of information, such as the number of replications to run from XML attributes. The information is then assigned to the appropriate Java variables in the main simulation management procedure of SISCO. The files are short, the processing is simple and direct, and there are no objects created or initialization procedures to coordinate.

6.2.3 Development Platform

The SISCO Engine was built with the Java programming language, utilizing the “Swing” Library for user-interface programming. All Java development was done using Symantec's Visual Cafe integrated development environment. The Silk™ simulation framework from ThreadTec Incorporated [4] is used to provide core simulation capabilities. To enable the SISCO Engine to parse SCML and EXP files, we utilize Sun Microsystem’s XML parser, which provides a Java implementation of both SAX and DOM-based parsers.

6.3 VSCE and ED

The VSCE and ED applications are the “user interaction” modules of the SISCO system. They are GUI driven and, through an organized set of menus, forms, and dialog boxes, provide a simplified manner for the user to specify relatively complex information. In both applications the information the user provides is temporarily stored in a set of Visual Basic data structures that mimic the structure (DTD) of the file format being used (SCML or EXP).

When a user chooses to save their work (in either application) the application converts the information the user has specified, and stored in the data structures, into an equivalent SCML or EXP file. A customized XML parsing routine, utilizing the DOM API, handles the task of creating the EXP or SCML file, creating the appropriate XML elements and attributes, transferring the information from the VB data structures to the XML elements and attributes and ensuring that the structure of the newly save file corresponds to the SCML or EXP document type definition (DTD) as well as general XML formatting conventions.

When a user chooses to read information from a previously created file back into the VSCE or ED for further editing the a different parsing routine is used. A customized parsing routine, utilizing the SAX API, reads the SCML or EXP file and creates Visual Basic data structures on-the-fly as various pieces of the file are identified and processed. Information stored in XML elements and attributes is transferred over to the corresponding fields in the VB data structures. When parsing is finished, the display (diagram, forms, and dialog boxes) is updated to reflect the newly opened supply chain information.

The VSCE and ED were both developed using Microsoft Visual Basic 6, Professional Edition. 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. Within the Visual Basic environment we employ Microsoft's MSXML parser to provide DOM parsing capabilities, while relying on VividCreation's ActiveSAX to provide SAX parsing capabilities.

6.4 Applications

SISCO was developed because simulation modeling can provide valuable insights into the operational characteristics of supply chains. Variability is a reality in all systems, but especially so in supply chain systems. 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 following are some issues in supply chain decision-making that would benefit from rigorous simulation 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 efforts have taken exchange rates and other global conditions into account, though these can be significant drivers of profitability.
- service-oriented supply chains have been only sparingly modeled.

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 (BWE). 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 BWE [2]. SISCO was invaluable in the creation, execution, and coordination of the simulations.

7 CONCLUSIONS

Variability is a fact of life in supply chains, but is often overlooked or assumed away when supply chain models are built for decision support. The Simulator for Integrated Supply Chain Operations (SISCO) provides a friendly but powerful approach to perform simulation analysis of supply chains. 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. As a fully object-oriented supply chain simulator, SISCO represents supply chain participants as autonomous units that communicate and interact, much like they do in reality. SISCO addresses the supply chain information standardization problem by using the XML-based SCML file format for supply chain model storage. Beyond 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 thank the Center for Supply Chain Research at Penn State University for partial funding of this project. We also thank ThreadTec for permission to use Silk™.

REFERENCES

[1] 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.

[2] Chatfield, D., J. Kim, T. Harrison, and J. Hayya. 2002. Order Flow in Serial Supply Chains. Working Paper, Department of Supply Chain and Information Systems, Penn State University, University Park, PA.

[3] Chatfield, D., T. Harrison, and J. Hayya. 2003. The Supply Chain Modeling Language (SCML). Working Paper, Department of Supply Chain and Information Systems, Penn State University, University Park, PA.

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

[5] Mosenhi, P. 1999. An introduction to XML for Java programmers. *Java Pro*, v3, n3, pp 48-52. Fawcette Technical Publications, Palo Alto, CA.

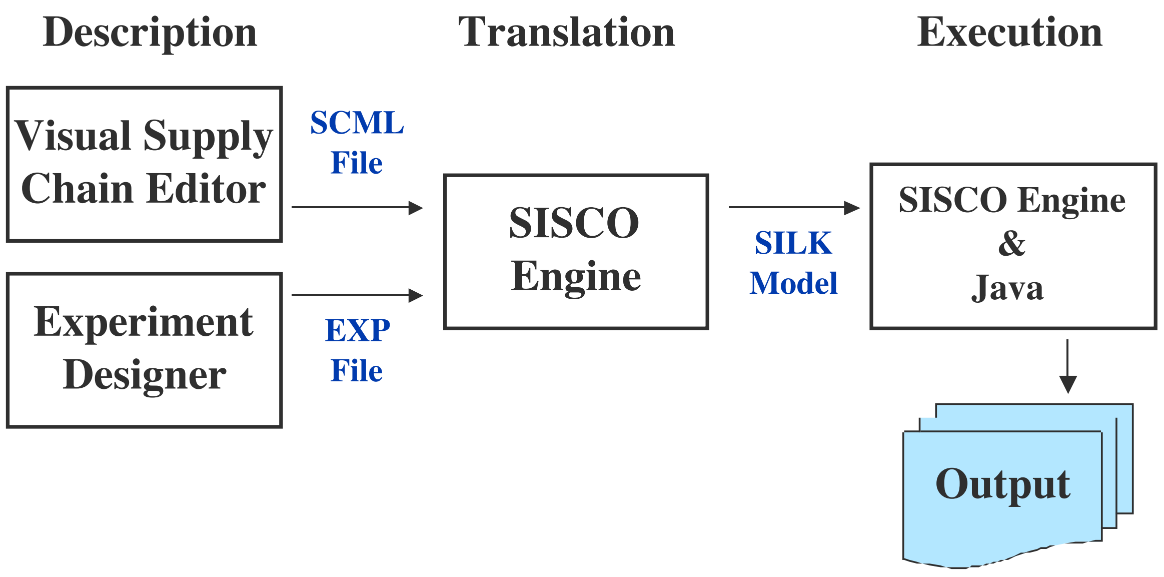


Figure 1. SISCO System Overview

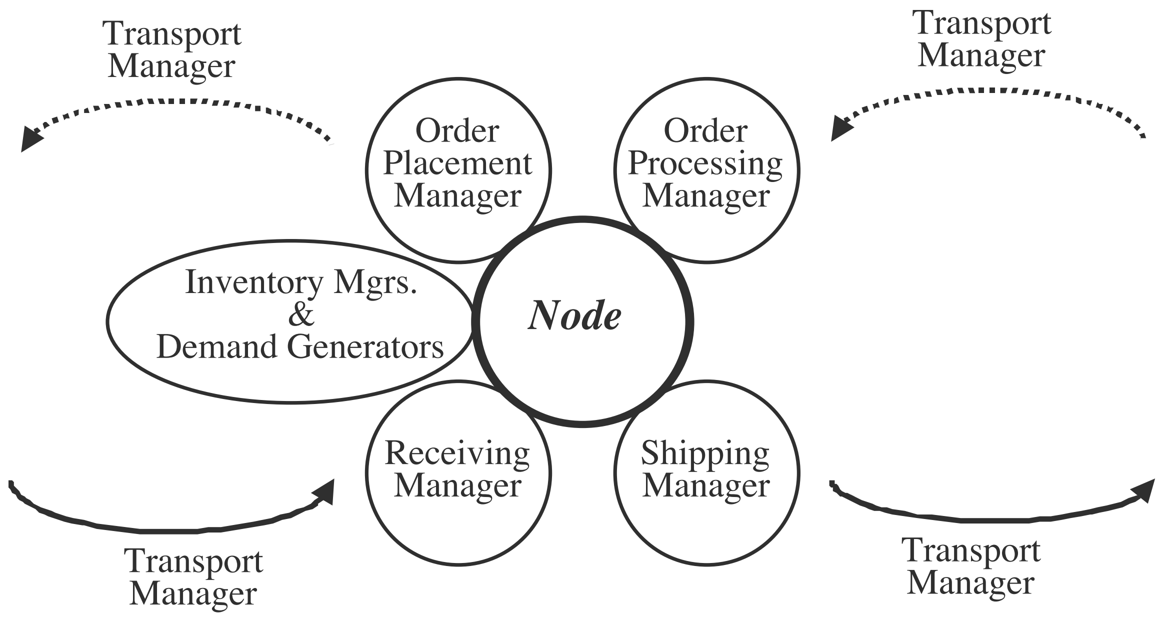


Figure 2. Owner-Manager-Actor Modeling Architecture

Appendix A – Selected ScreenShots

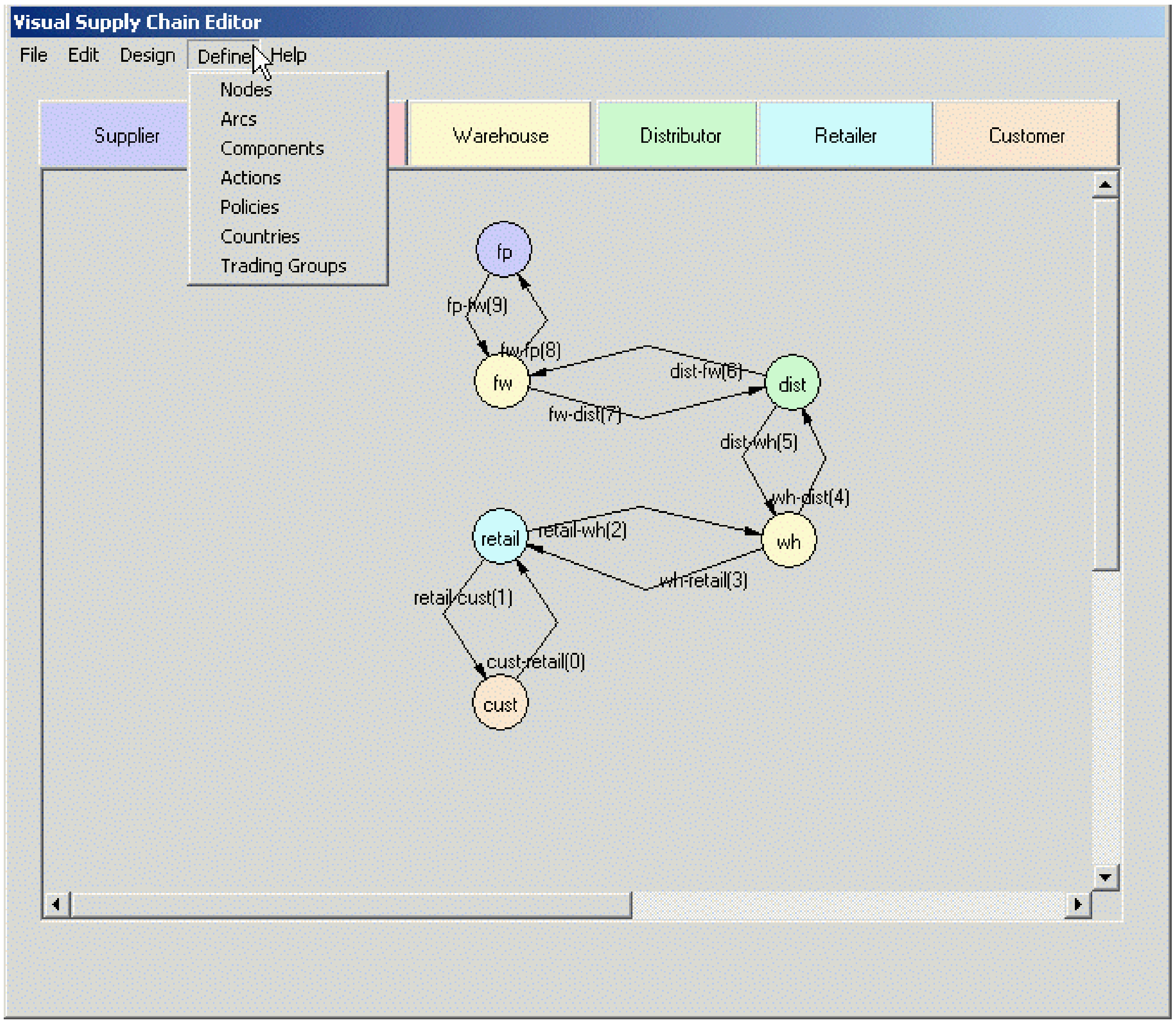
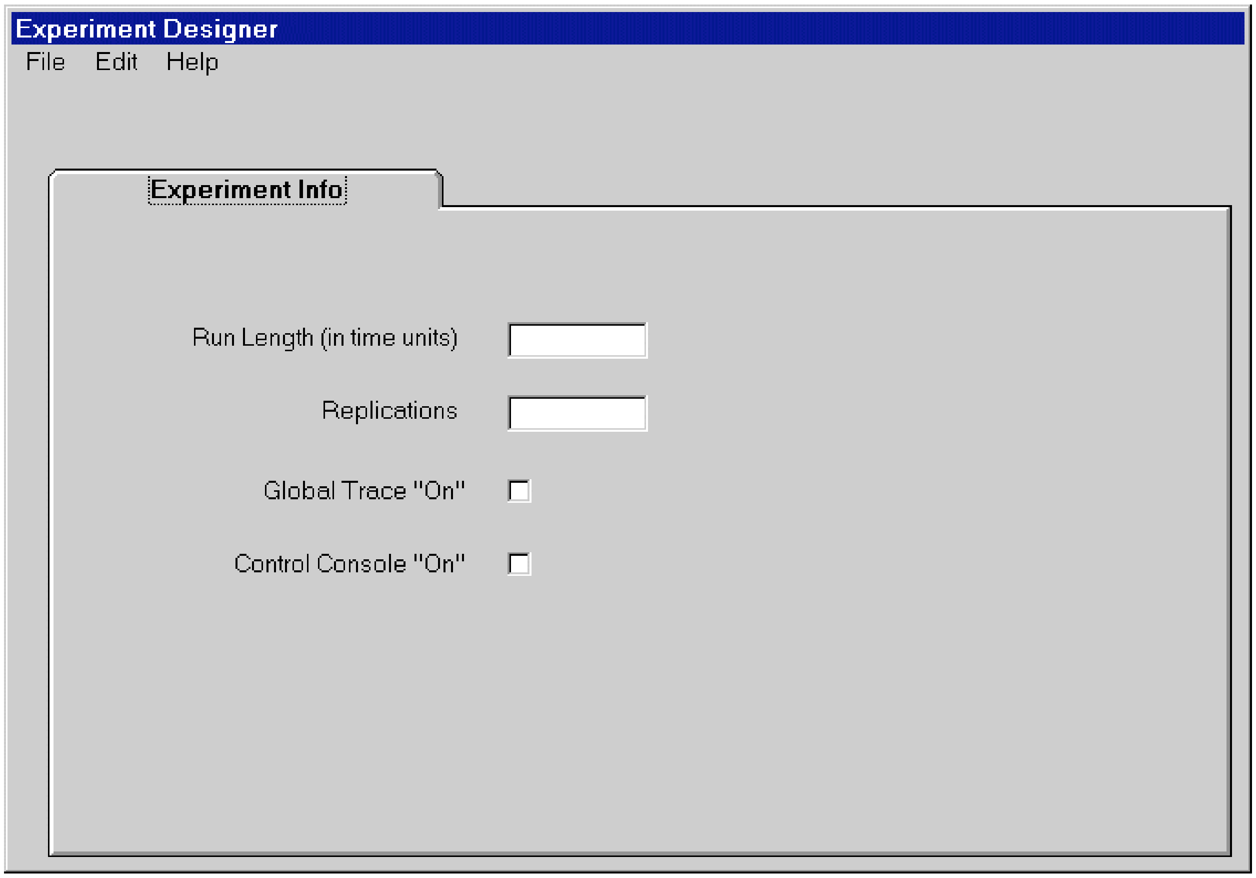


Figure A1: Visual Supply Chain Editor
Shown: "Design" form used to specify supply chain Nodes and Arcs
(Note menu selection showing construct categories to be further defined)



Run Length (in time units)

Replications

Global Trace "On" ☐

Control Console "On" ☐

Figure A2: Experiment Designer
Shown: Form for specifying basic run parameters

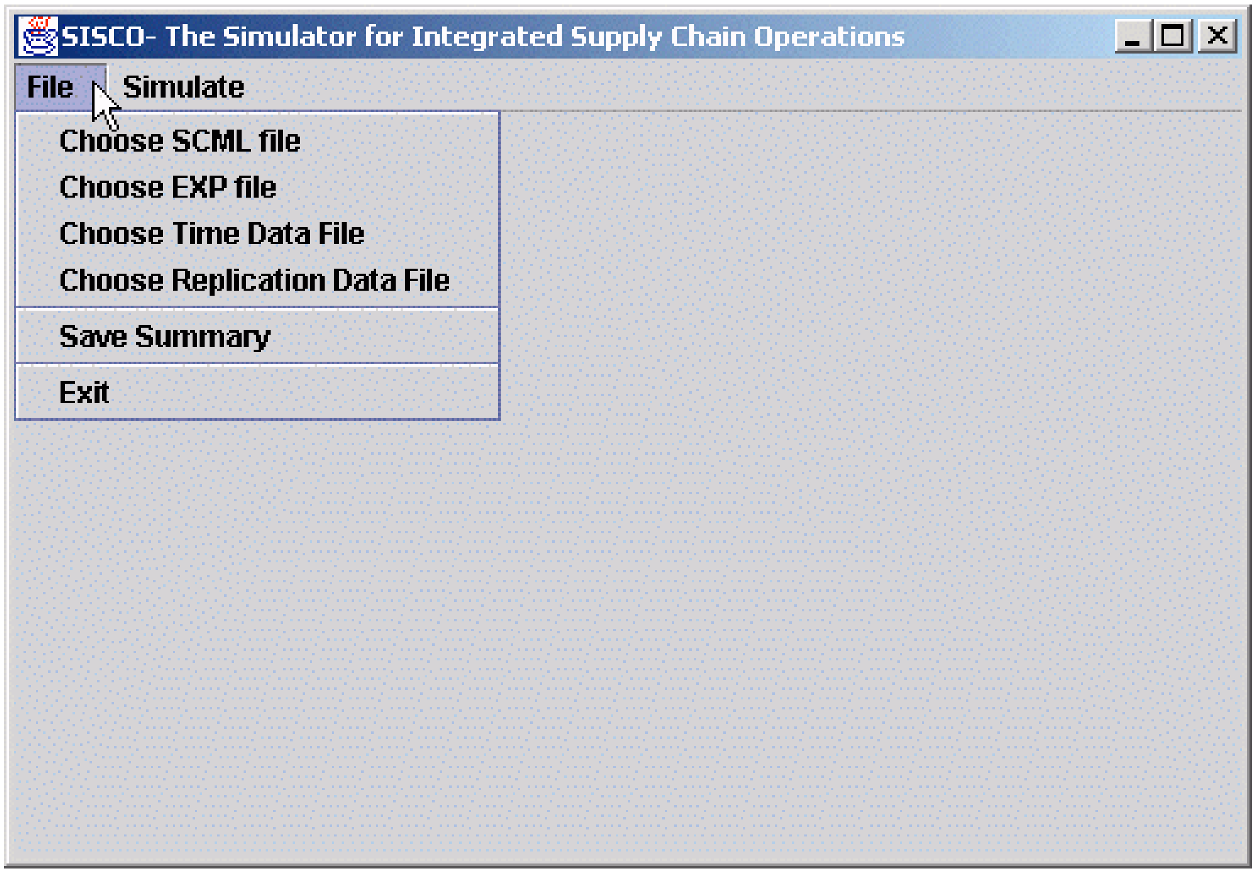

SISCO- The Simulator for Integrated Supply Chain Operations
File
Simulate
Choose SCML file
Choose EXP file
Choose Time Data File
Choose Replication Data File
Save Summary
Exit

Figure A3: SISCO
Shown: Menu for selecting input and output files

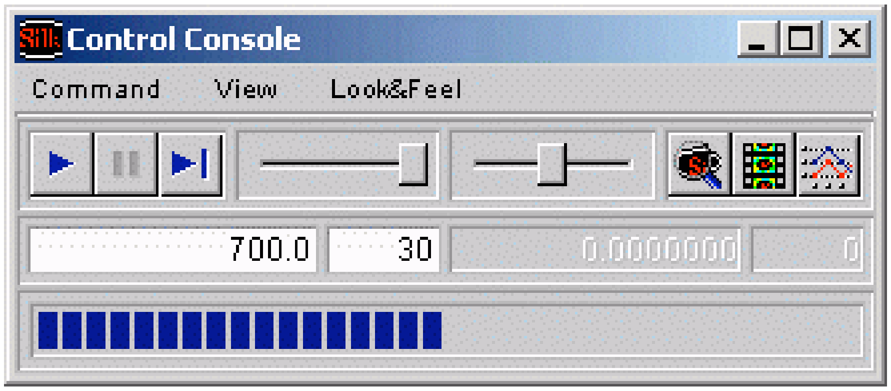
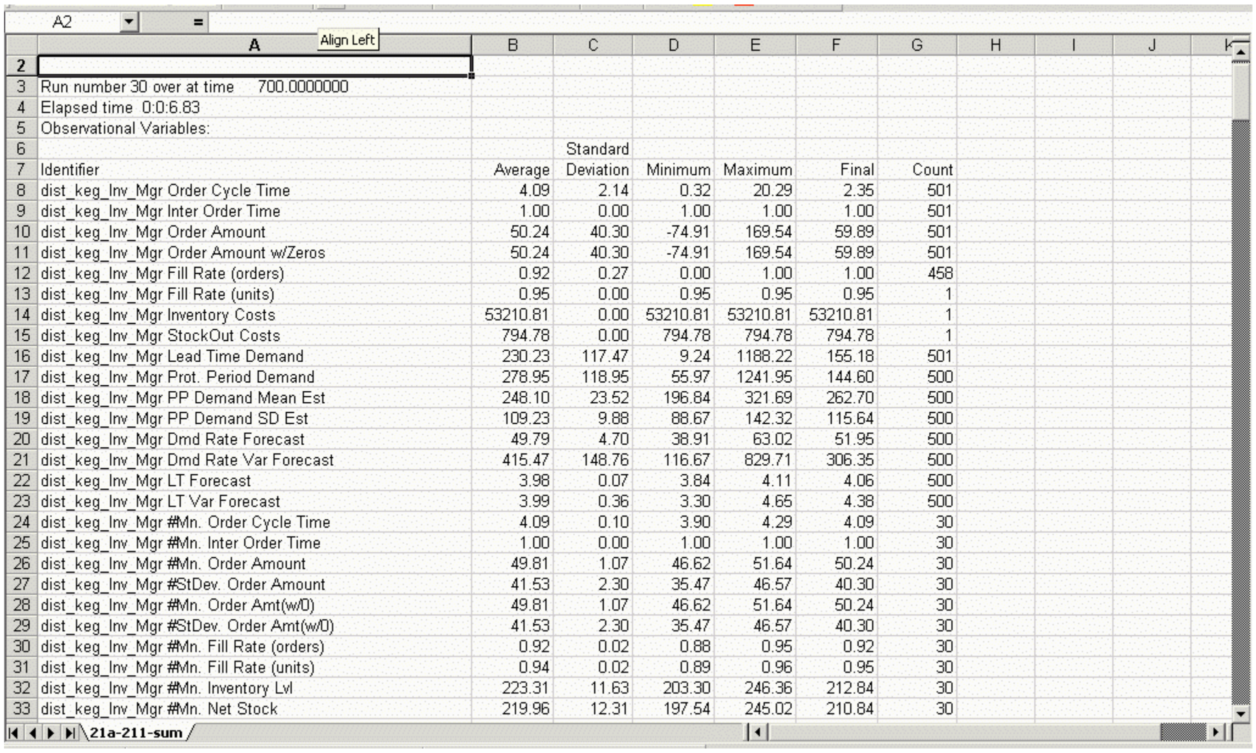


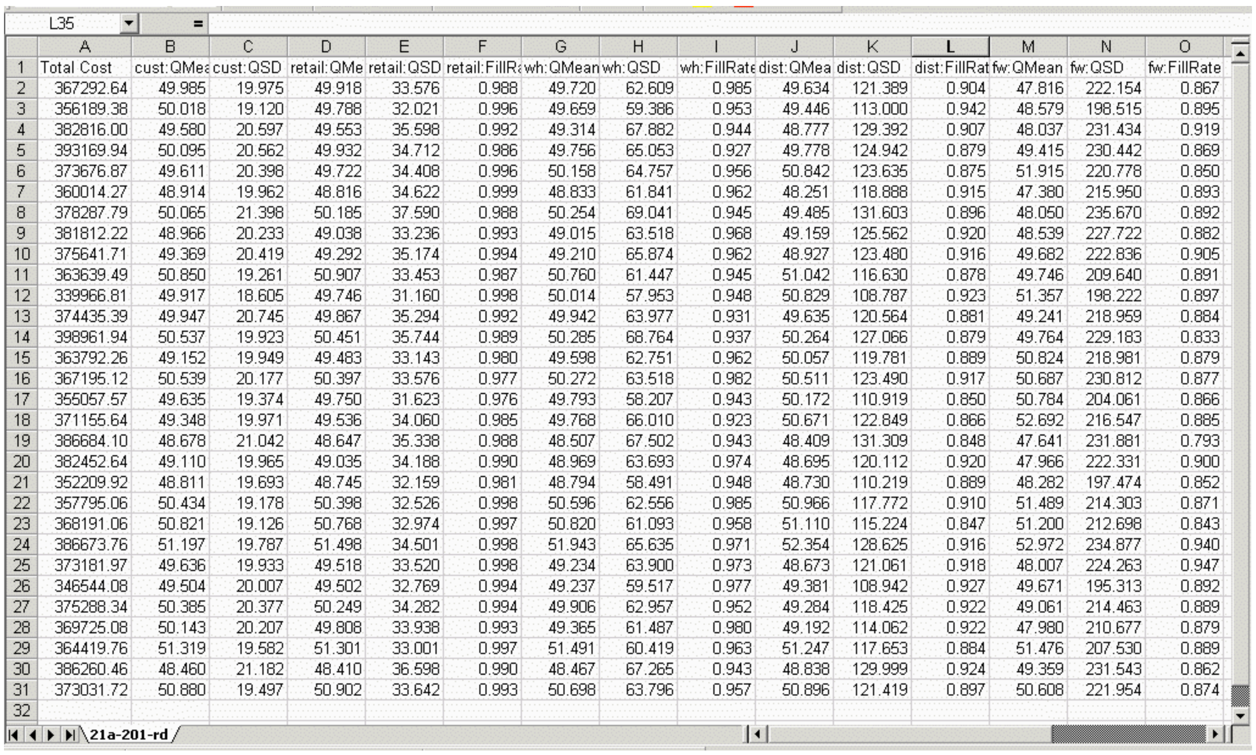
Figure A4: SILK Control Console
Used to begin simulation execution



Run number 30 over at time 700.0000000
Elapsed time 0:0:6.83
Observational Variables:

Identifier	Average	Standard Deviation	Minimum	Maximum	Final	Count
dist_keg_Inv_Mgr Order Cycle Time	4.09	2.14	0.32	20.29	2.35	501
dist_keg_Inv_Mgr Inter Order Time	1.00	0.00	1.00	1.00	1.00	501
dist_keg_Inv_Mgr Order Amount	50.24	40.30	-74.91	169.54	59.89	501
dist_keg_Inv_Mgr Order Amount w/Zeros	50.24	40.30	-74.91	169.54	59.89	501
dist_keg_Inv_Mgr Fill Rate (orders)	0.92	0.27	0.00	1.00	1.00	458
dist_keg_Inv_Mgr Fill Rate (units)	0.95	0.00	0.95	0.95	0.95	1
dist_keg_Inv_Mgr Inventory Costs	53210.81	0.00	53210.81	53210.81	53210.81	1
dist_keg_Inv_Mgr StockOut Costs	794.78	0.00	794.78	794.78	794.78	1
dist_keg_Inv_Mgr Lead Time Demand	230.23	117.47	9.24	1188.22	155.18	501
dist_keg_Inv_Mgr Prot. Period Demand	278.95	118.95	55.97	1241.95	144.60	500
dist_keg_Inv_Mgr PP Demand Mean Est	248.10	23.52	196.84	321.69	262.70	500
dist_keg_Inv_Mgr PP Demand SD Est	109.23	9.88	88.67	142.32	115.64	500
dist_keg_Inv_Mgr Dmd Rate Forecast	49.79	4.70	38.91	63.02	51.95	500
dist_keg_Inv_Mgr Dmd Rate Var Forecast	415.47	148.76	116.67	829.71	306.35	500
dist_keg_Inv_Mgr LT Forecast	3.98	0.07	3.84	4.11	4.06	500
dist_keg_Inv_Mgr LT Var Forecast	3.99	0.36	3.30	4.65	4.38	500
dist_keg_Inv_Mgr #Mn. Order Cycle Time	4.09	0.10	3.90	4.29	4.09	30
dist_keg_Inv_Mgr #Mn. Inter Order Time	1.00	0.00	1.00	1.00	1.00	30
dist_keg_Inv_Mgr #Mn. Order Amount	49.81	1.07	46.62	51.64	50.24	30
dist_keg_Inv_Mgr #StDev. Order Amount	41.53	2.30	35.47	46.57	40.30	30
dist_keg_Inv_Mgr #Mn. Order Amt(w/0)	49.81	1.07	46.62	51.64	50.24	30
dist_keg_Inv_Mgr #StDev. Order Amt(w/0)	41.53	2.30	35.47	46.57	40.30	30
dist_keg_Inv_Mgr #Mn. Fill Rate (orders)	0.92	0.02	0.88	0.95	0.92	30
dist_keg_Inv_Mgr #Mn. Fill Rate (units)	0.94	0.02	0.89	0.96	0.95	30
dist_keg_Inv_Mgr #Mn. Inventory Lvl	223.31	11.63	203.30	246.36	212.84	30
dist_keg_Inv_Mgr #Mn. Net Stock	219.96	12.31	197.54	245.02	210.84	30

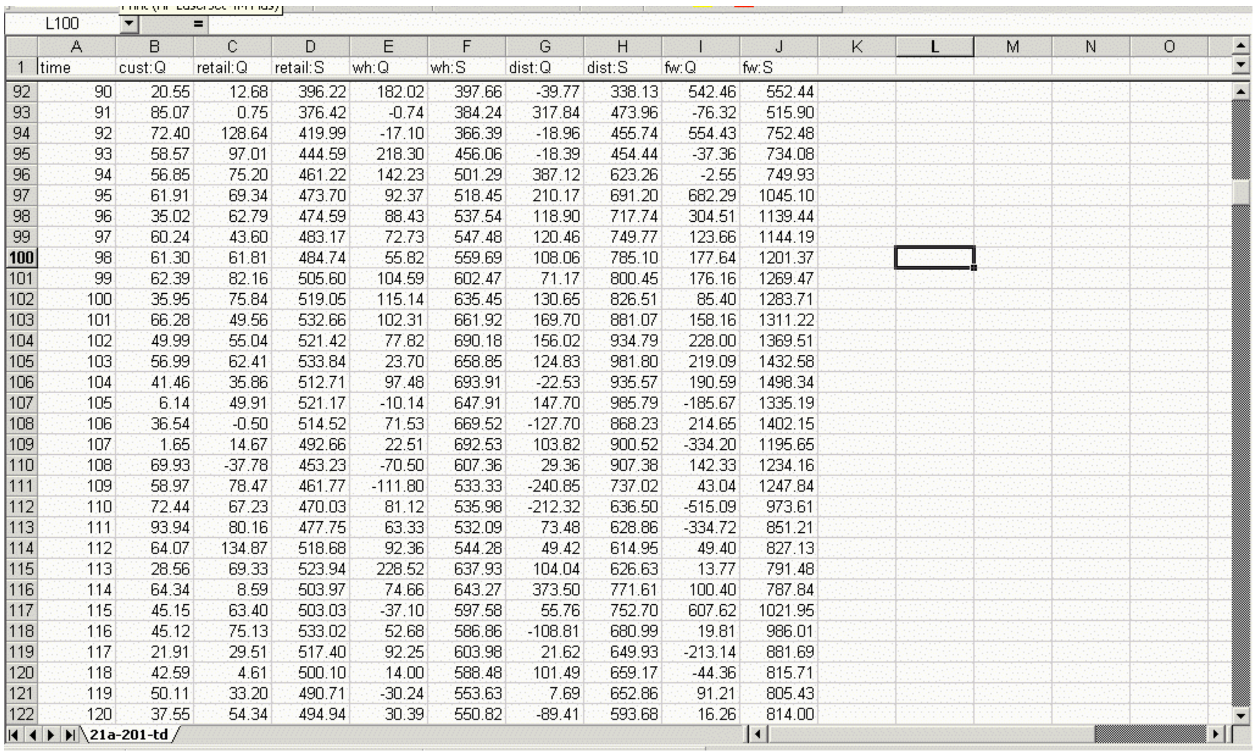
Figure A5: Summary Output File (displayed in Excel)



L35 =

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Total Cost	cust:QMea	cust:QSD	retail:QMe	retail:QSD	retail:FillRa	wh:QMean	wh:QSD	wh:FillRate	dist:QMea	dist:QSD	dist:FillRat	fw:QMean	fw:QSD	fw:FillRate
2	367292.64	49.985	19.975	49.918	33.576	0.988	49.720	62.609	0.985	49.634	121.389	0.904	47.816	222.154	0.867
3	356189.38	50.018	19.120	49.788	32.021	0.996	49.659	59.386	0.953	49.446	113.000	0.942	48.579	198.515	0.895
4	382816.00	49.580	20.597	49.553	35.598	0.992	49.314	67.882	0.944	48.777	129.392	0.907	48.037	231.434	0.919
5	393169.94	50.095	20.562	49.932	34.712	0.986	49.756	65.053	0.927	49.778	124.942	0.879	49.415	230.442	0.869
6	373676.87	49.611	20.398	49.722	34.408	0.996	50.158	64.757	0.956	50.842	123.635	0.875	51.915	220.778	0.850
7	360014.27	48.914	19.962	48.816	34.622	0.999	48.833	61.841	0.962	48.251	118.888	0.915	47.380	215.950	0.893
8	378287.79	50.065	21.398	50.185	37.590	0.988	50.254	69.041	0.945	49.485	131.603	0.896	48.050	235.670	0.892
9	381812.22	48.966	20.233	49.038	33.236	0.993	49.015	63.518	0.968	49.159	125.562	0.920	48.539	227.722	0.882
10	375641.71	49.369	20.419	49.292	35.174	0.994	49.210	65.874	0.962	48.927	123.480	0.916	49.682	222.836	0.905
11	363639.49	50.850	19.261	50.907	33.453	0.987	50.760	61.447	0.945	51.042	116.630	0.878	49.746	209.640	0.891
12	339966.81	49.917	18.605	49.746	31.160	0.998	50.014	57.953	0.948	50.829	108.787	0.923	51.357	198.222	0.897
13	374435.39	49.947	20.745	49.867	35.294	0.992	49.942	63.977	0.931	49.635	120.564	0.881	49.241	218.959	0.884
14	398961.94	50.537	19.923	50.451	35.744	0.989	50.285	68.764	0.937	50.264	127.066	0.879	49.764	229.183	0.833
15	363792.26	49.152	19.949	49.483	33.143	0.980	49.598	62.751	0.962	50.057	119.781	0.889	50.824	218.981	0.879
16	367195.12	50.539	20.177	50.397	33.576	0.977	50.272	63.518	0.982	50.511	123.490	0.917	50.687	230.812	0.877
17	355057.57	49.635	19.374	49.750	31.623	0.976	49.793	58.207	0.943	50.172	110.919	0.850	50.784	204.061	0.866
18	371155.64	49.348	19.971	49.536	34.060	0.985	49.768	66.010	0.923	50.671	122.849	0.866	52.692	216.547	0.885
19	386684.10	48.678	21.042	48.647	35.338	0.988	48.507	67.502	0.943	48.409	131.309	0.848	47.641	231.881	0.793
20	382452.64	49.110	19.965	49.035	34.188	0.990	48.969	63.693	0.974	48.695	120.112	0.920	47.966	222.331	0.900
21	352209.92	48.811	19.693	48.745	32.159	0.981	48.794	58.491	0.948	48.730	110.219	0.889	48.282	197.474	0.852
22	357795.06	50.434	19.178	50.398	32.526	0.998	50.596	62.556	0.985	50.966	117.772	0.910	51.489	214.303	0.871
23	368191.06	50.821	19.126	50.768	32.974	0.997	50.820	61.093	0.958	51.110	115.224	0.847	51.200	212.698	0.843
24	386673.76	51.197	19.787	51.498	34.501	0.998	51.943	65.635	0.971	52.354	128.625	0.916	52.972	234.877	0.940
25	373181.97	49.636	19.933	49.518	33.520	0.998	49.234	63.900	0.973	48.673	121.061	0.918	48.007	224.263	0.947
26	346544.08	49.504	20.007	49.502	32.769	0.994	49.237	59.517	0.977	49.381	108.942	0.927	49.671	195.313	0.892
27	375288.34	50.385	20.377	50.249	34.282	0.994	49.906	62.957	0.952	49.284	118.425	0.922	49.061	214.463	0.889
28	369725.08	50.143	20.207	49.808	33.938	0.993	49.365	61.487	0.980	49.192	114.062	0.922	47.980	210.677	0.879
29	364419.76	51.319	19.582	51.301	33.001	0.997	51.491	60.419	0.963	51.247	117.653	0.884	51.476	207.530	0.889
30	386260.46	48.460	21.182	48.410	36.598	0.990	48.467	67.265	0.943	48.838	129.999	0.924	49.359	231.543	0.862
31	373031.72	50.880	19.497	50.902	33.642	0.993	50.698	63.796	0.957	50.896	121.419	0.897	50.608	221.954	0.874
32															

Figure A6: Replication Output file (displayed in Excel)
Shown: Results for a 30 replication run



L100

	A	B	C	D	E	F	G	H	I	J
1	time	cust:Q	retail:Q	retail:S	wh:Q	wh:S	dist:Q	dist:S	fw:Q	fw:S
92	90	20.55	12.68	396.22	182.02	397.66	-39.77	338.13	542.46	552.44
93	91	85.07	0.75	376.42	-0.74	384.24	317.84	473.96	-76.32	515.90
94	92	72.40	128.64	419.99	-17.10	366.39	-18.96	455.74	554.43	752.48
95	93	58.57	97.01	444.59	218.30	456.06	-18.39	454.44	-37.36	734.08
96	94	56.85	75.20	461.22	142.23	501.29	387.12	623.26	-2.55	749.93
97	95	61.91	69.34	473.70	92.37	518.45	210.17	691.20	682.29	1045.10
98	96	35.02	62.79	474.59	88.43	537.54	118.90	717.74	304.51	1139.44
99	97	60.24	43.60	483.17	72.73	547.48	120.46	749.77	123.66	1144.19
100	98	61.30	61.81	484.74	55.82	559.69	108.06	785.10	177.64	1201.37
101	99	62.39	82.16	505.60	104.59	602.47	71.17	800.45	176.16	1269.47
102	100	35.95	75.84	519.05	115.14	635.45	130.65	826.51	85.40	1283.71
103	101	66.28	49.56	532.66	102.31	661.92	169.70	881.07	158.16	1311.22
104	102	49.99	55.04	521.42	77.82	690.18	156.02	934.79	228.00	1369.51
105	103	56.99	62.41	533.84	23.70	658.85	124.83	981.80	219.09	1432.58
106	104	41.46	35.86	512.71	97.48	693.91	-22.53	935.57	190.59	1498.34
107	105	6.14	49.91	521.17	-10.14	647.91	147.70	985.79	-185.67	1335.19
108	106	36.54	-0.50	514.52	71.53	669.52	-127.70	868.23	214.65	1402.15
109	107	1.65	14.67	492.66	22.51	692.53	103.82	900.52	-334.20	1195.65
110	108	69.93	-37.78	453.23	-70.50	607.36	29.36	907.38	142.33	1234.16
111	109	58.97	78.47	461.77	-111.80	533.33	-240.85	737.02	43.04	1247.84
112	110	72.44	67.23	470.03	81.12	535.98	-212.32	636.50	-515.09	973.61
113	111	93.94	80.16	477.75	63.33	532.09	73.48	628.86	-334.72	851.21
114	112	64.07	134.87	518.68	92.36	544.28	49.42	614.95	49.40	827.13
115	113	28.56	69.33	523.94	228.52	637.93	104.04	626.63	13.77	791.48
116	114	64.34	8.59	503.97	74.66	643.27	373.50	771.61	100.40	787.84
117	115	45.15	63.40	503.03	-37.10	597.58	55.76	752.70	607.62	1021.95
118	116	45.12	75.13	533.02	52.68	586.86	-108.81	680.99	19.81	986.01
119	117	21.91	29.51	517.40	92.25	603.98	21.62	649.93	-213.14	881.69
120	118	42.59	4.61	500.10	14.00	588.48	101.49	659.17	-44.36	815.71
121	119	50.11	33.20	490.71	-30.24	553.63	7.69	652.86	91.21	805.43
122	120	37.55	54.34	494.94	30.39	550.82	-89.41	593.68	16.26	814.00

21a-201-td

Figure A7: Time Series Output File (displayed in Excel)
Shown: Time Periods 90 – 120 from a 700 period time series file