Scalable Knowledge Interchange Broker: Design and Implementation for Semiconductor Supply Chain Systems

by

James Melkon Smith

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2012 by the
Graduate Supervisory Committee:

Hessam Sarjoughian, Chair
Hasan Davulcu
Georgios Fainekos

ARIZONA STATE UNIVERSITY

December 2012

ABSTRACT

A semiconductor supply chain modeling and simulation platform using Linear Program (LP) optimization and parallel Discrete Event System Specification (DEVS) process models has been developed in a joint effort by ASU and Intel Corporation. A Knowledge Interchange Broker ($KIB_{DEVS/LP}$) was developed to broker information synchronously between the DEVS and LP models. Recently a single-echelon heuristic Inventory Strategy Module (ISM) was added to correct for forecast bias in customer demand data using different smoothing techniques. The optimization model could then use information provided by the forecast model to make better decisions for the process model. The composition of ISM with LP and DEVS models resulted in the first realization of what is now called the Optimization Simulation Forecast (OSF) platform. It could handle a single echelon supply chain system consisting of single hubs and single products

In this thesis, this single-echelon simulation platform is extended to handle multiple echelons with multiple inventory elements handling multiple products. The main aspect for the multi-echelon OSF platform was to extend the $KIB_{DEVS/LP}$ such that ISM interactions with the LP and DEVS models could also be supported. To achieve this, a new, scalable XML schema for the KIB has been developed. The XML schema has also resulted in strengthening the KIB execution engine design. A sequential scheme controls the executions of the DEVS-Suite simulator, CPLEX optimizer, and ISM engine. To use the ISM for multiple echelons, it is extended to compute forecast customer demands and safety stocks over multiple hubs and products.

Basic examples for semiconductor manufacturing spanning single and two echelon supply chain systems have been developed and analyzed. Experiments using perfect data were conducted to show the correctness of the OSF platform design and implementation. Simple, but realistic experiments have also been conducted. They highlight the kinds of supply chain dynamics that can be evaluated using discrete event process simulation, linear programming optimization, and heuristics forecasting models.

ACKNOWLEDGEMENTS

I would like to acknowledge Dr. Hessam Sarjoughian for his mentorship throughout my work at ASU and for being the advisory for this research. From Intel, I would like to give special thanks to Dr. Gary Godding for his past work and ongoing help within the domain of supply chain. Also from Intel, I want to give thanks to Dr. Asima Mishra and David Bayba for help with the definition of the Inventory Strategy Module and Multi-Echelon Inventory Optimization. For implementing the first iteration of the Inventory Strategy Module within a Discrete Event System Specification (DEVS) model and going through the verification process of the code, I want to give thanks to Dr. Mohammed Muqsith, a former student of ASU. Thank you to the members of the supervisory committee for qualifying this work. Finally, thanks to Intel for continuing to sponsor this project.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

1 INTRODUCTION

1.1 Purpose Statement

This report was written to satisfy degree requirements for Masters of Science in Computer Science and course requirements for independent study with Professor Sarjoughian in order to describe the accomplishments made in the development of the Knowledge Interface Broker (KIB). This work is also used in the development of the multi-echelon supply chain simulation project. The bottom line goal of the Supply Chain project is to develop a multi-echelon simulation model with multi-echelon inventory strategy and optimization modules. All work must be scalable for large models on the order of hundreds of components. The purpose of model development within this system is to better understand the behavior of some semiconductor products.

1.2 Intended Audience

The intended audiences for this report are the members of the graduate committee Dr. Hessam Sarjoughian [chairperson], Dr. Hasan Davulcu, and Dr. Georgios Fainekos; sponsor Intel, which includes employees working with the Supply Chain Simulation project; and anyone in the field either continuing this work or using this as a source in their own work. A portion of the code accompanying the design described in this thesis is planned to be released for general public use.

1.3 Problem Definition

1.3.1 Semiconductor Supply Chain

In any type of industry that needs to distribute products in different physical locations with varying markets, there is a constant question of how much, how often, and where to distribute the products in order to meet the end demand of each

customer. Enough of each product needs to be shipped in order to meet demand as soon as products are requested to keep the customers happy and increase the chances of repeat business. On the other hand, if too much product is built, this often results in wasted inventory and financial loss to the product maker. Ideally, exactly enough product should be shipped at the right time instances to all customers in order to meet the exact demand value and nothing more.

To get a better idea of what the customers need, companies can ask for an estimate of how much product will be needed well in advance. If the customer could give perfect data, the problem could be relatively easily solved. Unfortunately, to keep customer ratings high, companies need to allow the customer to change their orders at a moment's notice, close to the delivery date. Companies need to look at the forecasted demand numbers and compare them to the historical data to make a prediction of the customer's actual need.

1.3.2 Optimization, Simulation, and Forecast

The Optimization, Simulation, and Forecast (OSF) platform is built atop previous efforts (Godding 2008, Huang 2008). The OSF platform (Sarjoughian et al, 2012) introduces forecasting capability to earlier simulation/optimization platform built using Linear Programming (LP), Discrete Event Simulator (DEVS), and $KIB_{DEVS/LP}$ (Godding 2008). The OSF is conceptualized and developed using a simple logistics supply chain which has a customer warehouse, a single shipping route, a single hub, and a single customer. The supply chain supports single products moving from customer warehouse and delivered to customer.

The optimization and simulation models are developed in OPL-Studio/CPLEX optimization engine and the DEVS-Suite simulator, respectively.

OPL-Studio is a platform managed by IBM. This platform is used to develop LP models. When an LP model is compiled by the platform, it may then run through the CPLEX optimizer to compute an optimal solution given values for the defined set of constraints. The DEVS-Suite modeling and simulation platform was built and is managed by staff and students at ASU under the guidance of Dr. Hessam Sarjoughian. For this research, the term "model" is used to refer to an engine that can execute a set of instructions. In this sense, the DEVS-Suite simulator combined with just mentioned supply-chain process model is called the DEVS model. The KIB transforms data and control messages between the optimization and simulation models. The KIB is itself a standalone model that can be specified in XML. The KIB model in the form of XMLs has an accompanying execution engine which is developed in Java. The KIB execution is governed using the DEVS-Suite simulator protocol. The optimization model is defined as a Linear Program (LP) and is used to compute an optimal solution given a set of constraints. It is used in this instance to determine how many products to be released from a component warehouse to a hub given the state on the model at each point. The model, built in DEVS, is a discrete-event representation of a single-echelon system which can handle a single hub shipping a single product to one customer. The executions of the DEVS, LP and KIB models are governed using the DEVS-Suite simulator protocol.

The structure of the OSF platform is shown in Figure 1. The forecast model consists of an Inventory Strategy Module (ISM) that looks at historic and forecast data to determine how much extra stock to hold at the hub. This data is sent to the optimization module for computing release command to the simulator. Even though

the execution of ISM is entirely functional, it is devised as an atomic model within DEVS in order to ensure it is used correctly alongside simulation model.

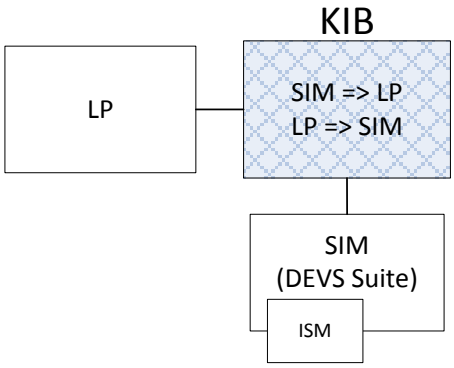

Figure 1. Supply Chain Model Composition Structure

1.3.3 Knowledge Interchange Broker

A KIB instance is defined using XML. This XML selects the functionality of each interface and the KIB interaction model. An interface is defined as a Java class that is written to connect the functionality of any external model to the KIB model. The high level view in Figure 2 shows an example of a KIB system interfacing DEVS and LP models.

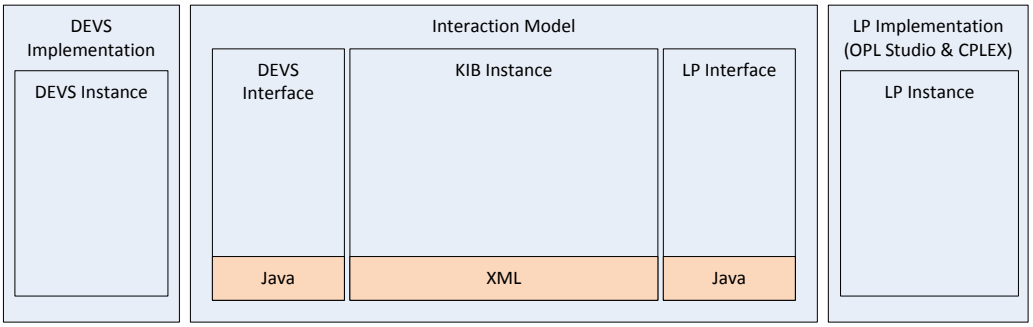

Figure 2. High Level View of KIB Interaction Model for DEVS and LP

1.3.3.1 XML Schema Design

The original KIB XML specification is difficult to be extended. Because many of the elements within the XML were labeled with the names of the interfaces themselves, if an XML schema was created, the schema would have to be re-written for each new

interface that is added to the KIB. Under each interface element defined, the schema would contain redundant definitions of nodes. This becomes cumbersome and is not scalable. The specification needs to be updated so that an XML schema can be defined to allow for XML instances using all current and future model interfaces without any change to the XML schema definition itself.

The structure of XML files created for the KIB allowed for multiple data variables for each data input or output, but these elements were defined as a single string for each data definition. This string would then have to be parsed within Java code to interpret the meaning. This same process was done upon each relationship and the control definition. This made it more difficult to define model elements, especially for someone who is not familiar with KIB semantics.

Modules within the KIB define partitions of data within the connecting models. Keep in mind that this should not be confused with the forecast modules which form the ISM, a separate model in the system. As new modules and their relationships were defined within the KIB XML, the file became large in size and difficult to manage. There was no systematic method to break up the XML definition into separate, manageable pieces. This also posed difficulties for XML reuse . If another model was created that was structurally similar, but contained different relationships and control, anew file had to be created.

1.3.3.2 KIB Structure

A module is an autonomous component within an interaction model that is given different definitions depending on the interface implementation within Java code. The design of the KIB itself called for a set of interfaces to be held within each module definition. Refer to Figure 3 for an example of a KIB instance model at the

conceptual level. When creating a KIB instance model, modules had to be conceptualized as entities that existed between two interfaces. This design was implemented due to the early formalization that each module component should have a single corresponding module component in the mapped model. As the design was expanded, the constraints needed to be relaxed in order to allow data transformation between two modules of differing names. This can be seen in Figure 3 when DEVS in Module B needs to communicate to LP in Module C. This design added difficulty in conceptualizing the structure because now modules within the KIB not only provided links between interfaces, but links between modules of differing names were acceptable by the design too.

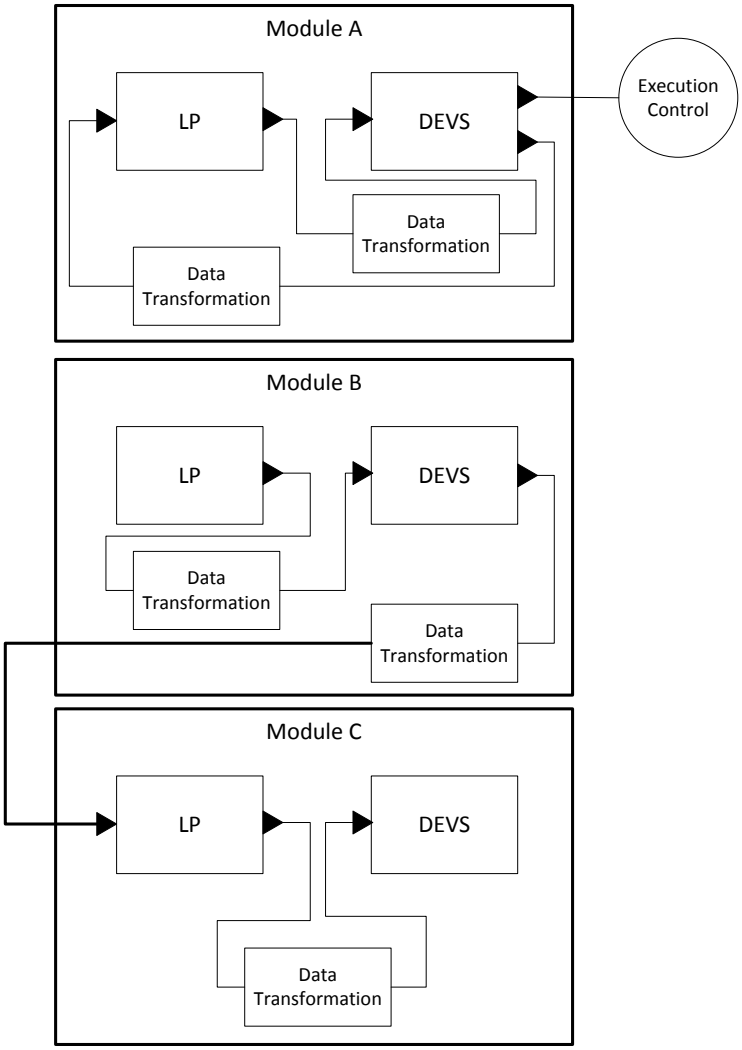

Figure 3. Original Conceptual Design of Model to Model Transformation

The KIB also assumed that each model within the system was labeled with the same name. This not only restricts the designer to label each model with the same name, but it also makes it impossible to define multiple models within the same interface for the KIB. There was a 1:1 distinction between an interface and an instance of a model defined in the KIB. This restricted the definition to a maximum of a single model for each interface.

1.4 Contributions

The main contribution of this work has been the following:

- The development of KIB XML schema design and refactoring of the KIB serves as a foundational component in the scalability of the OSF platform.
    - The structure of the KIB has been redesigned for better usability and comprehension of model design. Since separate model designs are formed using their own definitions of module components, it makes sense to define these elements separately within the KIB.
    - To define a schema that meets the current need and is scalable for future development, elements within the XML design has been generalized. The naming of elements is not used to enumerate elements within the KIB. Instead, attributes are used to select items within an enumeration.
    - To better define and constrain the structure of a KIB model built in XML, post-process parsing of data has been removed. In order to accomplish this, each element of the KIB has been well defined. From these definitions, a schema was developed with the proper structure and constraints. The culmination of all this data can serve as a user's guide for future use and development.
    - This document itself serves as a user's guide for future development of KIB models.
- The ISM has been greatly expanded to test the scalability of the KIB.
    - The modularization of the ISM was necessary as it is designed as a functional formalism. Communication to and from the ISM is now handled through the KIB using a new interface.

- The ISM has been expanded to handle multiple hubs and products. Initialization of data sent through the KIB now contains arrays defining hub/product pairs.
- The design of the ISM was extended to handle multiple echelons using research on Multiple Echelon Inventory Optimization (MEIO) methods. Although, this work has yet to be completed, the structure has been put in place to send the correct sets of data through the KIB.

2 BACKGROUND AND RELATED WORKS

2.1 Background

2.1.1 Definition of Supply Chain

The work in this project specifically references supply chains in the semiconductor domain. A supply chain, in a very general sense, contains product generator elements followed by shipping and inventory elements with customers as end nodes. Product usually moves in one direction toward the customer, but in some circumstances, may move in a vertical or opposite direction. For simplicity, the model used in this research only allows for product to move toward the customer which, in most cases, is the path with the lowest cost and highest return.

When looking at a supply chain purely in terms of inventory movement, the supply chain consists of the model elements of inventory, shipping, and customer components which are diagrammed in figures 4, 5, and 6 respectively. The inventory model receives product from the previous element at any time. At some point, that product is moved from the incoming bucket into the store where it is processed. Product is only moved from the store to the outgoing bucket when a release command is received from the optimizer. The shipping model is similar to the inventory element with two distinctions: 1) When product is moved into the store, it is stored in buckets of higher granularity. As time progresses, product moves from one intransit bucket to the next. 2) Once the product reaches the final intransit bucket, it is immediately moved to the outgoing bucket without any release command. This means that the shipping element cannot be externally controlled once product has entered it. The customer model is simplistic in that it receives whatever product that comes to it in order to meet demand.

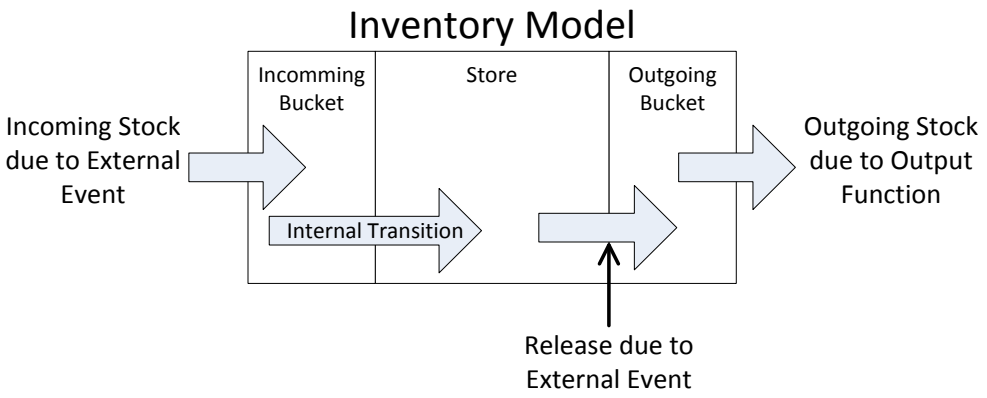

Figure 4. Inventory Model

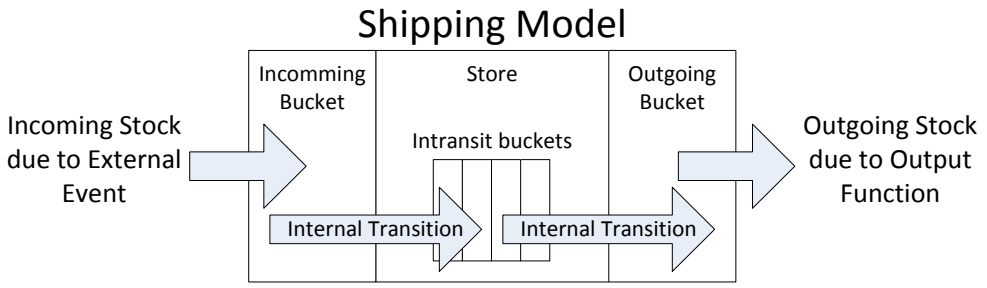

Figure 5. Shipping Model

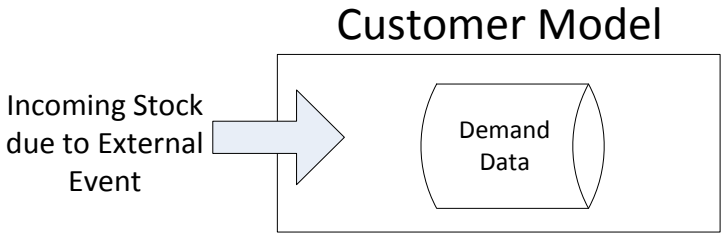

Figure 6. Customer Model

Figure 7 shows an example of a simple double echelon supply chain. The "FA" inventory element on the very left represents a factory. This is where inventory is generated. The product is then shipped to the "CW" inventory element which represents the Component Warehouse. The CW may handle the packaging or another phase in assembly. Finally, the product is shipped to either the "Hub1" or "Hub2" inventory elements. Product is then immediately distributed to the "GC" elements or Geo Customer which, in the real world, is located in the same physical location as the hub.

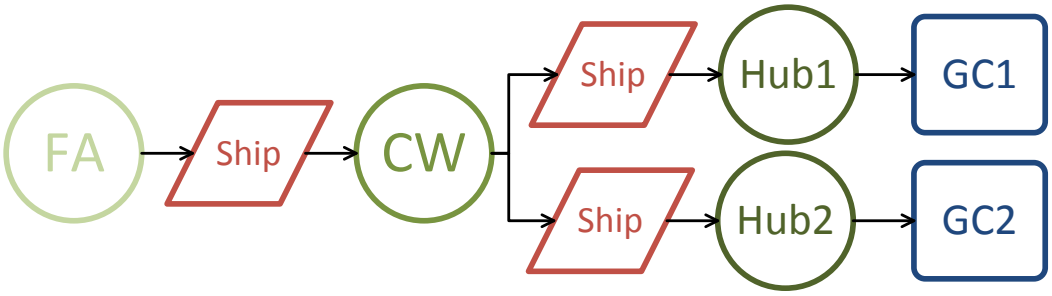

Figure 7. Supply Chain Example

The Supply Chain Council is involved in creating a standardized framework for supply chains called the Supply Chain Operations Reference (SCOR) model. Although this research does not follow this model, the work that has been done ties in with the section of the SCOR model that pertains to "capturing the configuration of a supply chain" ("What is SCOR" n.d.). In this section, the supply chain model is broken up into segments: plan, source, make, deliver, and return.

2.1.2 Multi-Echelon Inventory Optimization & Sequential Based Stock

The theory behind Multi-Echelon Inventory Optimization (MEIO) is to compute a safety stock value for each single echelon starting from the most downstream element and passing the result up. As we move up the supply chain, the average delay of each stage is applied to the customer demand. In other words, to satisfy the demand of the downstream element on time, we must release stock X weeks early where X is the time that it takes to ship product to the downstream element (Graves & Willems 2000).

2.1.3 XML and XML Schemas

XML was originally developed in 1996 by the World Wide Web Consortium (W3C) for "ease of implementation and for interoperability with both SGML and HTML." One of the goals of the XML specification is to make it easy for a developer to create documentation ("Extensible Markup Language" n.d.). This makes an XML

definition easy to read, but XML files are generally not lightweight. This data formalism was used to create KIB models because it allows someone who is less familiar with code design to develop a model.

The structure of an XML file is largely comprised of elements and attributes. Elements can contain simple or complex data. Under the classification of complex data, there may be a set of single or multiple child elements. Attributes are singleton simple data variables that can only exist within an element. The W3C organization provide a set of uses for an attribute, but in this research, all attributes defined will be used to specify a value that is attributable to an element. For further definition of XML, refer to the specifications available on the W3C website ("Extensible Markup Language" n.d.).

The structure of an XML schema was first designed in 2001 by the W3C to define the structure and constraints of an XML document ("W3C XML Schema" n.d.). Any simple value defined can be constrained to a set of values such as an enumeration of strings or a numeric value within a set range and/or granularity. For further definition of an XML schema, refer to the specifications available on the W3C website ("W3C XML Schema" n.d.).

2.1.4 DEVS/LP Knowledge Interchange Broker (KIB)

2.1.4.1 History

Since 2003, Intel has been working with ASU to develop models for evaluation and improvement of its supply-chain processes. More recently, this effort has expanded to address the optimization of inventory stocking to meet or exceed specified service levels across a multiple logistics echelon. The DEVS simulator has been used to develop a skeleton model of a single-echelon supply chain. This model consisted of

inventory and shipping components. An inventory component processes stock then holds it until an appropriate release command is generated. A shipping component will hold its incoming products for a period of time before delivering them to the next component in the supply-chain process.

At every interval of time that the system runs for, an LP model is used to determine the optimal plan for the supply chain to generate release commands for the inventory components. The LP is modeled in OPL Studio which is written in C++, unlike DEVS which is developed in Java. In order to get the two models to communicate, an interface was designed to overcome not only the differences in implementation languages, but in the simulation operation as well. This interface is known as the Knowledge Interchange Broker (KIB) (Godding 2008).

The KIB at its core is conceptualized to be generic as part of Gary Wade Godding's (2008) defense for his doctoral thesis entitled, "A Multi-Modeling Approach Using Simulation and Optimization for Supply-Chain Network Systems" (Godding 2008). It has a model that formulates data transformations under a time-based execution control scheme. Time is updated from the controlling model which is the model that also calls the KIB. For DEVS/LP, at some time interval, the LP model receives information from the DEVS (controlling) model via the KIB model. LP then computes release commands which are sent to the DEVS model via the KIB model. In this way, all communications (i.e., data transformation and control logic) between the LP and DEVS models are managed by the KIB. It is important to understand the distinction of the controlling model in the supply chain system. DEVS depends on the execution of the LP. Therefore, even though DEVS is labeled as the controlling model for the KIB, from a systematic perspective, the LP is the

model that controls the DEVS models with release commands. The purpose of Gary Godding's thesis was to make a generic modeling system that brokers the interaction between a simulation and optimization models synchronously. The motivation behind the project was to develop a supply chain system from a modeling and simulation perspective. Different aspects within the supply chain planning process depended on differing principals and are modeled using different formalisms. The KIB concept with a basic theory is described in (Sarjoughian 2006; Sarjoughian and Plummer 2002). The concept of KIB was further developed by Gary Mayer (2009). His work can be seen in (Mayer 2009; Mayer and Sarjoughian 2009).

From the DEVS/LP KIB, different branches were created to support new methods with realizations. This includes a supply-chain system communicating between DEVS and Model Predictive Controller (MPC) as well as human and landscape dynamics with communication between DEVS and a Cellular Automata (CA) model. These realizations can be seen in (Godding 2008; Godding, Sarjoughian, and Kempf 2004; Godding, Sarjoughian, and Kempf 2007; Huang 2008; Huang et. al. 2006; Huang et. al. 2009). The work done in this thesis can be applied to any branch of the KIB as well as any possible future research.

2.1.4.2 Overview of Transformations

The way that a KIB model was defined was through an XML file where source and target modules were defined for the required interfaces as well as the necessary transformation. The XML file also provided a control which defined a controlling model element and an interval to execute. A controlling model element needed to be defined to keep track of the current time and when the interfacing models need to be executed.

As the KIB system was developed, several transformations between 3 dimensions were added. In the KIB, a set of data form a table where key values are defined. Each set may contain several single values or 1-dimensional arrays. As the KIB receives this data, it is time stamped, which forms the third dimension (Godding 2008). Refer to Figure 8 for an example of aggregation of data from current set or sets over time. Doing the computation on the 3 dimensions of data through the KIB instead of at the source or destination models simplifies the process for the model designer. The aggregation and disaggregation of data over time accounts for differing model granularity. Simple mathematical functions are built in such as min, max, and mean. Data may also be set to be treated as sets or units. All these features are further documented in chapter 0 with the development of the KIB XML schema.

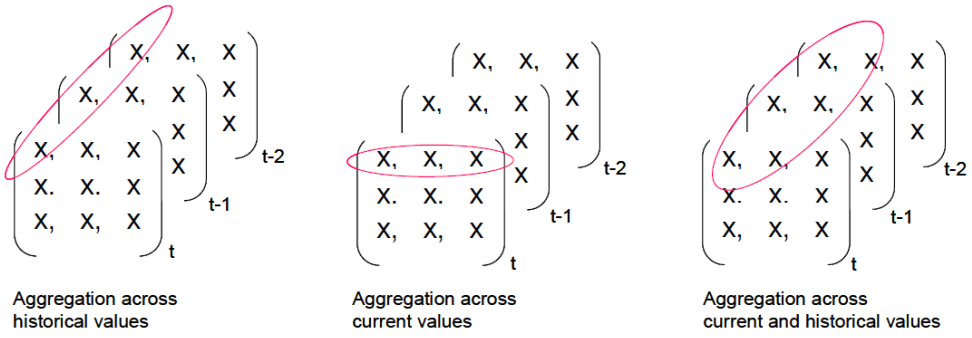

Figure 8. Types of Data Aggregation (Godding 2008)

If the model in Figure 7 needed to be implemented into the OSF platform, this model would first need to be designed in the simulator. In this example, the simulation runs at a daily time granularity and the optimizer runs at a weekly granularity. This means that the optimization executes once for every 7 time ticks of the simulator. At each tick of time, the inventory models report to the KIB their Beginning On Hand (BOH) and Actual Out (AO) data; the shipping models report

their in-transit data and AO data; and finally the time stamp value as a single integer value. At time of execution of the optimizer, 7 sets of data are available at the simulator data store. Only the most recent state data is meaningful to the optimizer, so only the newest set of data should be transformed.

For the optimization to read the in-transit data correctly, the array of values needs to be transformed into a row for each value. Refer to Table 1 for an example of this transformation. The array of size 2 with the value of [50, 75] is transformed so that each row contains a single integer value for quantity. This is achieved by adding the key column of period. The BOH and AO data don't need any conditioning and can be passed right through.

Table 1. Array to Set Lines Transformation Example

Shipping Name	**Product**	**Quantity[2]**
CW2HxShip	P1	[50, 75]

TRANSFORMATION
\/

Shipping Name	**Product**	**Period**	**Quantity**
CW2HxShip	P1	0	50
CW2HxShip	P1	1	75

The optimizer generates a set of release commands which needs to be distributed to each of the inventory elements. To do this, a disaggregation transformation is used. Refer to Figure 9 for an example of disaggregation. Each value is divided equally between the 7 time buckets within the simulation. The standard rounding algorithm is used to round each resulting value to the nearest integer. At each time tick, the KIB provides to the simulation a single set of values over 7 ticks. At the end of the 7th tick, the optimizer is run again for another 7 sets of data.

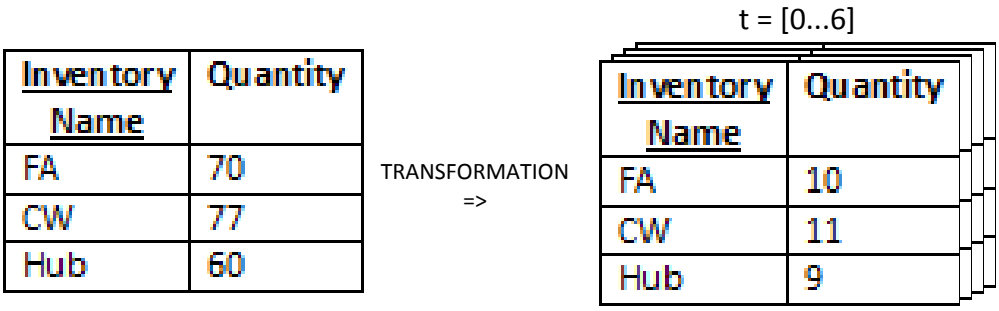

Figure 9. Disaggregation Example

2.1.5 Integrating Forecast Model with Optimization and Simulation Models

The definition of the ISM was provided by personal communication with employees of Intel. This definition was then used to create a functional implementation of the model. Some arbitrary test data was also provided in order to test and qualify the functionality.

Working with Input Demand Data

The combination of hub H1 and product P4 was selected to do analysis on for this project. The chart in Figure 10 shows the comparison of Historic Forecast demand, HFC, and Actual Customer Demand, ACD, for hub H1 and product P4. This data is used to compute a bias within the ISM. All data in this chart is considered as historic data. Therefore, for example, if the current time period is week 7 then the ISM would only be able to view the data up to week 7.

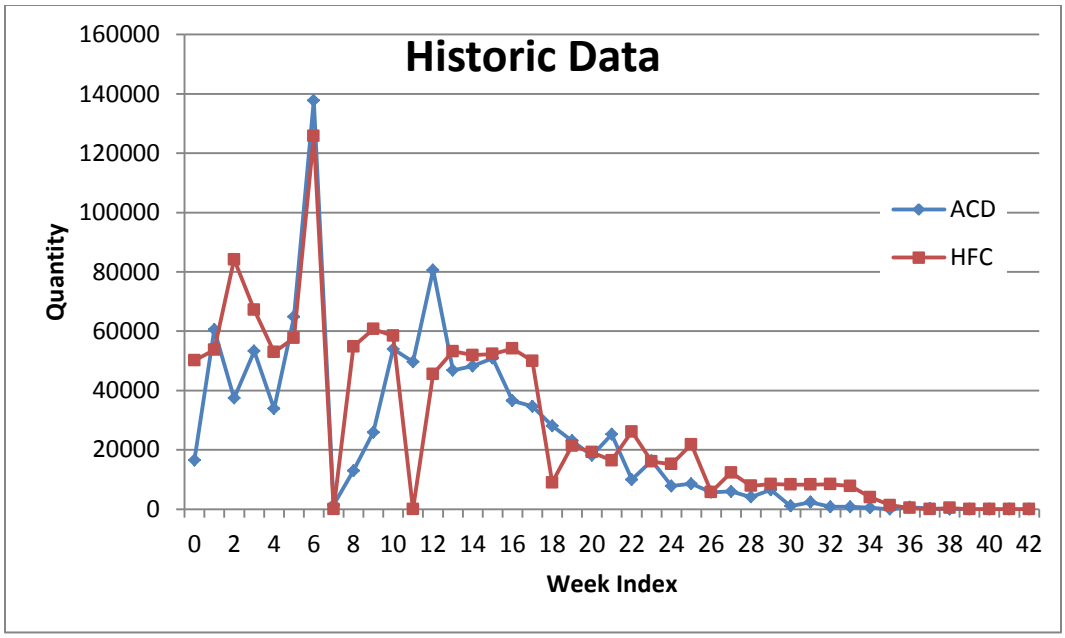

Figure 10. Hub H1, Product P4 – Historic Data

At each time period, forecasts of future weeks are made. The chart in Figure 11 shows a three dimensional representation of the forecast data for the ISM. Forecasts evolve over time as new data arrives. For example, the forecast for week 11 at week 7 is 46816 units. The following week, week 8, the forecast for week 11 is 46654. Between week 7 and week 8, a total of 162 units were canceled for week 20. This volatile forecast data adds difficulty when finding an optimal solution for the system. This is, of course, what the ISM is going to bias against.

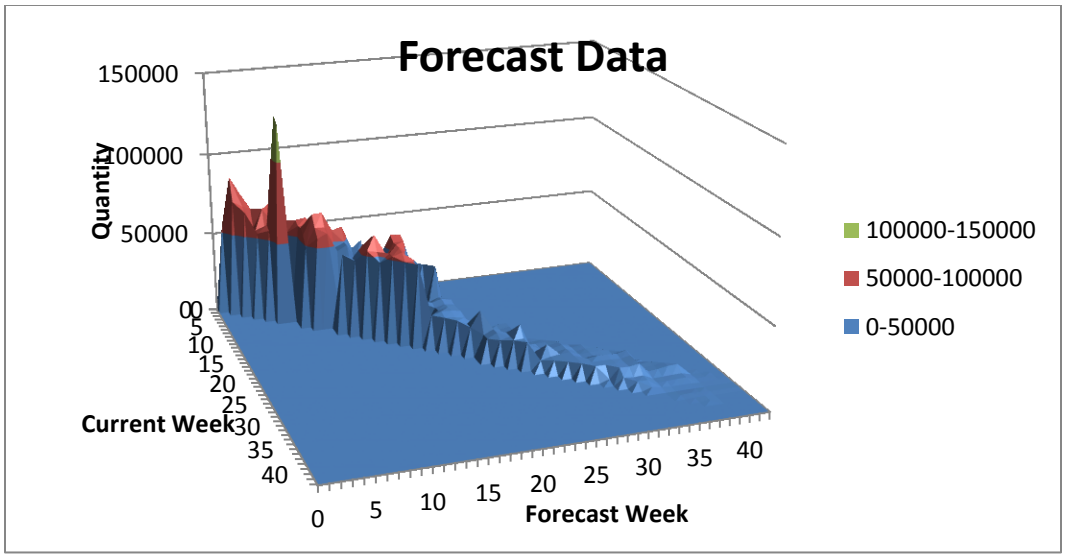

Figure 11. Hub H1, Product P4 – Forecast Data over Time

Computation of Safety Stock

Refer to Table 2 and the corresponding graph in Figure 12 for an example of the data the ISM uses for week 7. The data shown up to week 7 is historic data, while the data after week 7 is forecasted data. The single echelon ISM first computes a multiplier based on the smoothing algorithm, target service level, replenishment time, and how well the historic actual customer demand did against the historic forecasted demand. Historic data is marked in blue in Table 2. The smoothing algorithm was an implementation of the smoothing interface of either exponential, kernel, or no smoothing. The target service level is a value between 0 and 100%. This this can be seen as a customer satisfaction level to be targeted. Replenishment time is the time in weeks for inventory to go from the upstream inventory, through a shipping delay, and be available for the customer in the downstream element. This includes any time that the downstream inventory takes to process the product.

Table 2. Historic and Forecasted Data Example

	Week Index	Actual Customer Demand	Forecasted Customer Demand	
	0	800	460	Historic Data
	1	470	530	
	2	480	520	
	3	510	520	
	4	370	540	
	5	350	500	
	6	280	90	
This Week->	7	230	210	
Next Week->	8		190	Forecasted Data
	9		160	
	10		150	
	11		120	
	12		130	
	13		140	
	14		50	
	15		0	

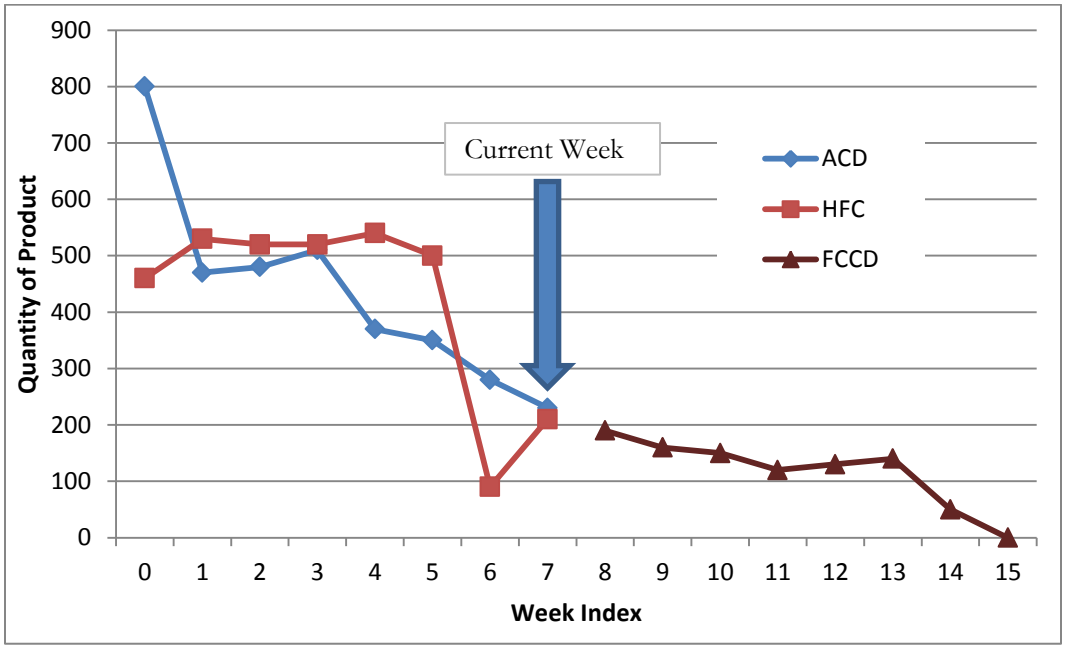

Figure 12. Graph of Table 2

A bias is calculated based on all the data that the ISM uses for the current time period. This bias is then applied to the forecasted data to produce a safety stock value. The safety stock value tells the optimizer how much extra stock on top

of the future demand to keep in inventory in order to achieve the desired service level.

2.2 Related Works

2.2.1 Using Model Predictive Control in a Supply Chain

The work done by Jay D. Schwartz, Manuel R. Arahal, Daniel E. Rivera, and Kirk D. Smith (2009) focuses on a supply chain planner in which the main goal is to keep inventory at a set level at a specific location using a Model Predictive Control (MPC). In this design, an MPC is connected to an inventory component in a feedback loop configuration with an injected feed-forward demand forecast signal. At each time instance, there is a set level of stock that must be left in the inventory after the inventory release of the previous instance. This set level is similar to a safety stock as discussed in 2.1.5. A fluid analogy is used to describe the process where a fluid needs to stay at a certain level within a tank. More fluid needs to be added to the tank at the same rate the fluid is released in order to maintain a given fluid level (Schwartz et al. 2009; Schwartz and Rivera 2010).

The MPC used in the configuration as described above handles the prediction of inventory movement at each individual location. Since it does not make a prediction of the movement of product as a whole, it may not be scalable to more complex models. The LP as discussed in 2.1.4 does the same job of the MPC in this instance for simple models by setting constraints such that the expected inventory after a release will not fall below the given safety stock value.

2.2.2 Inner and Outer Loop Optimization

The work done by Wenlin Wang, Daniel E. Riviera, and Hans D. Mittelmann (2009) focuses on a semiconductor supply chain system with a stochastic "outer loop,"

which runs planning at a lower granularity, and a stochastic "inner loop," which makes day-to-day decisions at a higher granularity. Similar to what is discussed in 2.1.4, an LP optimization model is used to create planning at a lower granularity. Inventory algorithms are used to compute safety stock values, similar to the ISM discussed in 2.1.5. After the LP model is executed, the results are split over 7 days and sent to the Model Predictive Control (MPC) which makes day-to-day optimization and planning in feedback and feedforward configurations. The MPC works similar to what is described in 2.2.1 except the data computed by the LP is also used by the MPC in order to make better predictions relating to the state of the system as a whole. (Wang, Riviera, Mittelmann 2009).

The focus of the work discussed above is on how to handle higher granularity, stochastic demand with a plan generated by a lower granularity optimizer. This issue is outside the scope of this thesis since no day-to-day demand is provided to the system, so day-to-day demand is generated by evenly distributing the given week-to-week data into 7 buckets. Therefore, it is much easier to predict demand on a day-to-day basis by simply dividing the weekly plan generated by the LP into 7 equal buckets. However, since the work in this thesis demonstrates a design to connect model components with scalability in mind, it would be feasible to combine these works in the future.

3 APPROACH

3.1 Knowledge Interchange Broker Model XML Schema

3.1.1 Premise for Design

The structure of the KIB needs to be updated without changing the underlying functionality. The diagram in Figure 13 corresponds to the high level view of KIB model in Figure 2 and shows how different elements in each model are mapped to a module within the KIB. Each model implementation has a different definition as to how its module is defined. In general, a module is an atomic component of a model. During runtime, the interfaces can read from the data stored in the KIB module outputs and deposit the data into their respective models. After a model is executed, an interface can then read the output data given by its model and deposit into the correct module inputs of the KIB.

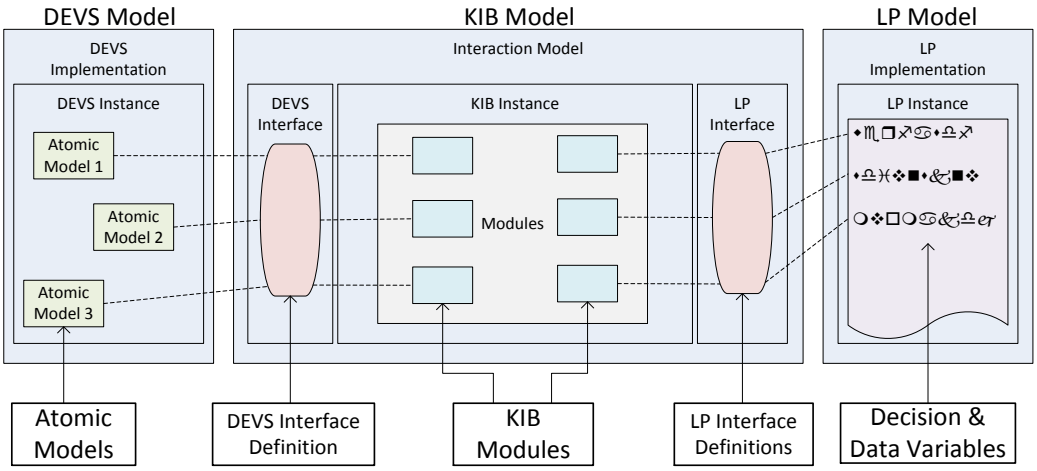

Figure 13. KIB Model Interaction

Also defined within a KIB instance model is a set of mappings as depicted in Figure 14. A mapping itself defines the source module output and destination module input though which data will be routed. Within a mapping, there are a set of transformations that define how data is to be manipulated in this block. Transformations can manipulate data as a whole and/or as a singular value. When a

mapping is called to execute during runtime, all transformations within that mapping will be executed based on a priority set within the KIB.

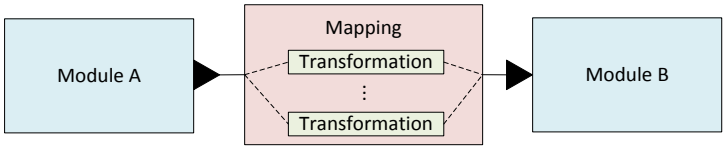

Figure 14. KIB Mapping

The diagram in Figure 15 shows a conceptual representation of the new XML schema design which corresponds to the same example given in Figure 3. Each model is associated with an interface within the KIB. Within each model, there are a set of modules. Unlike the previous design, modules belong to each model (interface) and the transformations are completely separate entities from the modules. Take note that the solid arrows in the diagram show how data moves though the components during transform and not how the schema is to be defined. Since the transformation blocks are now completely separated from the modules, a source module output and target module input need to be explicitly referenced using the distinct module and data input/output names for each transformation. The control component, like the previous design, is still a separate singleton entity which references a data output line.

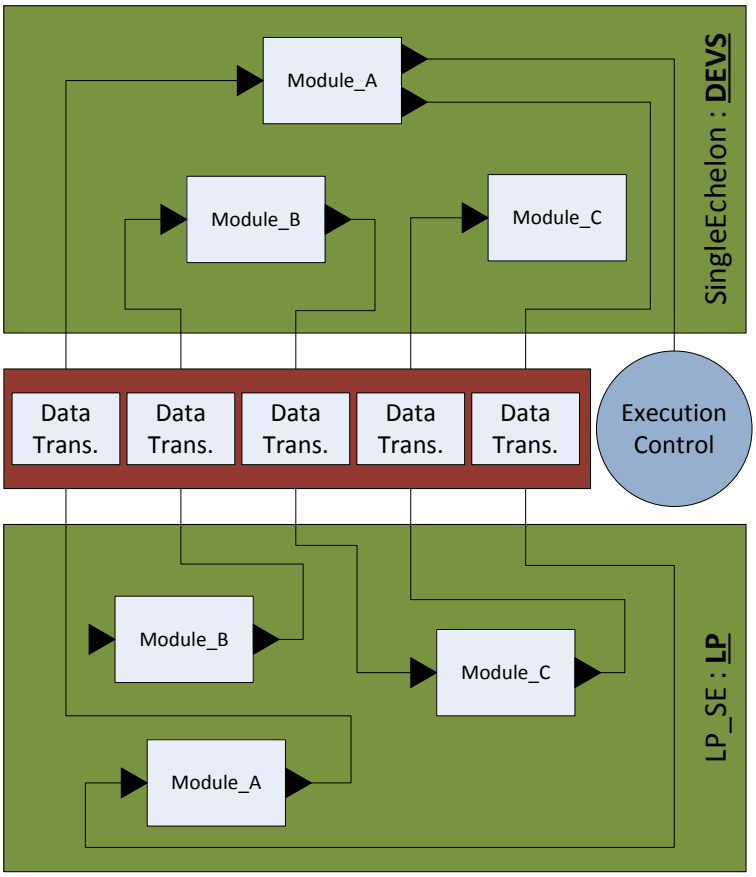

Figure 15. Proposed Conceptual Design of Model to Model Transformation

3.1.2 Decomposition of XML

To assess the problem with code reusability, the KIB XML can be broken up into smaller pieces. After removing the transformation entities from the modules themselves, the XML can then be broken down into three types of data. The diagram in Figure 15 shows this split with the colors green, red, and blue. The green elements represent groupings of model definitions with their accompanying modules for the KIB. The red element in the middle then represents the module to module mapping and transformations. Finally, the blue element represents what data variable will be used as the KIB clock. The model and transformation elements can then be further decomposed by the designer as necessary.

3.1.3 Generalization

Refer to Figure 16 for an example of how a KIB XML file was originally structured. The way that nodes were named did not allow a schema to be designed to satisfy current and future functionality. To correct this problem, the naming of each node can be made more general and instead use attributes and child elements to define what specific type of data there is under the element.

```
<?xml version="1.0" encoding="utf-8" ?>
<KIBMODEL>
      .
      .
      .
  <MODULE_SPECIFICATION Name = "H1">
    <LPINTERFACE>
      <DataVariable Name="H_BOH">
        <Type>Collection,Record,Key:String:hub,Key:String:product,Int:quantity</Type>
      </DataVariable>
      <DecisionVariable Name="H_RELEASE">
        <Type>Collection,Record,Key:String:product,key:String:destination,Int:period,Float:quantity</Type>
      </DecisionVariable>
    </LPINTERFACE>
    <DEVSINTERFACE>
      <DataOutput Name="BOH">
        <Type>Collection,Record,Key:String:hub,Key:String:product,Int:Quantity</Type>
      </DataOutput>
      <DataInput Name="RELEASE">
        <Type>Collection,Record,Key:String:product,Key:String:destination,Int:Quantity</Type>
      </DataInput>
    </DEVSINTERFACE>
    <INTERFACE_RELATIONSHIP>
      <DEVSLPMAP>
        <DEVSNAME>BOH</DEVSNAME>
        <LPNAME>H_BOH</LPNAME>
        <DATA_TRANSFORMATION>NONE</DATA_TRANSFORMATION>
      </DEVSLPMAP>
      <LPDEVSMAP>
        <LPNAME>H_RELEASE</LPNAME>
        <DEVSNAME>RELEASE</DEVSNAME>
        <DATA_TRANSFORMATION>FloatToInteger:Round,Index:period,quantity,Quantity</DATA_TRANSFORMATION>
      </LPDEVSMAP>
    </INTERFACE_RELATIONSHIP>
  </MODULE_SPECIFICATION>
      .
      .
      .
  <KIBCONTROL>
        <CONTROLLING_MODEL>DEVS</CONTROLLING_MODEL>
        <MODULENAME>Synchronization</MODULENAME>
        <VARIABLENAME>LP_SYNC</VARIABLENAME>
        <CONTROLTYPE>Periodic:DEVSCYCLES:1</CONTROLTYPE>
  </KIBCONTROL>
</KIBMODEL>
```

Figure 16. Original KIB XML Definition Example

The original design contained 2 interfaces; DEVS and LP. Within a KIB XML file under a module, each interface needed to be enumerated by using one of the nodes <DEVSINTERFACE> or <LPINTERFACE>. In the redefined design, a node <Interface> may be used with an attribute 'name' to enumerate the type of interface. To take this a step further, a model name can be attached to the interface so that there may be multiplicity of models within a KIB design. So now a node <Model> may be used with a 'name' attribute naming the model and an 'interface' attribute that connects the model to an interface. The distinct name of the model must now be referenced within the remainder of the XML.

In the previous design, in order to select which is the source interface and which is the target interface, nodes with names like <DEVSLPMAP> or <LPDEVSMAP> were used to select a DEVS->LP or LP->DEVS mapping respectively. To make this more generic, a node <Map> may now be used with the attributes 'source' and 'target' which determine which interface is the source and which is the target. To take into account the design of the <Model> node in the generalization above and the original premise in mind, a source model, module, and data output with a target model, module, and data input must instead be defined under the <Map> element. Instead of defining source and target attributes, source and target elements may now be defined each with the attribute set model, module, and data.

The LP interface required the use of <DataVariable> and <DecisionVariable> element for inputs and outputs while the DEVS interface used <DataInput> and <DataOutput>. This broke any sort of generalized input/output elements that could be created. Since data variables and decision variables still map

to what the LP considers as input and an output, the definitions of these elements can be changed in code to match the other interfaces to come to a more generalized definition of an input and output within the schema.

3.1.4 Removal of String Parsing

Any entity of the XML that can be parsed should be further decomposed into XML attributes and elements. The following shows how each string is parsed and how the decomposition of this string can be handled by the XML file. Deeper explanations of what each of the definitions mean will be handled later on in this paper.

- Module Input/Output Definition

  The string in Figure 17 shows a sample definition of an input or output of a module.

  - Parts ① and ② tell the KIB that the following data is a collection of record definitions, but this is more or less ignored since all data should be a collection of records. Therefore, it will be ignored in the schema design.
  - Parts ③, ④, and ⑤ each define a record. If the flag "Key" is given before the definition then the record will be a key field. This can be handled by an attribute giving a Boolean value to specify whether the field is a key or not. If the flag "Array" is given before the definition then the record will be an array. The size of the array must be given after the definition of the type or set to "Variable" if the size of the array is a variable size. This can be handled with an optional attribute where if set, then the value is an array type. The value of this attribute should be either a positive integer or the string "Variable." In every field definition, there needs to be a type string that is either "String," "Float," or "Int" which can be handled by a type

attribute. Finally, the name of each field is defined which can be handled by a name attribute.

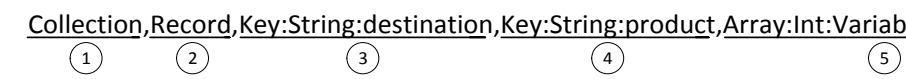

Figure 17. Module Input/Output String Definition Example

- Data Transformation Definition

  The strings in figures 18, 19, and 20 show a few examples of how a transformation string is defined which, as a whole, cover all the different attributes and flags that make up a transformation string.

  - In each figure, part ① defines the name of the transformation used. This can be handled by a name attribute with an enumerated list of all possible transformations available.
  - Part ② in Figure 18 and part ③ in Figure 20 set different types of rounding flags. The rounding can be "Round," "Ceiling," or "Floor" which map to the corresponding rounding function, with "Round" being the default. An optional rounding attribute can handle this with an enumerated list of the three type strings.
  - Part ② of Figure 20 defines how the data should be handled (granularity) in the transformation. The value here can be "Units," "Sets," "CurrentUnits," or "CurrentSets." This can be handled by another optional attribute with an enumerated list of strings.
  - Part ④ in Figure 20 gives an optional multiplier value by which the transformed value is multiplied by before being sent to the destination. This can easily be handled by an optional multiplier attribute.

- Part ③ of Figure 18 gives the index variable and a value associated with it. Only a single index variable may be given per transformation. This can be handled by an optional index element where both a name and index value needs to be given if it is defined.
- Parts ② and ③ of Figure 19 give field definitions. Some transformations require one or more fields to be defined while others do not require any at all. This can be handled by an optional field element where there must be a name and value if a field is defined. This element must have [0..n] multiplicity.
- Part ④ in figures 18 and 19 and part ⑤ in Figure 20 give the source variable. Part ⑤ in Figure 20 give optional starting and ending index values. This can be handled with a mandatory source element with the attributes that represent the variable name, starting index, and ending index. The name is required, but starting and ending indices are optional.
- Part ⑤ in figures 18 and 19 and part ⑥in Figure 20 give the target variable. Like the source variable, starting and ending index values may also be provided. This can be handled with a mandatory target element that has a definition same as the source element.

FloatToInteger:Round,Index:period=0,quantity,Quantity
① ② ③ ④ ⑤

Figure 18. Data Transformation String Definition Example 1

FieldValueToVariable,Field:product=prodX,Field:Destination=route66,quantity,quantity
① ② ③ ④ ⑤

Figure 19. Data Transformation String Definition Example 2

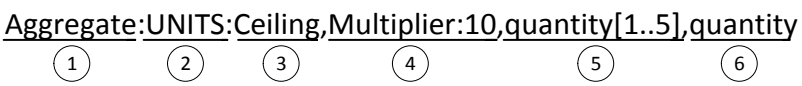

Figure 20. Data Transformation String Definition Example 3

- Control

  The string in Figure 21 shows an example of how a control type string is defined.

  - Part ① defines the type of control execution. In code, this value is saved, but never used. To leave room for future development, it was decided that this entry should be used in the new design. This can be handled by an attribute that has a one value enumeration of "Periodic" for the current version of the KIB.
  - Part ② was read in and ignored. This value has no meaning and will be removed.
  - Part ③ gives the frequency value. This can be handled with an attribute constrained to a positive, non-zero integer value.

Figure 21. Control Type String Definition

3.2 Using the KIB with the Inventory Strategy Module

Figure 1 above shows how the entire system was conceptualized at a high level. As stated in the problem, the Inventory Strategy Module (ISM) was a functional model, but was contained within an atomic model within DEVS. Since the ISM is built upon a different formalism, it makes sense to be a completely separate entity. Figure 22 shows how the ISM can be separated and the communication lines to be established through the KIB. The dashed lines show the communication that did not exist with

the previous design. There is a sequential order of communication and execution in this network:

1. Execute the SIM for one step
    - For our model, "one step" means the length of time the LP will optimize over. This is usually over one week.
2. Transform data SIM => LP and SIM => ISM
3. Execute ISM
4. Transform data ISM => SIM and ISM => LP
    - Communication from ISM back the SIM should only be used for transducer accumulation of data.
5. Execute LP
6. Transform LP => SIM
7. Repeat from step 1 until complete

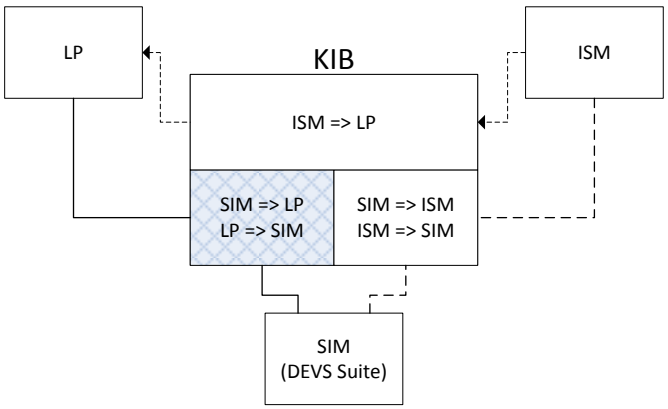

Figure 22. Separating ISM from SIM

3.3 Development of KIB

Combining the KIB design premise described in 3.1.1 with the modularization of the ISM as described in 3.2, the code structure should then look like it does in Figure 23. Like before, DEVS, being the controlling model, starts the KIB through the DEVS

Interface. At each DEVS time tick, the time is updated in the KIB and, if it is time to do a transformation, the transformations and execution sequence in 3.2 is run. When transformations or executions in any of the interfaces are required, the interface object functions are called to perform the desired action. Once the sequence is complete, the resulting data is returned back to DEVS and the simulation continues.

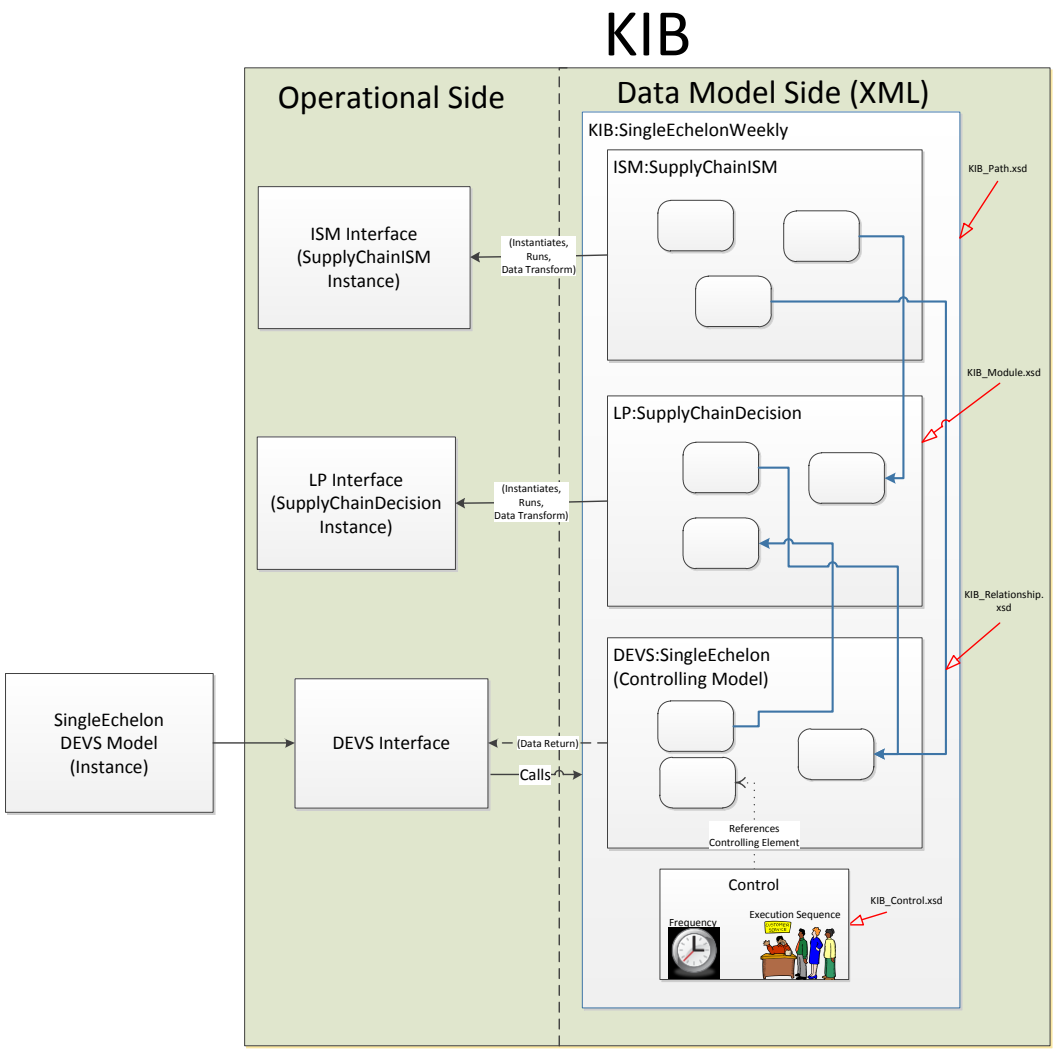

Figure 23. Interface Relationship with KIB

3.4 Experimentation/Evaluation

Refactoring the XML schema must not disrupt the functionality of the KIB execution within Java. To determine whether the redesign is a success, the previous

OSF model's KIB xml file will be re-written and the results will be tested to make sure the model produces the exact same data. On top of this, the previous unit tests must be rewritten and pass all execution pertaining to the KIB.

Some quantifiable measurements will test the scalability of the new KIB structure such as number of lines and number of elements. The new KIB design will then be used to develop a multi-echelon supply chain model. The ISM, being a new interface within the KIB, will be used to test the scalability of the KIB XML design when an addiction of an interface is required. The new XML schema and Java code must be easier to follow and manage. In other words, the design must make logical sense to one who is not familiar with the design.

XML code reuse is an important feature in any code design. As the KIB XML instance model is developed for the OSF platform, previous elements of the design that needs to be reused must be implemented without being rewritten.

There must be no post-process string parsing of the XML definition in Java code. All elements must be decomposed to their smallest atomic element within the XML schema. All constraints must be well documented.

A set of experiments will test the OSF system as a whole. The generation of meaningful experimental results shows that the entire system works with the new KIB and provides some data pertaining to the original goal to create a supply chain simulation model capable of running dynamic single and multiple echelon models. The run time scalability of the system should be tested to determine how feasible it will be to run models with thousands of components.

4 CONCEPT & XML DESIGN OF KIB

4.1 File Decomposition

With the new design premise, the XML schema is decomposed into three separate components: the modules, transformation and a control. The KIB_Paths.xsd schema defines the paths to where each piece of the KIB model is defined. Figure 24 shows a graphical representation of the KIB_Paths.xsd schema. Within this schema, one or more paths need to be defined for each set of components with the exception of the control which requires exactly one path. The path must define another XML file with the correct pieces of the KIB model. The paths can be absolute or be relative to the location of the KIB_Paths instance. Any XML that is referenced here must begin with a KIBMODEL element which signifies that the file is part of the KIB model.

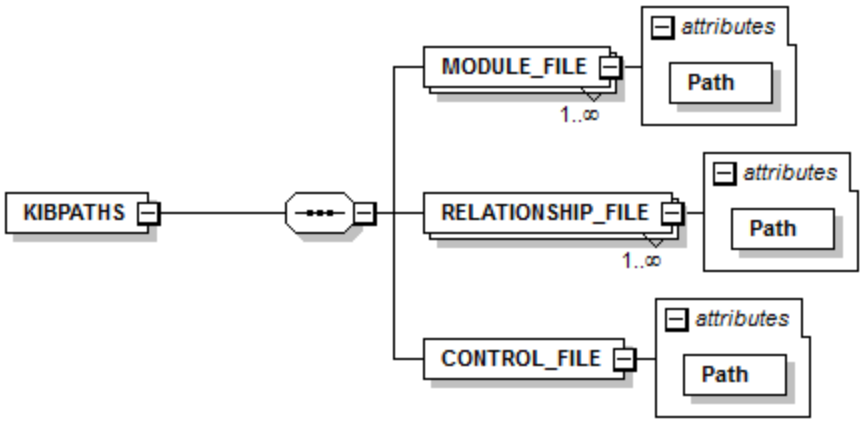

Figure 24. KIB_Paths.xsd Schema Graphic Representation

This design allows the entire KIB model to be decomposed into multiple parts and allow for partial definition in each file. This can greatly help out with development when the structures of the models stay the same, but the way they operate changes. For instance, with the supply chain model, when the discrete event simulator model changes from a weekly step size to a daily one and the LP model

still runs every week, the module definitions stay the same, but the transformations and control will change to accommodate this.

4.2 Module Schema

The module component defined in KIB_Modules.xsd does two things: it ties an interface to a model name, and it defines a set of modules that belong to the model. Figures 25 and 26 show the graphical representation of the KIB_Modules.xsd schema.

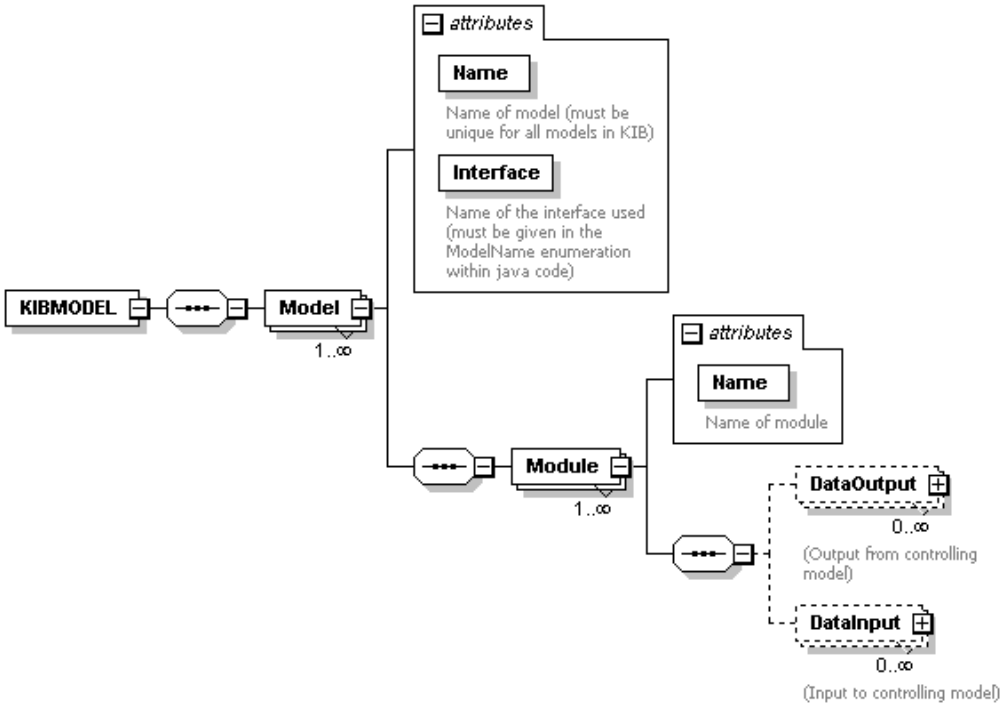

Figure 25. KIB_Modules.xsd Schema Graphic Representation Level 1

4.2.1 Model Element

Under the Model element, there are two attributes named 'Name' and 'Interface'. The definition for 'Name' is a distinct name for the model within the KIB instance. This name is used as a referencing name to this model for the transformations definition. This name is also sent to the model interface in order to open the correct model. The definition for 'Interface' must be one of the enumerated values set

within code. Currently, this includes "DEVS," "LP," and "ISM." This maps the distinct model name to an interface definition within code. Providing a unique model name for the interface allows a developer to create multiple models under the same interface. It also helps with labeling each component in the KIB.

4.2.2 Module Element

Each model contains a set of modules. Under the Module element, there is a 'Name' attribute. The definition for 'Name' here is a distinct string name that references something within the model. This name is also referenced within the transformations definition. As Gary points out in his research, these modules can be seen as different things depending on the environment used (Gary 2008). Within DEVS, a module is closely tied to an atomic model component whereas within an LP the module does not hold much meaning.

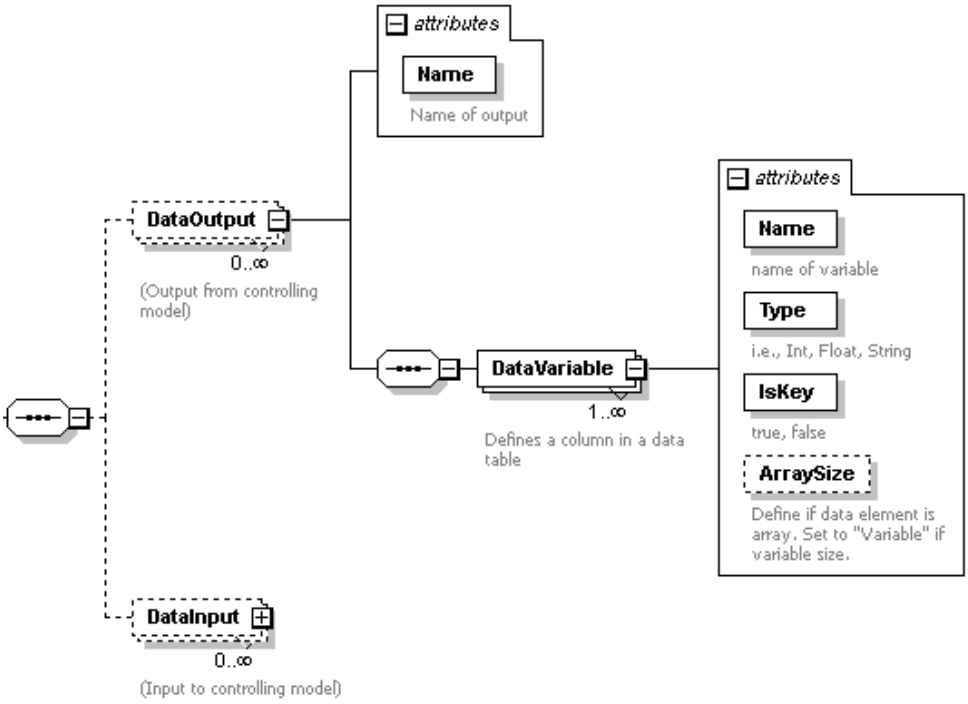

Figure 26. KIB_Modules.xsd Schema Graphic Representation Level 2

4.2.3 DataInput and DataOutput Elements

Each module has a set of input and output data lines associated with it. This would be the DataInput and DataOutput elements. Even though providing neither of these elements in a KIB XML would be semantically correct according to the schema, the module would serve no purpose. Therefore, a designer should always define at least one input or output. There is no difference between the schema definition of DataInput and DataOutput. The only difference is the way that they are treated within the KIB. A DataInput is used when data goes into the model and a DataOutput is used when data comes out of the model. Under the DataInput and DataOutput elements is a 'Name' attribute which uniquely identifies the port and is also associated with something within the model instance.

4.2.4 DataVariable Element

The type of data that is produced and consumed under a DataOutput or DataInput is defined as the DataVariable element. Each entry defines a column within a table. The DataVariable element contains the following attributes:

- **Name** – label for the variable. The name must be distinct for the DataInput or DataOutput group.
- **Type** - must be one of the following:
  - String
  - Int
  - Float
- **IsKey** – must either be "true" or "false" depending on if the variable is a key for the set. During runtime, there can never be two entries where all the key values are the same. This is similar to how a primary key set works in a

database. The newer set will overwrite the older set if all of the key values match. If no keys are set, the key is assumed to be 'null' for each incoming value and only the newest value set may be passed at each time step.

- **ArraySize** – is an optional value. This is set if the data variable is an array field and signifies the size of the array. This value must be a positive integer value greater than 0 or the string "Variable" if the size of the array is unknown.

4.3 Control Schema

The control schema defines frequency and the order of the transformation actions. Figure 27 shows the graphical representation of the KIB_Control.xsd schema.

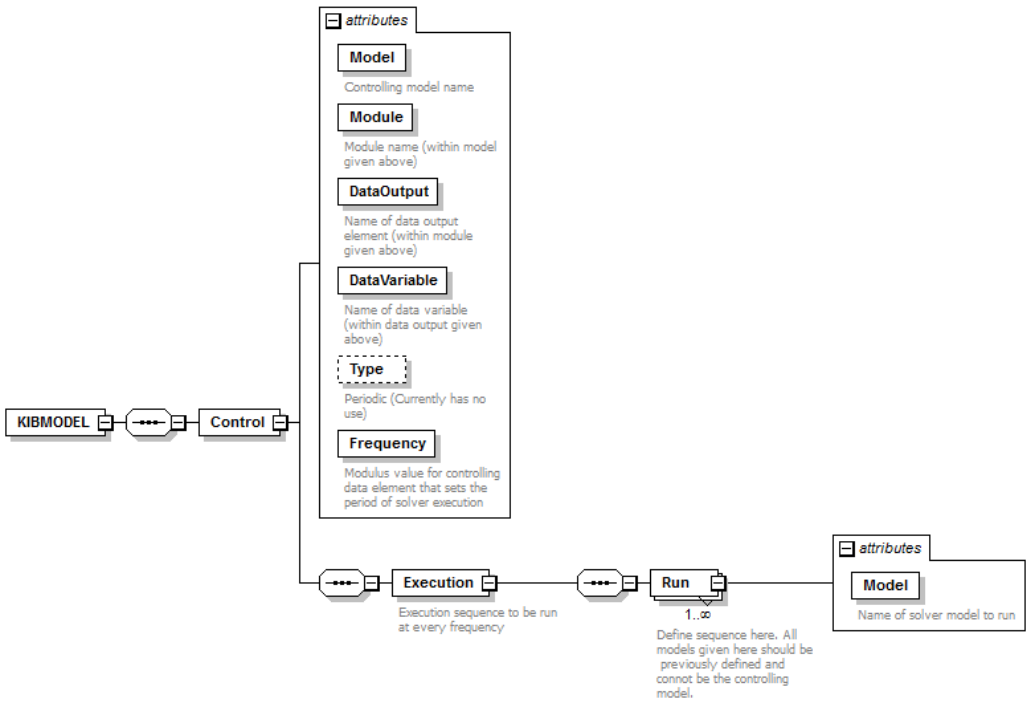

Figure 27. KIB_Control.xsd Schema Graphical Representation

4.3.1 Control Element

Within the Control element are the following attributes:

- Model
  - Defines the name of the model which contains the controlling variable

- Module
    - Defines the name of the module under the previously defined model which contains the controlling variable
- DataOutput
    - Defines the name of the data output under the previously defined module which contains the controlling variable
- DataVariable
    - Defines the name of the data variable under the previously defined data output which is the controlling integer variable
- Type
    - Can only be set to the string "Periodic" for the current version of the KIB. This signifies that transformations happen periodically. Future developments of the KIB may allow for other options.
- Frequency
    - Must be an integer value greater than 0 which defines how often the non-controlling model elements be executed – For instance, if this value is set to 2, execution will occur at instances 0, 2, 4, and so on until the model terminates.

4.3.2 Execution Element

A single Execution element must be defined under the Control element. The Execution element gives the order of execution models to run at the frequency instances. One or more Run elements must be defined under the Execution element and order of given elements is crucial. For each Run element that is defined, the Model attribute should give the name of the model to execute. The model name

must be previously given elsewhere within the KIB modules definitions and cannot be the model that is defined as the controlling model.

4.4 Relationship Schema

The relationship schema provides the structure and constraints of the transformation definitions. Figures 28, 29, 30, and 31 show the graphical representation of the KIB_Relationship.xsd schema.

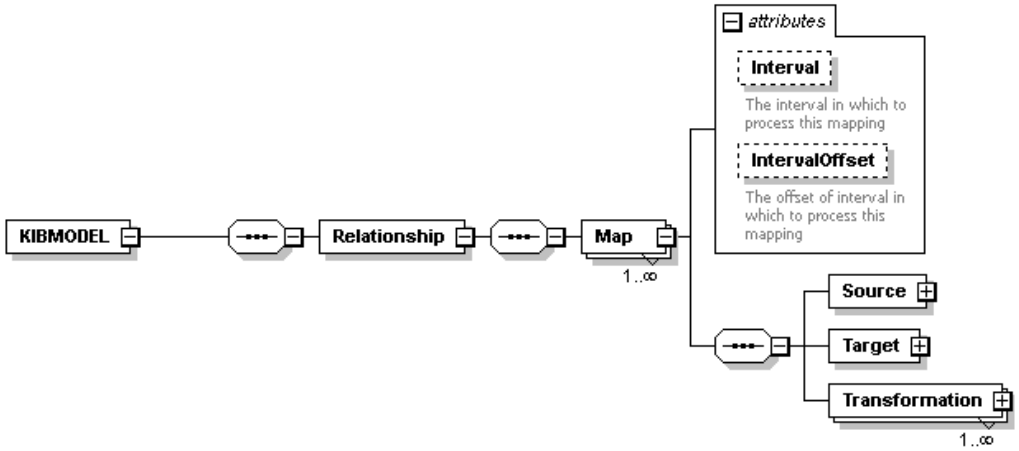

Figure 28. KIB_Relationship.xsd Schema Graphic Representation Level 1

4.4.1 Relationship Element

A single Relationship element signifies that this is an XML that defines the relationships of the KIB model.

4.4.2 Map Element

One or more Map elements must be defined under the Relationship element. The Map element defines a mapping between a DataOutput of one Module to a DataInput of another Module. The 'Interval' and 'IntervalOffset' attributes within the Map element define when all the transformations within a mapping should take place. These values are relative to the control frequency that is defined above in 4.3. The 'Interval' defines how often to execute this mapping transformation. This must be a positive, non-zero integer value. For example, if the frequency in the control is

set to 7 and the interval here is set to 2, the mapping will be executed on 0, 14, 28, and so on until the system terminates. The 'IntervalOffset' defines at what time the first transformation is executed. This must be a non-negative integer value. For example, taking the above case where frequency set to 7 and interval of mapping set to 2, if the interval offset is set to 5, the mapping will be executed on 5, 19, 33, and so on until the system terminates.

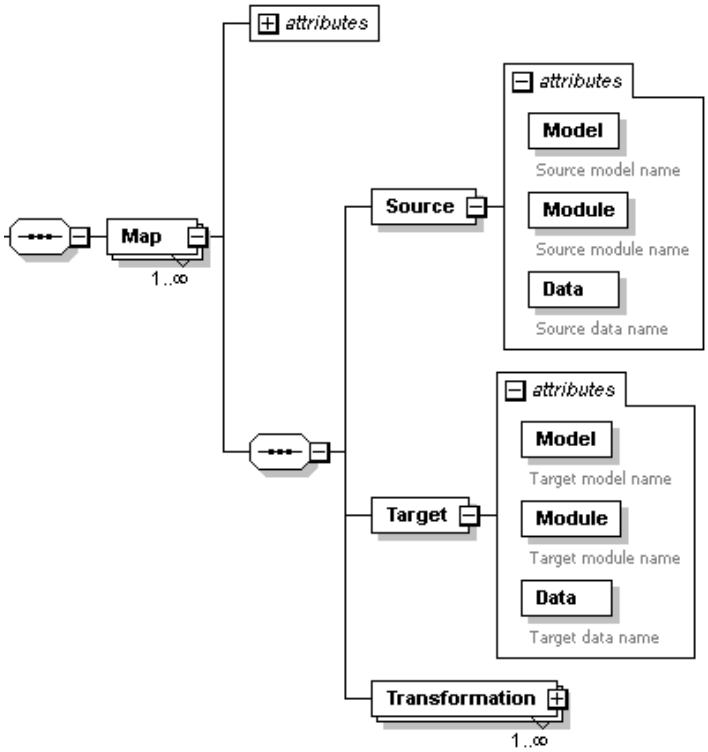

Figure 29. KIB_Relationship.xsd Schema Graphic Representation Level 2

4.4.3 Source DataOutput Element

Under each Map element is a Source element. The Source element contains the attributes 'Model', 'Module', and 'Data' which correspond to the names of a Model, Module, and DataOutput that has been previously defined. This makes up the address of a DataOutput element.

4.4.4 Target DataInput Element

Under each Map element is also a Target element. The definition of a Target element is the same as the Source element. The only distinction is that the 'Data' attribute references a DataInput element.

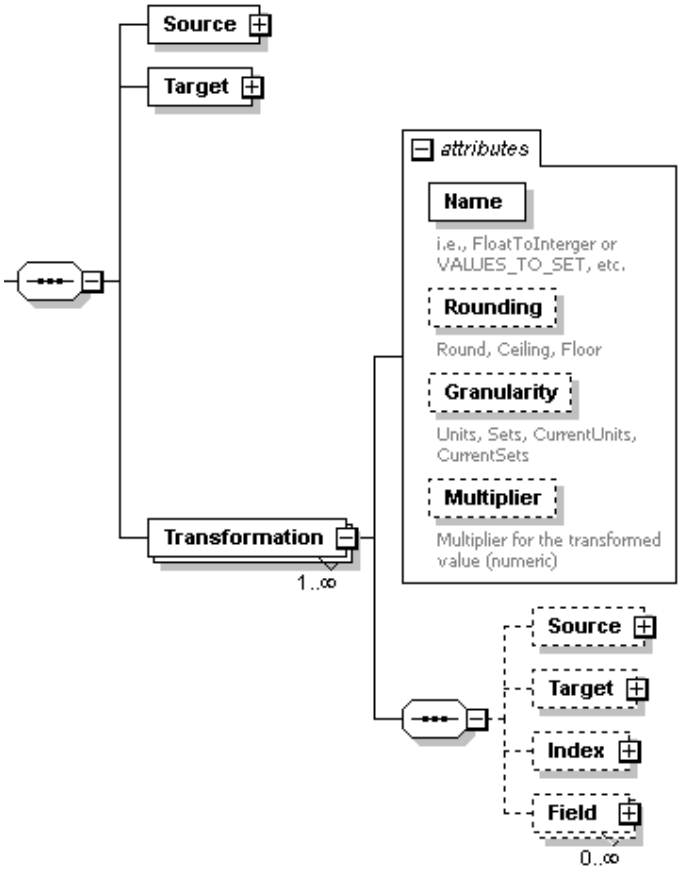

Figure 30. KIB_Relationship.xsd Schema Graphic Representation Level 3

4.4.5 Transformation Element

At least one Transformation definition must be given for each mapping. A Transformation element defines how data shall be transformed at the execution of this mapping within the KIB. The order of transformations does not make any different in the end result. Within the 'Name' attribute is an enumeration of transform types which corresponds to a name of a transformation within the KIB's

execution object. This name may be one of the names given in the list below. For a deeper explanation of each of these transformations, refer to APPENDIX B.

- NONE
- COPY
- IntegerToFloat
- FloatToInteger
- Aggregate
- MAX
- MIN
- MEDIAN
- MEAN
- NewestValue
- OldestValue
- SET_TO_VALUES
- VALUES_TO_SET
- FieldValueToVariable
- VariableToFieldValue
- ASSIGN_FIELD_VALUES
- DisaggregateIntoEqualBuckets
- AllToOneValue
- AllCurrentToOneValue

The attribute 'Rounding' defines how to round the data before it is transformed. If the source value is already an integer value, any of the rounding

functions will not affect the value in any one of these cases. The 'Rounding' attribute may be defined as one of the following:

- **Round**: Rounds a floating point value to the nearest integer value.
- **Ceiling**: Rounds a floating point value up to the next integer value.
- **Floor**: Rounds a floating point value down to the previous integer value.

The attribute 'Granularity' defines how data should be aggregated when a transformation takes a set of values and changes it to a single value or single set of values. It may be one of the following:

- **Units**: Data input is in terms of units, handled horizontally. For example, if 'MIN' transformation is used and the sets {5, 9, 0} {2, 2, 2} are sent from the source array, the target value will be 0; the smallest overall value.
- **Sets**: Data input is in terms of sets of data, handled vertically. For example, if 'MIN' transformation is used and the sets {5, 9, 0} and {2, 2, 2} are sent from the source array, the target value will be {2, 2, 0}; the minimum value of each index value individually.
- **CurrentUnits:** Same as 'Units,' but use only the most recent data.
- **CurrentSets**: Same as 'Sets,' but use only the most recent data.

The 'Multiplier' attribute defines a value in which should be used to multiply the target value by after transformation has been completed. This may be any real value. The multiplier value will only be used if the target value is numerical.

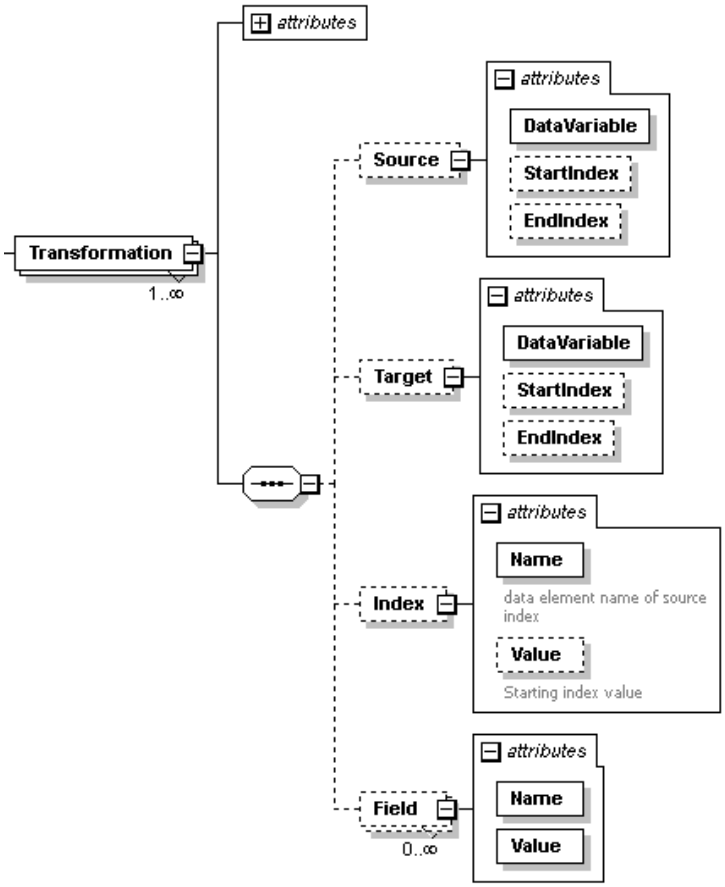

Figure 31. KIB_Relationship.xsd Schema Graphic Representation Level 4

4.4.6 Source DataVariable Element

Under each Transformation element a single Source element may be defined. This element is optional if the transformation does not require its definition. The 'DataVariable' attribute references a name of a DataVariable. This DataVariable must be defined under the Source DataOutput of the mapping. The 'StartIndex' and 'EndIndex' attributes are used only if the DataVariable referenced is an array type and a specific range needs to be selected for this transformation. The 'StartIndex' is constrained to a non-negative integer value and the 'EndIndex' is constrained to a positive, non-zero number. A StartIndex may be defined without an EndIndex if only a single value within an array is selected. If the EndIndex is defined, it must be greater than the StartIndex.

4.4.7 Target DataVariable Element

The Target element provides the same definition as the Source element within the schema. The only distinction is that the Target's DataVariable referenced the name of a DataVariable within the Target DataInput of the mapping.

4.4.8 Index Element

The Index element should define the DataVariable name on either the source or target that data should be indexed by. This is used in certain transformations when data is being transformed to or from an array. The value attribute must be an integer value greater than or equal to 0. This defines the starting index value (usually either 0 or 1).

4.4.9 Field Element

The Field element gives a set of key/value pairs used when a transformation requires it such as FieldValueToVariable and VariableToFieldValue.

5 SCHEMA IMPLEMENTATION

5.1 Object Structure and Data Structure Mapping

With the changes made to the schema as drawn out with the example in Figure 15, the object structure in code has been implemented with this structure as well. This provides a more succinct definition between the connection of XML structure and code structure.

5.1.1 KIB Entry Point

The diagram in Figure 32 shows a high level UML diagram of the entry point into the KIB. An instance of the KIBExecution object instantiates the KIBDataStore then sends the reference to ConfigurationReader where the KIBDataStore is filled with the appropriate metadata. This structure did not change since the previous version of the KIB. The ConfigurationReader has been updated to follow the new schema design.

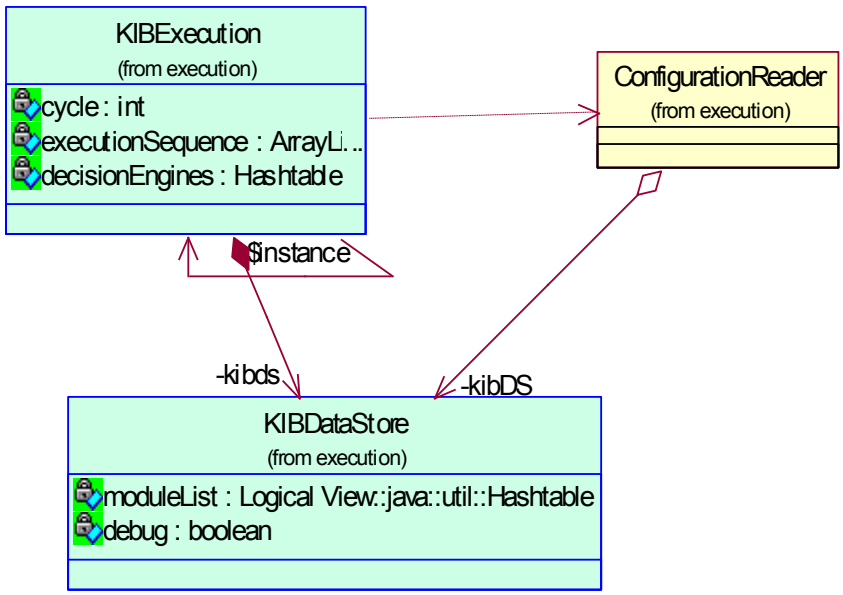

Figure 32. UML KIB Entry Point Objects

5.1.2 KIB Module Objects

Since now, from the schema design modules belong to model elements, this matches in the module metadata elements within the object design. The diagram in Figure 33 shows the upper level of the module objects. The KIBDataStore maps each model to a list of KIBModules. The ModelName object connects a string name to a value in the interface enumeration. Now each instance of a KIBModule belongs to a single model. The KIBModule contains a list of DataModelNodes which correspond to all DataInputs and DataOutputs associated with the Module in the schema. If the KIBModule instance is a target, it will contain a list of all DataRelationship objects for which this module is a target.

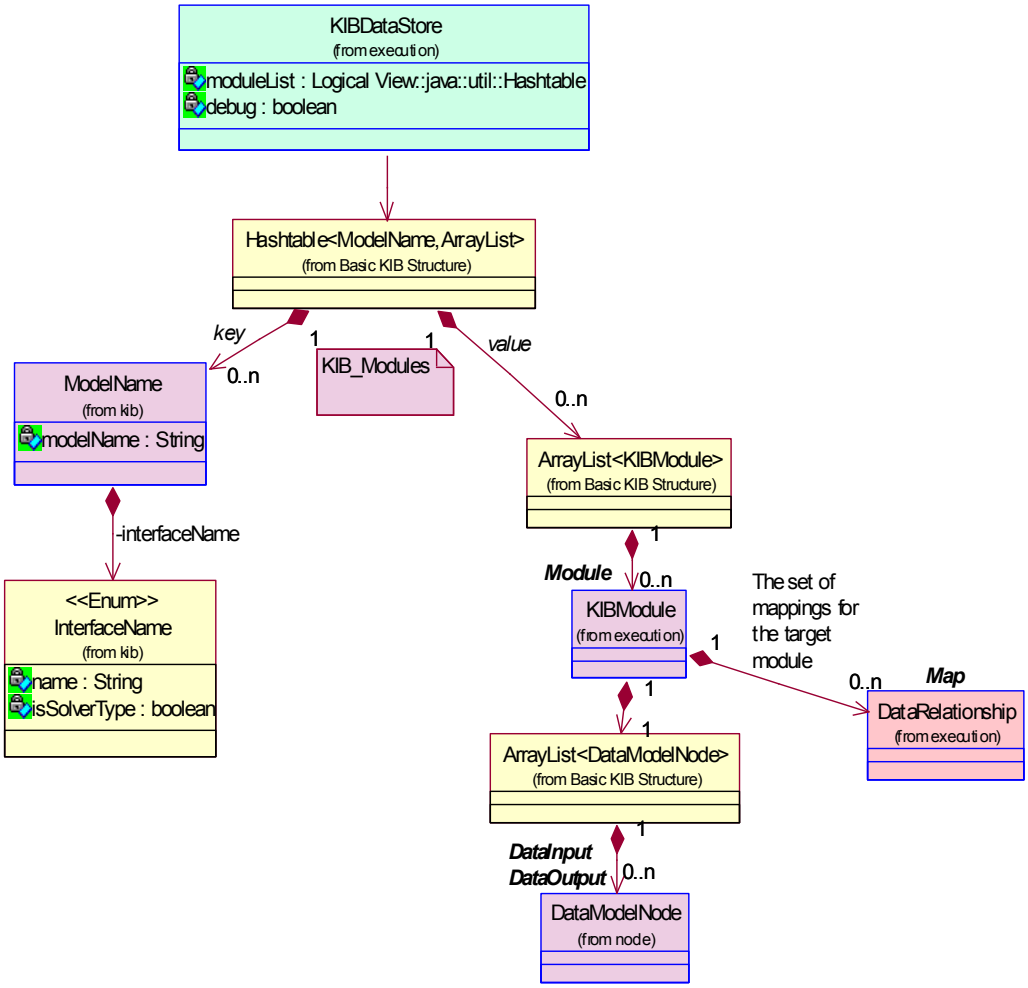

Figure 33. UML Objects Relating to KIB_Modules.xsd Part 1

Figure 34 shows the UML object diagram of the second level of the module objects. The only addition to this structure since the previous version is the DataType enumeration which enumerates String, Int, or Float and holds their string representations. Each DataVariable maps into an instance of NameTypeValue. The RecordDefinition object keeps the list of all NameTypeValues and marks the variables that are key values. As data is entered during runtime, instances of DataRecord are created with the value entries.

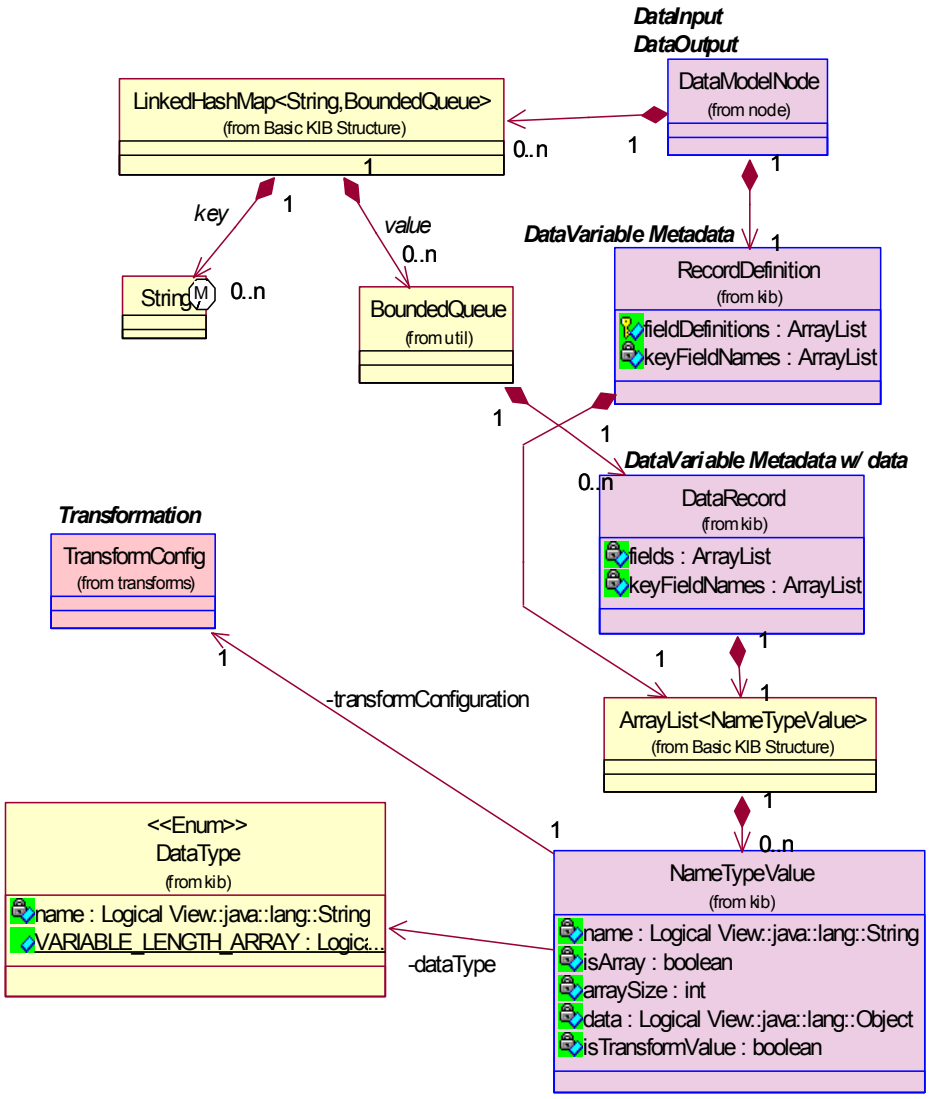

Figure 34. UML Objects Relating to KIB_Modules.xsd Part 2

5.1.3 KIB Control Object

The ControlConfiguration object shown in Figure 35 has a direct mapping to the data given in the KIB_Control.xsd schema. This object's structure does not differ from the previous version. The main difference is that the executionSequence is filled and executed in the given order which was previously ignored.

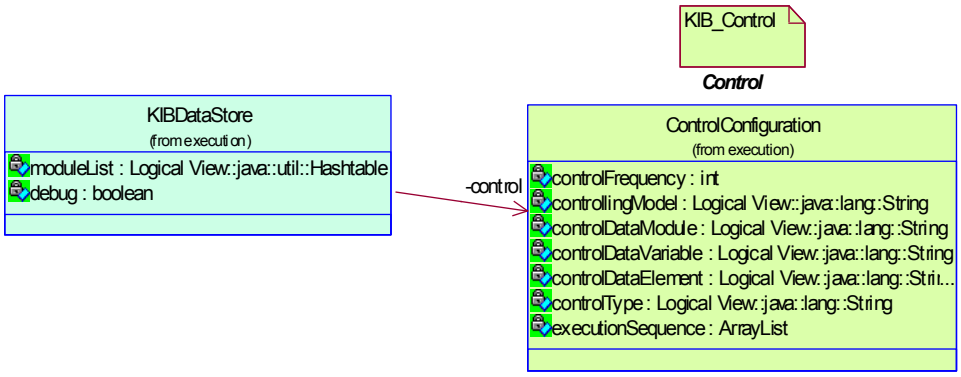

Figure 35. UML Object Relating to KIB_Control.xsd

5.1.4 KIB Relationship Objects

Figure 36 shows a UML diagram of the objects that map to the data in the KIB_Relationship.xsd schema. A ModelRelation object has been added to map the source model to the target model. Enumerations for granularity, rounding, and transformation name have been added to map string definitions within XML to flags in code. Definitions of these enumerations match what is given in 4.4.5.

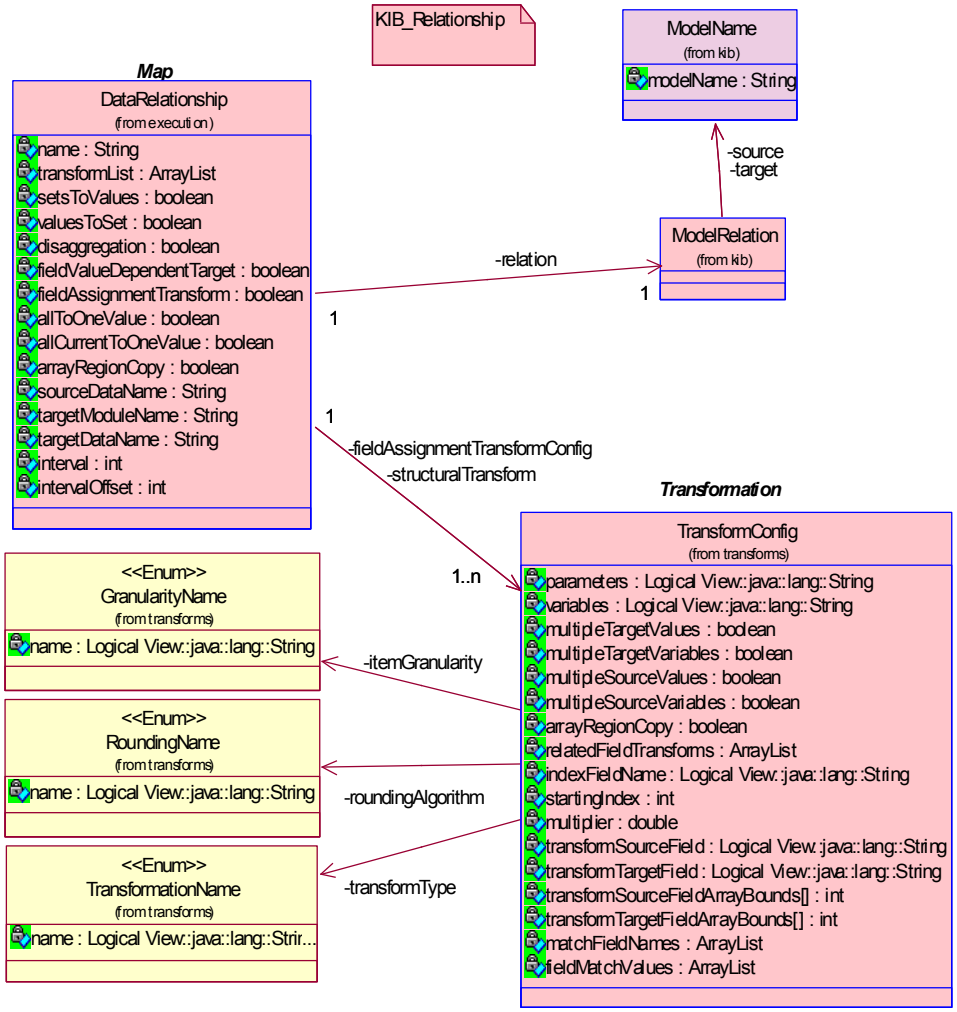

Figure 36. UML Objects Relating to KIB_Relationship.xsd

5.1.5 Adding an Interface

The process to add a new interface was made to be as simple as possible. When adding a new interface for the KIB, the following steps need to be taken:

1. Create an interface model node extending the DataModelNode class. This step is optional if no extensions of the DataModelNode are necessary for the model. This can be seen, for example, with the addition of the ISM model.

2. Create a KIB interface object implementing DecisionEngineInterface; define all required operations using a DataModelNode object either as defined in step 1 or the base DataModelNode itself
3. Create a name that will represent the interface and add it to the enumeration InterfaceName (refer to Figure 37)

```
public enum InterfaceName {
    LP("LP"), OPC("OPC"), ISM("ISM"), DEVS("DEVS", false);
```

Figure 37. InterfaceName Name Definitions

4. If a new interface model node has been created, have the ConfigurationReader.addVariable() function instantiate the new DataModelNode from step 1 with the given InterfaceName from step 3 (refer to Figure 38)

```
/*
 * Instantiate appropriate data model node here
 */
if (solver.getInterfaceName() == InterfaceName.LP)
    dmn = new LPDataModelNode(variableName, type, rd,
            kibDS.getHistorySize());
else if (solver.getInterfaceName() == InterfaceName.DEVS)
    dmn = new DEVSDataModelNode(variableName, type, rd,
            kibDS.getHistorySize());
else if (solver.getInterfaceName() == InterfaceName.OPC)
    dmn = new OPCDataModelNode(variableName, type, rd,
            kibDS.getHistorySize());
else
    dmn = new DataModelNode(variableName, type, rd,
            kibDS.getHistorySize());
```

Figure 38. Instantiating DataModelNode

5. Have the KIBExecution.initializeEngine() function instantiate the new DecisionEngineInterface from step 2 with the given InterfaceName from step 3

```
/*
 * Instantiate solver
 */
if (model.getInterfaceName() == InterfaceName.LP) {
    decisionInterface = new LPInterface(decisionModelPath,model);
} else if (model.getInterfaceName() == InterfaceName.OPC) {
    decisionInterface = new OpcHoneywellAdapter(
            decisionModelPath, model);
    decisionInterface.initializeEngine(kibds);
} else if (model.getInterfaceName() == InterfaceName.ISM) {
    decisionInterface = new ISMInterface(model);
    decisionInterface.initializeEngine(kibds);
}
```

Figure 39. Instantiating DecisionEngineInterface

5.1.6 Designing a KIB Model

Steps in a certain order should be taken in order to develop a KIB model. In general, following the following steps will lead to a working KIB model:

1. Create all necessary interfaces as described in 5.1.5
2. Implement and test each model separately to make sure each model component is formalized properly
3. For each model, create a separate XML file that implements the KIB_Modules.xsd schema defining things in a way that goes in line with how the interface for each model is designed from step 1
    - A separate file is not necessary to define each model contents, but this helps partition the components in succinct way. For smaller models, it may be reasonable for all modules to be defined in one file.
4. Decide where the controlling DataVariable resides and define them as such in an XML file that implements KIB_Control.xsd
5. Decide the data couplings and transformation scheme for all data and define as such in an XML file that implements KIB_Relationship.xsd
6. Reference all XML definitions from steps 3, 4, and 5 into an XML file that implements KIB_Paths.xsd

7. In code, load the file defined in step 6 into the ConfigurationReader object which will setup a KIBDataStore

5.2 ISM Implementation

To completely separate the ISM component from the rest of the project, Java's Remote Method Invocation (RMI) technology is used. RMI is a way to setup a server/client connection using Java interfaces as if the implementations for the interfaces reside on the client system. Once a server's method is called, the required attribute values are sent to the server where the function is executed remotely. The return value of the function is then sent back to the client if one exists. The definition of any complex structure that is being used in the communication must reside on both the client and server. The diagram in Figure 40 shows a block level communication between the KIB and each model. In this setup, the ISM is the RMI server and the ISM interface within the KIB is an RMI client. An interface package is also created that contains the RMI interface and the object, ISMResult, which contains the result of the ISM computation for a time step.

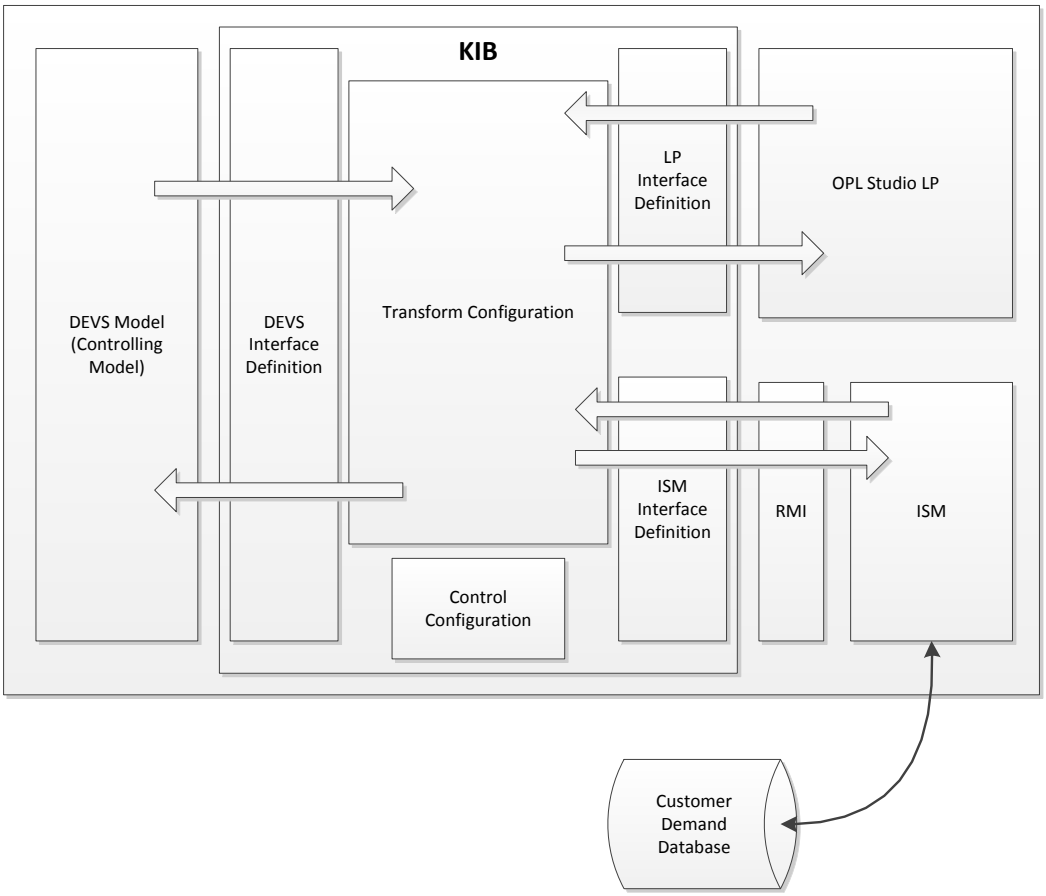

Figure 40. KIB with ISM

The modularized ISM model has a static set of variables associated with it. Below are a list of these elements; the first level being the modules, second level being either data input or data output, and third level being the set of data variables for the data input/output. A short explanation is given for each data variable.

- ISM_TARGET
    - FC_CD (O)
        - echelon_index (Int): Index for the selected echelon
        - hub (String): Inventory element name
        - product (String): Product name
        - quantity[] (Int): Values of forecasted customer demand for the current period up to the length of the planning horizon

- HUB_SS (O)
    - echelon_index (Int)
    - hub (String)
    - product (String)
    - quantity[] (Int): Values of safety stock in the hub inventories for the current period up to the length of the planning horizon
- CW_SS (O)
    - echelon_index (Int)
    - hub (String)
    - product (String)
    - quantity[] (Int): Values of safety stock in the component warehouse inventories for the current period up to the length of the planning horizon
- LOG_SS_FC (O)
    - echelon_index (Int)
    - hub (String)
    - product (String)
    - quantity [] (Int): Values of forecasted customer demand for the current period up to the length of the planning horizon
    - target (Int): target order up to
    - destination (Int): safety stock
    - value (String): week label
    - current_time (Float): clock time

        - weight (Int): weight for the week (0 or 1)

- ISM_INIT
    - ISM_INIT_DATA (I)
        - name (String): Name of the key element for initialization data
        - value (String): Value of the value element for initialization data
    - HUB_LIST (I)
        - echelon_index (Int)
        - hub[] (String): List of hub names for the echelon (names may exists more than once)
    - PRODUCT_LIST (I)
        - echelon_index (Int)
        - product[] (String): For each hub in the HUB_LIST input, this list gives an accompanying product creating (hub, product) pairs
    - TO_INVENTORY_LIST (I)
        - echelon_index (Int)
        - hub (String)
        - destination[] (String): List of destination inventory elements in the downstream echelon that this inventory ships to
    - TO_SHIP_TIME_LIST (I)
        - echelon_index (Int)
        - hub (String)

        - value[] (Int): Shipping time for each of the lanes given in TO_INVENTORY_LIST

- ISM_RUN
    - BOWK (I)
        - name (String): This key variable is not used in the current implementation of ISM
        - value (Int): The Beginning Of Week index
    - BOH (I)
        - echelon_index (Int)
        - hub (String)
        - product (String)
        - quantity (Int): Value for the Beginning On Hand value for the product in the inventory given in the key
    - INTRANSIT (I)
        - echelon_index (Int)
        - hub (String)
        - product (String)
        - quantity[] (Int): The in-transit values for the shipping for the product to the inventory given in the key
    - INTRANSIT_AO (I)
        - echelon_index (Int)
        - hub (String)
        - product (String)

        - quantity (Int): The value coming out of the shipping element for the product to the inventory given in the key

- Synchronization
    - ISM_SIM_SYNC (I)
        - current_time (Int): The current time value used for labeling

There are some potential issues with the KIB when it is expanded to allow three models communicating between each other. The KIB is designed to transform data between two models. Therefore, with three model communication, each transformation must be executed between pairs of models only. The diagram in Figure 41 shows a conceptual representation of three models communicating through the KIB. The execution and transformation are not only synchronous, but are also sequential, so the execution sequence of DEVS, ISM, LP is selected. This is because DEVS, being the controlling model in the KIB, needs to first send its state to both the ISM and LP. The LP is then dependent upon the computations done in the ISM. Since the LP and ISM are both point solution models, this execution sequence is simplified. For KIB solutions with models that aren't executed sequentially and/or contain multiple models that aren't point solutions, the control may need to be redesigned to allow a more complex environment. Concepts and methods for such control were developed in a dissertation by Dongping to allow asynchronous execution between DEVS and MPC (Huang 2008).

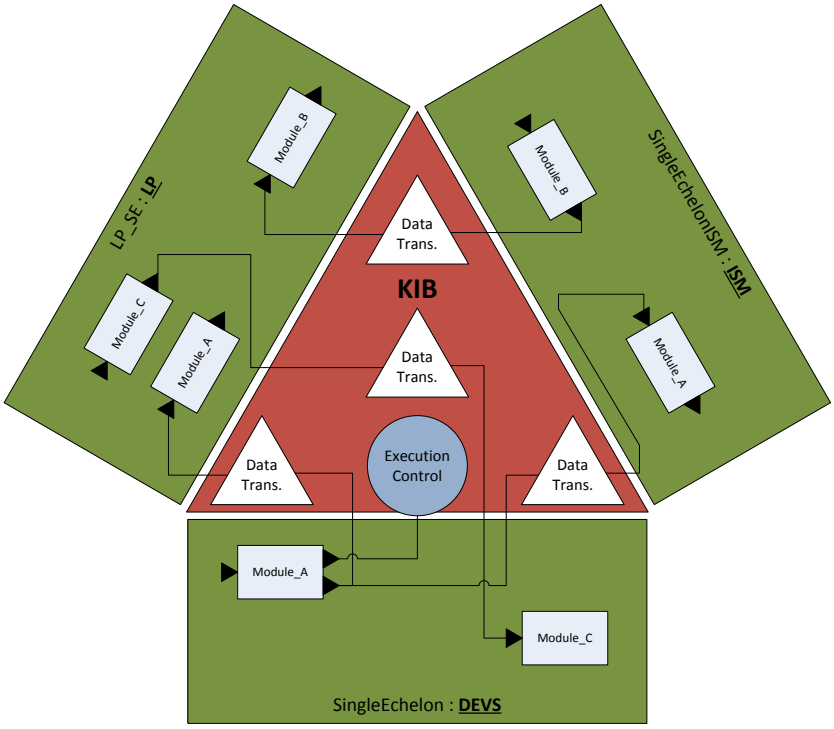

Figure 41. Three Model KIB Communications

A block diagram showing the communication between the three models of the OSF platform is given in Figure 42. This gives a high level view of the data that is being sent between the models. DEVS sends its state to the LP and the week index to the ISM. The ISM uses the week index to look up the correct Actual Customer Demand (ACD), Historic Forecast (HFC), and Forecast Customer Demand (FCCD) for the correct period. ISM then computes a Safety Stock (SS) value and then sends this along with the FCCD data to the LP. The ISM also sends some data back to DEVS (not explicitly depicted in the figure) to be used strictly by the transducers for data collection. LP then computes a plan for a period of time and sends this plan back to DEVS in the form of release commands. DEVS also uses Lot Generator (LG) to determine the amount of inventory to be generated in the most upstream inventory in order to replenish released inventory. The ACD data is used by the

customers in the DEVS model to determine if demand has been fulfilled at each period of time.

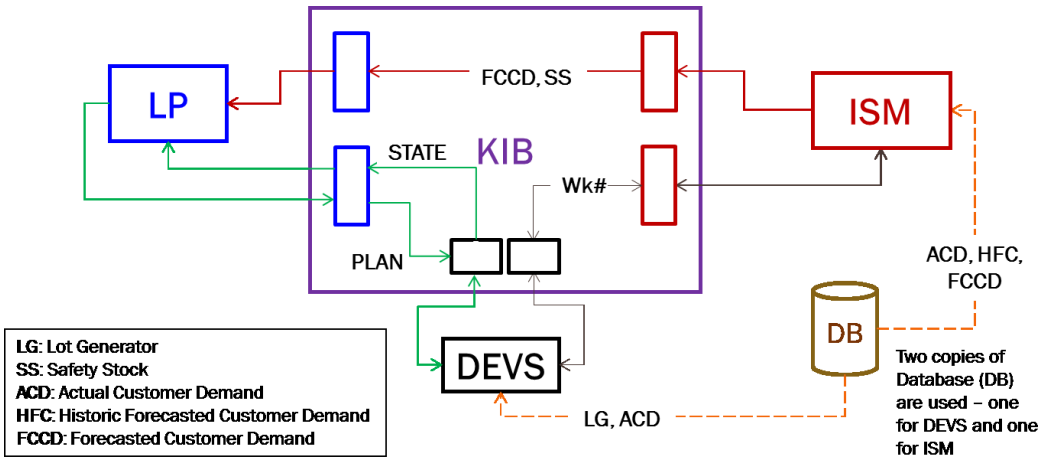

Figure 42. OSF Model

5.3 Single Echelon Implementation

5.3.1 Single Echelon Timeline

To properly map the execution and time base of the simulation with the execution and time base of the LP and ISM, the sample diagram in Figure 43 was created. This diagram represents a model with a single hub, shipping to a single component warehouse through one shipping lane. The following assumptions are also used.

- Model provides perfect fulfillment for demand
- The Hub and Component Warehouse models are set to process arrivals immediately
- Hub delivers all of its inventory to the Geo Customer at each time period

The simulation, ISM, and LP all execute on a weekly time step. The Decision Connector component contains the connection to the KIB. Single arrows downward represent product being sent to the next component. Double arrows upward are state messages being sent to the Decision Connector. Double arrows downward are

release messages to be delivered to their respective components. The following constraints are set:

$T = \mathbb{R}_{[0,\infty]}$

$T' = \mathbb{Z}_{[0,\infty)}$

$t \in T$ (Time in DEVS is in real increments)

$t' \in T'$ (Time used for KIB is on integral steps)

$\Delta s \in \mathbb{Z}_{[0,\infty)}$ (Shipping time from CW to hub)

$R(t') = \mathbb{Z}_{[0,\infty)}$ (Release command function at the CW)

$$R(x) = 0 \text{ where } x < 0$$

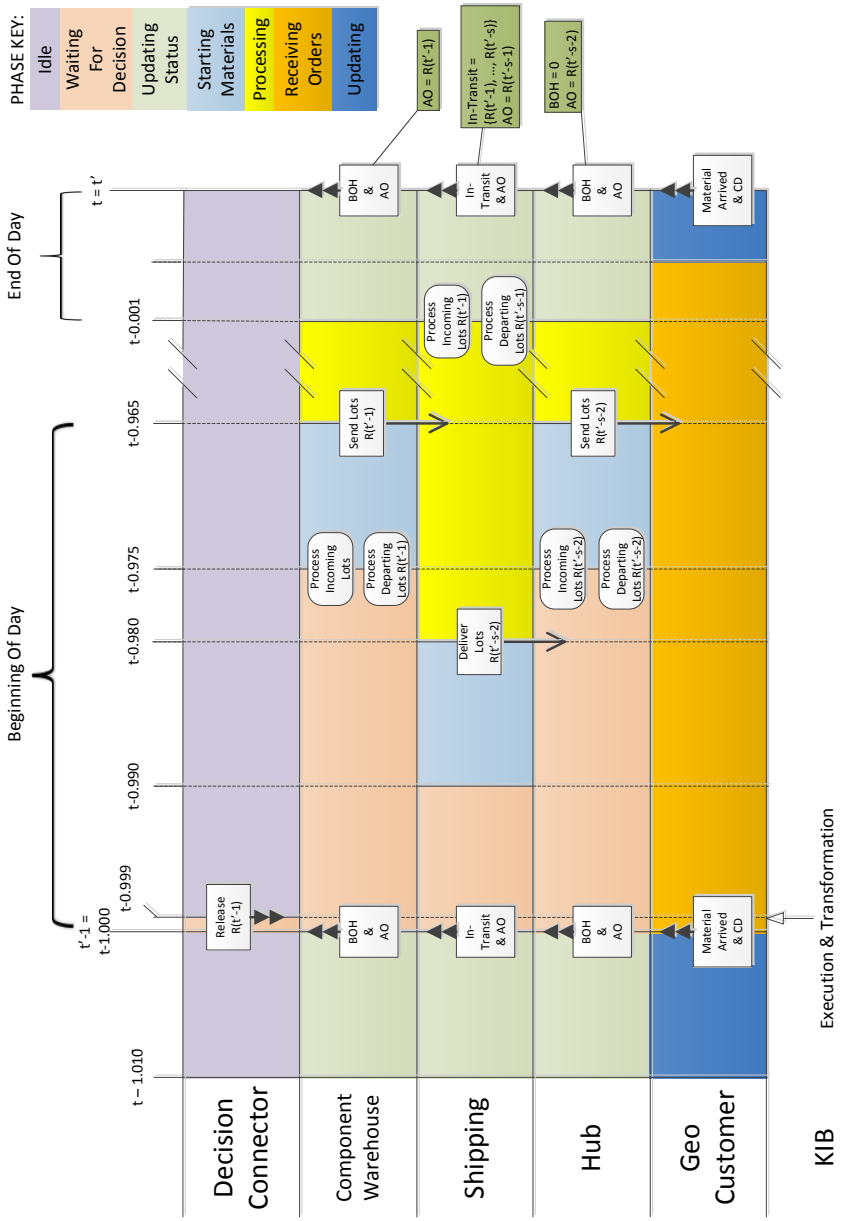

Figure 43. Single Echelon Timeline

To the simulation, the execution of each week is not an instantaneous event. A set of events occur over a period of time. The model was implemented to execute certain things at certain periods of time in order to force a deterministic execution order within DEVS. Thus, the simulation has varying states throughout each time step. The ISM and LP only care about the state of the system at the integer time values as they only execute over integer granularity. If this was expanded to a setup

where the simulation ran on a daily time step with ISM and LP running weekly, the ISM and LP would only care about the state of the system at times {0, 7, 14, …, n}.

5.3.2 Configuration and GUI

A configuration schema and GUI was created in order to quickly run experiments on the single echelon model. The ISM Client schema in Figure 44 and the accompanying GUI in Figure 45 provide entries for connecting to the ISM server. The ISM client configuration consists of the following information:

- Hostname

  The hostname is the name of the host for which the ISM server resides. This can be in the form of an IP address, a URL, or the string "localhost" if the server resides on the local host machine. The elements useLocalHost and useMyIP can be set to true or false. If useLocalHost is set to true, the string "localhost" is used. If useMyIP is set to true, the IP address of the local machine is retrieved and used. Otherwise, a host name string can be given under the name element. Only one of these three entries should be given.

- Port

  This is the port number, as an integer, that the ISM server is setup on. The default port for the ISM server is 2020.

- Create Server and Server Path

  If the createServer flag is set to true, the executable jar file at the serverPath is executed to start the server with the given hostname and port arguments. If the createServer flag is set to false, the serverPath is ignored.

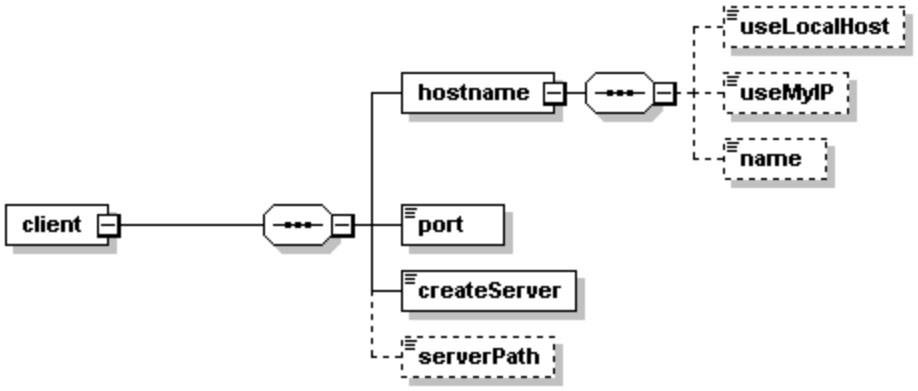

Figure 44. ISM Client Schema

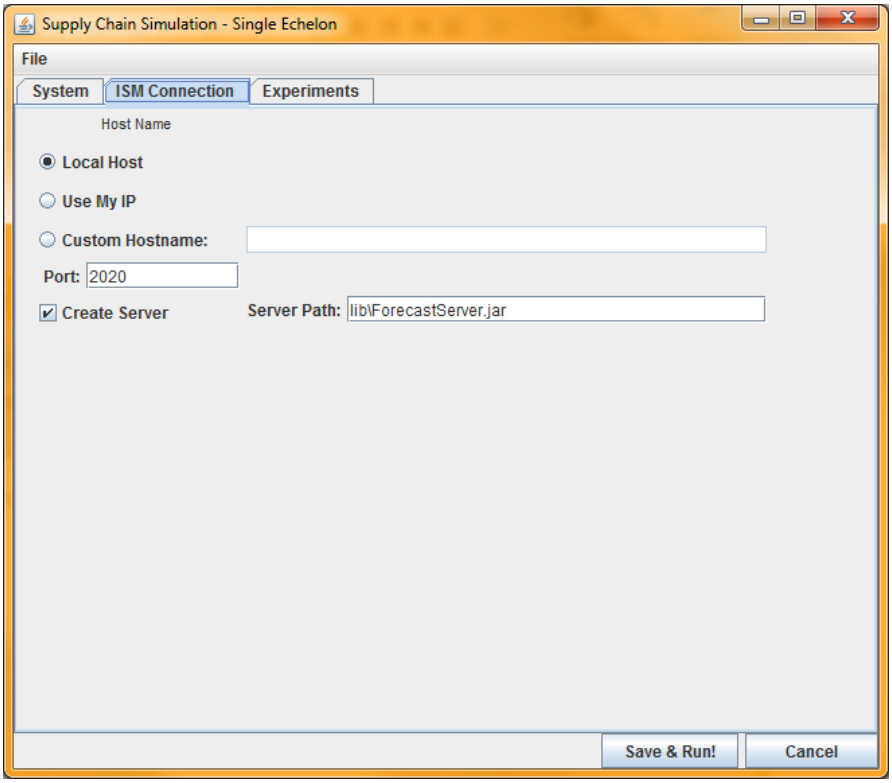

Figure 45. Single Echelon GUI: ISM Connection Tab

The system schema in Figure 46 and accompanying GUI in Figure 47 provide the system entries. This should hold any information that need to be sent to the system as a whole. The system configuration contains the following data:

- configPath

The configuration path is the path to the directory containing "KIB" and "LP_Models" folders where the KIB and LP models are stored.

- dataDir

The data directory is the path to the directory containing the input customer demand data.

- outDir

The output directory is the path to the directory where the resulting data from the model run should be stored.

- stepSize

This should be set to "Weekly" or "Daily" depending on the step size granularity for the simulation model.

- lpstepSize

This should be set to "Weekly" or "Daily" depending on the step size granularity for the optimization model.

- hubs

The hubs element should contain a list of hubs that can be selected to run in the experiments.

- products

The products element should contain a list of products that can be selected to run in the experiments.

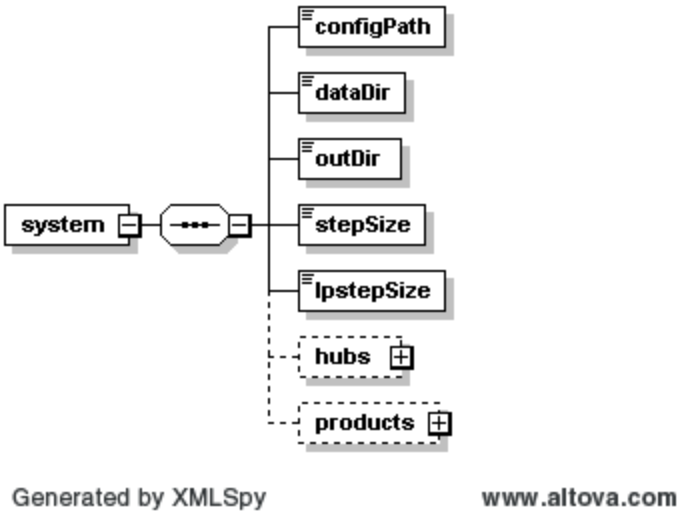

Figure 46. System Schema

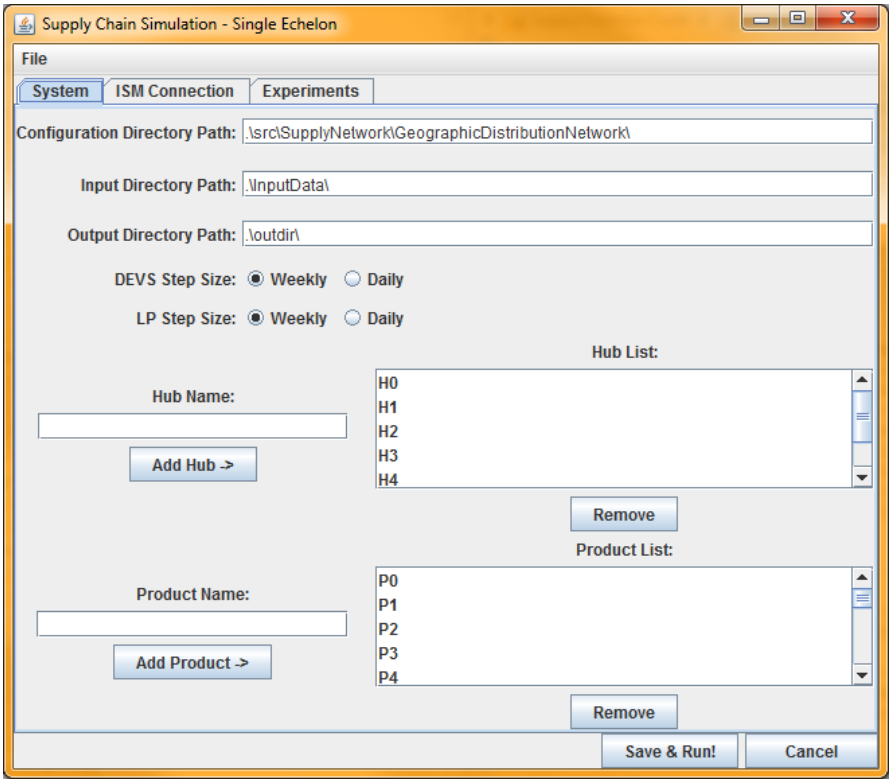

Figure 47. Single Echelon GUI: System Tab

For this project, the scope for what is known as an experiment configuration is any data that is inputted to select certain functionality for a model to run. This should not affect the overall structure of the model. The experiment schema in Figure 48 and accompanying GUI in Figure 49 provide the experiment

configuration. This holds a list of experiment configuration settings to run. Each experiment element in the experiment configuration should contain the following information:

- name

  This element defines the name of the experiment used to label the data.

- products

  This element defines a list of products to run in this experiment.

- hubs

  This element defines a list of hubs to run in this experiment.

- startingWeek

  This element defines a string representation of the week for which the model is initialized to.

- endingWeek

  This element defines a string representation of the week when the model should terminate.

- smoothing

  This element defines a list of smoothing names to be used. Each smoothing selection is run with each service level (below).

- serviceLevel

  This element defines a list of service level values (between 0 and 100) that should be used; run one at a time with each smoothing name (above).

- planningHorizon

This element defines how many weeks into the future to plan for in the ISM.

This value must be greater than 0.

- historySize

This element defines how many weeks to smooth with historic data in the ISM.

This value must be greater than 0.

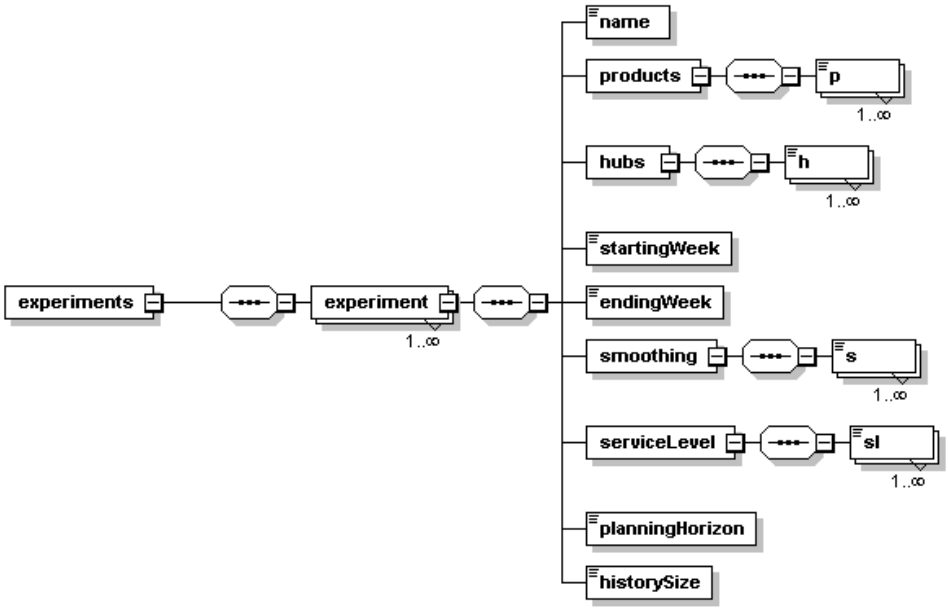

Figure 48. Experiment Schema

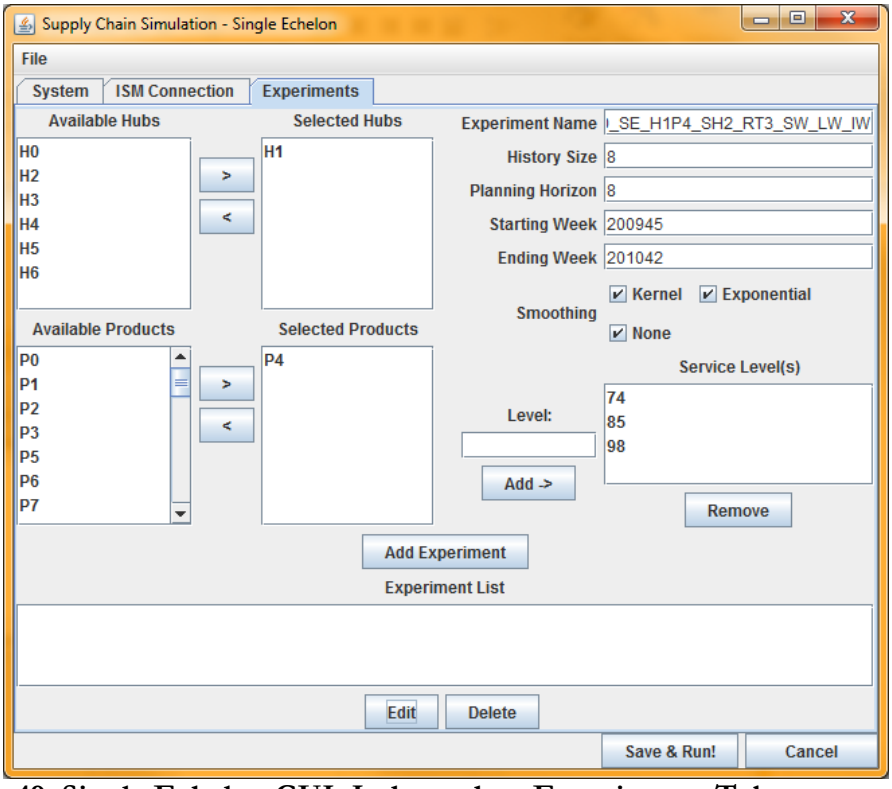

Figure 49. Single Echelon GUI: Independent Experiments Tab

5.3.3 KIB Implementation

For each component, the granularity of daily or weekly has been considered. The LP in the real world runs on a weekly basis, but could be run on a daily basis even though it may not produce any meaningful results. The simulation in the past only ran on a weekly basis, but a daily basis would better match real world operations. The configuration with the simulation running on a weekly basis with the LP running on a daily basis should be ignored since is does not make sense to optimize faster than the simulation can run. The data provided for customer demand and forecast is given in weekly granularity. Therefore, running the ISM on a daily granularity with this data would not provide any better results. With these constraints defined, we have the following 3 configurations of running granularity:

A. Simulation: weekly, LP: weekly, ISM: weekly

B. Simulation: daily, LP: weekly, ISM: weekly [closest to real world]

C. Simulation: daily, LP: daily, ISM: weekly

For each of these configurations, the KIB needs to be setup in a different way. Configuration A requires a 1:1:1 execution scheme; the LP and ISM execute once for every step in the simulation. Configuration B requires a 7:1:1 execution scheme; the LP and ISM execute once for every 7 steps in the simulation. Configuration B also requires disaggregation transformations. Configuration C will be ignored for this implementation since it may or may not produce meaningful results. The KIB model has been partitioned with the following xml files:

1. Instances of KIB_Paths
    a. SingleEchelon_Sim[W]_LP[W]_ISM[W].xml
    b. SingleEchelon_Sim[D]_LP[W]_ISM[W].xml
2. Instances of KIB_Modules
    a. DEVS_Modules.xml
    b. ISM_Modules.xml
    c. LP_Modules.xml
3. Instances of KIB_Control
    a. Control_Freq1.xml
    b. Control_Freq7.xml
4. Instances of KIB_Relationship
    a. ISM_Relationships.xml
    b. Relationship_Sim[W]_LP[W]_ISM[W]
    c. Relationship_Sim[D]_LP[W]_ISM[W]

To load the KIB for configuration A, the file SingleEchelon_Sim[W]_LP[W]_ISM[W].xml is called. This loads the module definitions in all of 2, the control 3.a, and the relationships defined in 4.a and 4.b. To load the KIB for configuration B, the file SingleEchelon_Sim[D]_LP[W]_ISM[W].xml is called. This loads the module definitions in all of 2; the control 3.b; and the relationships defined in 4.a and 4.c. See the XML code in Figure 50 for the definition of this XML file. For either of these KIB models, all of the defined instances of KIB_Modules are called since the structure does not change between configurations. All of the relationships going to or from the ISM do not change per configuration, so it was a design choice to use the same definition between the two as well.

```
<?xml version="1.0" encoding="UTF-8"?>
<KIBPATHS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="KIB_Paths.xsd">
        <MODULE_FILE Path="DEVS_Modules.xml"/>
        <MODULE_FILE Path="LP_Modules.xml"/>
        <MODULE_FILE Path="ISM_Modules.xml"/>

        <RELATIONSHIP_FILE Path="Relationship_Sim[D]_LP[W]_ISM[W].xml"/>
        <RELATIONSHIP_FILE Path="ISM_Relationships.xml"/>

        <CONTROL_FILE Path="Control_Freq7.xml"/>
</KIBPATHS>
```

Figure 50. Path Definitions for KIB

The XML code in Figure 51 shows the definition of the module H1 for the DEVS side of the KIB. The H1 module directly maps to the atomic inventory component with the name "H1." The H1 inventory model creates a BOH message which contains the amount of product that is left in the inventory from the previous step. An input to this inventory model is a release message. When the model receives a release message, it will release the given amount of inventory.

```
<?xml version="1.0" encoding="UTF-8"?>
<KIBMODEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="KIB_Modules.xsd">
        <Model Interface="DEVS" Name="SingleEchelon">
                .
                .
                .
                <Module Name="H1">
                        <DataOutput Name="BOH">
                                <DataVariable Name="hub" Type="String" IsKey="true"/>
                                <DataVariable Name="product" Type="String" IsKey="true"/>
                                <DataVariable Name="Quantity" Type="Int" IsKey="false"/>
                        </DataOutput>

                        <DataInput Name="RELEASE">
                                <DataVariable Name="product" Type="String" IsKey="true"/>
                                <DataVariable Name="source" Type="String" IsKey="true"/>
                                <DataVariable Name="destination" Type="String" IsKey="true"/>
                                <DataVariable Name="Quantity" Type="Int" IsKey="false"/>
                        </DataInput>
                </Module>
                .
                .
                .
        </Model>
</KIBMODEL>
```

Figure 51. DEVS Modules H1 KIB Definition

The XML code in Figure 52 shows the definition of H1 and HX modules on the LP side of the KIB. Modules do not mean anything to the LP, so the definitions of modules here are only symbolic to the KIB itself. The H_BOH input under the HX module aggregates all the data from every hub into a single input. The H_RELEASE output under the H1 module contains release messages for every hub that will later need to be filtered within the mapping definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<KIBMODEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="KIB_Modules.xsd">
        <Model Interface="LP" Name="LP_SE">
                .
                .
                .
                <Module Name="HX">
                        <DataInput Name="H_BOH">
                                <DataVariable Name="hub" Type="String" IsKey="true"/>
                                <DataVariable Name="product" Type="String" IsKey="true"/>
                                <DataVariable Name="quantity" Type="Int" IsKey="false"/>
                        </DataInput>
                </Module>

                <Module Name = "H1">
                        <DataOutput Name="H_RELEASE">
                                <DataVariable Name="product" Type="String" IsKey="true"/>
                                <DataVariable Name="source" Type="String" IsKey="true"/>
                                <DataVariable Name="destination" Type="String" IsKey="true"/>
                                <DataVariable Name="period" Type="Int" IsKey="false"/>
                                <DataVariable Name="quantity" Type="Float" IsKey="false"/>
                        </DataOutput>
                </Module>
                .
                .
                .
        </Model>
</KIBMODEL>
```

Figure 52. LP Modules H1 and HX KIB Definition

The XML code in Figure 53 shows a section of the relationship definition for configuration B. This portion of the relationship definition gives the mappings for H1 and HX in the LP and DEVS modules. The first Map element defines the mapping between the BOH of H1 in DEVS to H_BOH of HX in LP. The transformation is set to NONE which means that all data is passed right through. Since the LP interface only uses the newest data, only the BOH data from the latest definition will be passed over. The second Map element defines the mapping for the release commands to H1 in DEVS. The FieldValueToVariable transformation removes any elements in the source DataOutput that doesn't match the source DataVariable with the value of "H1." The ASSIGN_FIELD_VALUES transformation sets all target DataVariables with the name destination to the value "GC_H1." This assumes that H1 in the simulation is attached to the customer with the name GC_H1. The final transformation, DisaggregateIntoEqualBuckets, will

take the quantity data from the source and distribute the value equally to all target time buckets. The divisor is determined by the frequency value in the control.

```
<?xml version="1.0" encoding="UTF-8"?>
<KIBMODEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="KIB_Relationship.xsd">
        <Relationship>
                .
                .
                .
                <Map>
                        <Source Model="SingleEchelon" Module="H1" Data="BOH"/>
                        <Target Model="LP_SE" Module="HX" Data="H_BOH"/>
                        <Transformation Name="NONE"/>
                </Map>

                <Map>
                        <Source Model="LP_SE" Module="H1" Data="H_RELEASE"/>
                        <Target Model="SingleEchelon" Module="H1" Data="RELEASE"/>
                        <Transformation Name="FieldValueToVariable" Rounding="Round">
                                <Source DataVariable="source"/>
                                <Target DataVariable="source"/>
                                <Field Name="source" Value="H1"/>
                        </Transformation>
                        <Transformation Name="ASSIGN_FIELD_VALUES">
                                <Source DataVariable="destination"/>
                                <Target DataVariable="destination"/>
                                <Field Name="destination" Value="GC_H1"/>
                        </Transformation>
                        <Transformation Name="DisaggregateIntoEqualBuckets" Rounding="Round">
                                <Source DataVariable="quantity"/>
                                <Target DataVariable="quantity"/>
                        </Transformation>
                </Map>
                .
                .
                .
        </Relationship>
</KIBMODEL>
```

Figure 53. KIB Relationship Mapping for H1

The XML code in Figure 54 shows the definition of the control for configuration B. This file defines the DataVariable named current_time in the DataOutput LP_SYNC in the Model SingleEchelon, previously defined as a DEVS model, as the controlling time value. The frequency is set to 7 which means that the models defined under execution will execute once for every 7 time steps in the SingleEchelon DEVS model. Under the execution element, it is defined that the model SupplyChainISM will be executed followed by the model LP_SE.

```
<?xml version="1.0" encoding="UTF-8"?>
<KIBMODEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="KIB_Control.xsd">
        <Control Frequency="7" Type="Periodic" DataVariable="current_time" DataOutput="LP_SYNC"
                Module="Synchronization" Model="SingleEchelon">
                <Execution>
                        <Run Model="SupplyChainISM"/>
                        <Run Model="LP_SE"/>
                </Execution>
        </Control>
</KIBMODEL>
```

Figure 54. KIB Control Definition for Single Echelon, 7:1:1

5.4 Multi-Echelon Implementation

To create an OSF implementation with multi-echelon support, a double-echelon LP model was created and the ISM was extended to handle multiple echelons. The ISM requires the model structure and state data from each time period in order to compute the appropriate data for the upper echelons.

5.4.1 KIB Implementation

The same configuration that was setup for the single-echelon is used for the multi-echelon instance that is labeled "DoubleEchelon." An echelon index was incorporated to address each echelon separately in the KIB. The XML code in Figure 55 shows the definition of the ISM with multiple-echelon support. Model structure data is passed to the ISM through TO_INVENTORY_LIST and TO_SHIP_TIME_LIST. Every upper echelon needs to have data about where product is shipped to in the lower echelon. State data is passed through BOH, INTRANSIT, and INTRANSIT_AO. Every upper echelon needs to know how much stock there is in the lower echelon in order to compute demand.

```
<?xml version="1.0" encoding="UTF-8"?>
<KIBMODEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="KIB_Modules.xsd">
        <Model Interface="ISM" Name="SupplyChainISM">
                .
                .
                .
                <Module Name="ISM_INIT">
                        <DataInput Name="ISM_INIT_DATA">
                                <DataVariable Name="name" Type="String" IsKey="true"/>
                                <DataVariable Name="Value" Type="String" IsKey="false"/>
                        </DataInput>
                        <DataInput Name="HUB_LIST">
                                <DataVariable Name="echelon_index" Type="Int" IsKey="true"/>
                                <DataVariable Name="hub" Type="String" IsKey="false" ArraySize="Variable" />
                        </DataInput>
                        <DataInput Name="PRODUCT_LIST">
                                <DataVariable Name="echelon_index" Type="Int" IsKey="true"/>
                                <DataVariable Name="product" Type="String" IsKey="false" ArraySize="Variable" />
                        </DataInput>
                        <DataInput Name="TO_INVENTORY_LIST">
                                <DataVariable Name="echelon_index" Type="Int" IsKey="true"/>
                                <DataVariable Name="hub" Type="String" IsKey="true"/>
                                <DataVariable Name="destination" Type="String" IsKey="false" ArraySize="Variable"/>
                        </DataInput>
                        <DataInput Name="TO_SHIP_TIME_LIST">
                                <DataVariable Name="echelon_index" Type="Int" IsKey="true"/>
                                <DataVariable Name="hub" Type="String" IsKey="true"/>
                                <DataVariable Name="value" Type="Int" IsKey="false" ArraySize="Variable"/>
                        </DataInput>
                </Module>
                <Module Name="ISM_RUN">
                        <DataInput Name="BOWK">
                                <DataVariable Name="name" Type="String" IsKey="true"/>
                                <DataVariable Name="Value" Type="Int" IsKey="false"/>
                        </DataInput>
                        <DataInput Name="BOH">
                                <DataVariable Name="echelon_index" Type="Int" IsKey="true"/>
                                <DataVariable Name="hub" Type="String" IsKey="true"/>
                                <DataVariable Name="product" Type="String" IsKey="true"/>
                                <DataVariable Name="Quantity" Type="Int" IsKey="false"/>
                        </DataInput>
                                <DataInput Name="INTRANSIT">
                                <DataVariable Name="echelon_index" Type="Int" IsKey="true"/>
                                <DataVariable Name="hub" Type="String" IsKey="true"/>
                                <DataVariable Name="product" Type="String" IsKey="true"/>
                                <DataVariable Name="Quantity" Type="Int" IsKey="false" ArraySize="Variable"/>
                        </DataInput>
                        <DataInput Name="INTRANSIT_AO">
                                <DataVariable Name="echelon_index" Type="Int" IsKey="true"/>
                                <DataVariable Name="hub" Type="String" IsKey="true"/>
                                <DataVariable Name="product" Type="String" IsKey="true"/>
                                <DataVariable Name="Quantity" Type="Int" IsKey="false"/>
                        </DataInput>
                </Module>
                .
                .
                .
        </Model>
</KIBMODEL>
```

Figure 55. Multi-Echelon ISM Modules

6 RESULTS

6.1 Regression Testing

A set of JUnit Tests were formulated for the previous version of the KIB. These unit tests covered all of the functionality of the KIB from definition to execution. In order to use these tests for the updated KIB, all of the XML definitions for the JUnit Tests were updated for the new XML schema. Where applicable, the JUnit Tests were also updated to accommodate the new Java structure. After running this updated set of unit tests, the result in Figure 56 was returned in the Eclipse IDE with 0 errors and 0 failures. A 100% pass shows that the new structure does not affect any of the KIB functionality from past revisions.

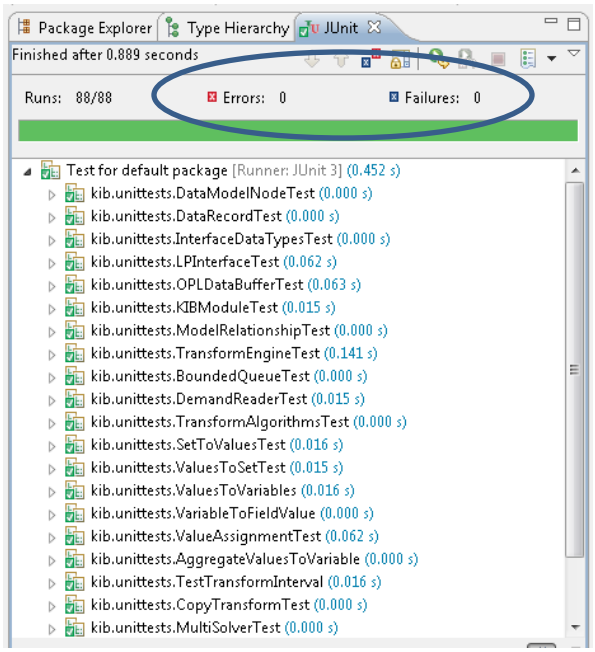

Figure 56. JUnit Test Output

6.2 Evaluation of Scalability

To evaluate scalability, the structure of the single echelon model in the previous version is compared against the same model defined using the new structure. Table 3 gives a quick breakdown that quantifies the definition of the same KIB in the

previous version of the XML code with the redesigned version of the XML code. Because definitions are broken down into each atomic element, it takes more than twice as many lines and elements to define the same KIB. However, since the code is broken down into multiple pieces, there is about a third less content per file.

Table 3. XML File Content Breakdown

	Original Version	Redesigned Version
Number of Files:	1	7
Total Number of Lines:	248	530
Total Number of Elements:	147	388
Average Lines Per File:	248	76
Average Elements Per File:	147	55

6.3 Experiments

6.3.1 Single-Echelon Results

For the single-echelon model, the hub H1 and product P1 was selected for the experiment set. The single-echelon model in Figure 57 shows the configuration setup for this set of experiments.

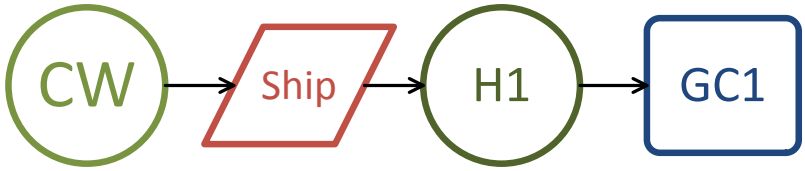

Figure 57. Single-Echelon Model

6.3.1.1 Execution Time Analysis

An analysis of the required run-time was done to determine how the scalability of the OSF platform is as a whole. The chart in Figure 58 shows the running time of a single-echelon model with the simulation running on a weekly granularity and the optimization running on a weekly granularity for 41 weeks. The values on the X axis represent the product of the number of hubs and the number of products the model is running with. As this number increases, the total time to execute increases

exponentially with most of the time taken in the execution of the optimization model.

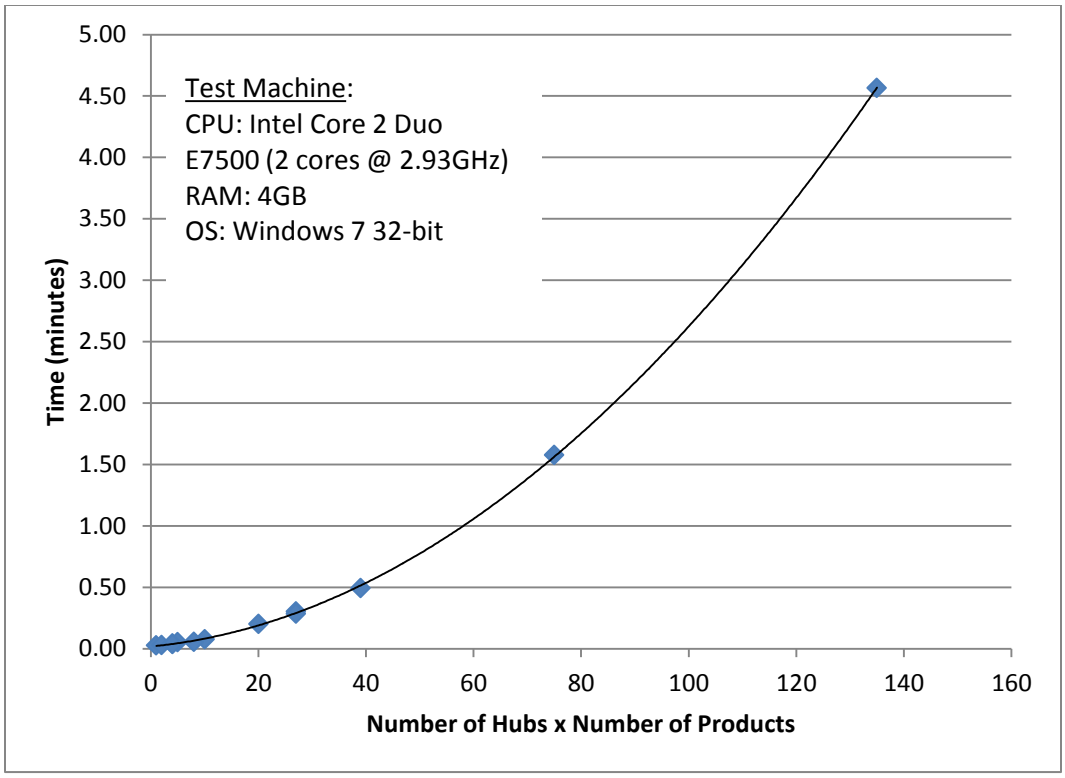

Figure 58. Single Echelon Execution Time

6.3.1.2 Verification of the OSF Model

In this model, "perfect data" is defined as the data used in order to give the model perfect knowledge of the future. In other words, the model knows exactly how much demand will be needed in the future for the length of the planning horizon. When using a deterministic shipping time with safety stock set to 0, this should result in 100% service level with 0 average BOH value.

The chart in Figure 59 shows how well the customer demand, CD, is satisfied using perfect input data with the safety stock set to 0 for every week. With inventory stock in the hub initially set to 0, it takes 3 weeks for the first shipment to be sent to the customer. This is due to the 2-week shipping between the CW and the

hub plus the 1-step processing time in the hub. Since the simulation model for this experiment was set to weekly granularity, 1 step equals 1 week, so the total time from the output of CW to the output of the hub is 3 weeks. After this ramp up time, exactly enough is shipped to the hub 1 time step before it needs to be delivered to the customer. Disregarding the first three weeks, the end result is 0 average stock at the hub with 100% service level as expected.

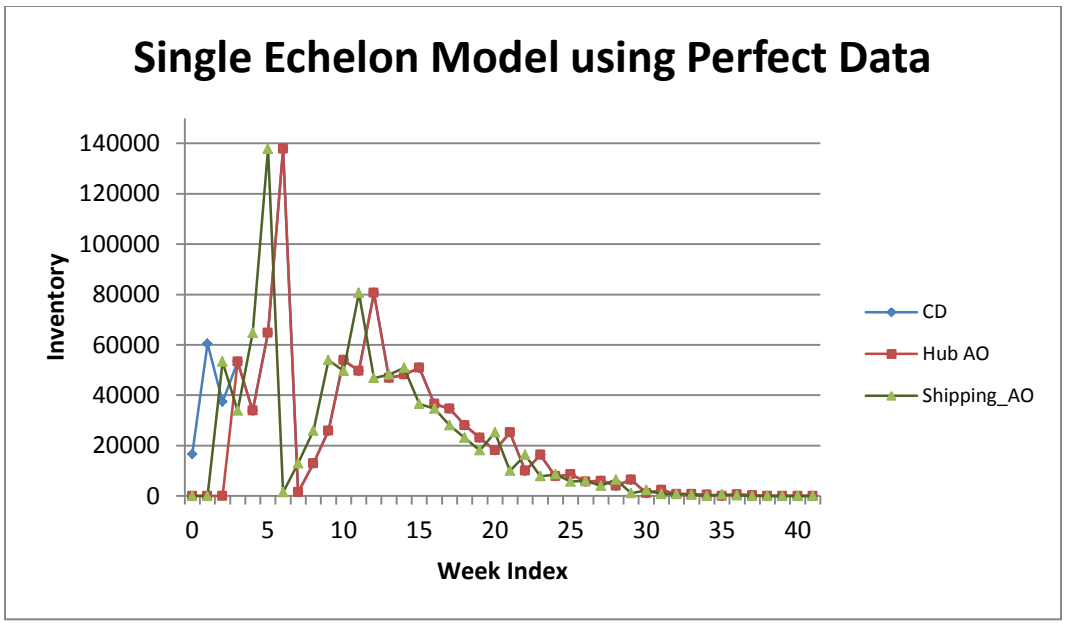

Figure 59. Using Perfect Data

6.3.1.3 Simulation Weekly Step/Optimization Weekly Step

To begin on the experimentation on the OSF platform as a whole, the same model from past work was tested. The chart in Figure 60 shows the results of running a year's worth of data in the OSF platform on hub H1 an product P4. Based on this data, the no-smoothing algorithm outperforms other smoothing techniques. This corresponds to the data retrieved previously for this hub and product.

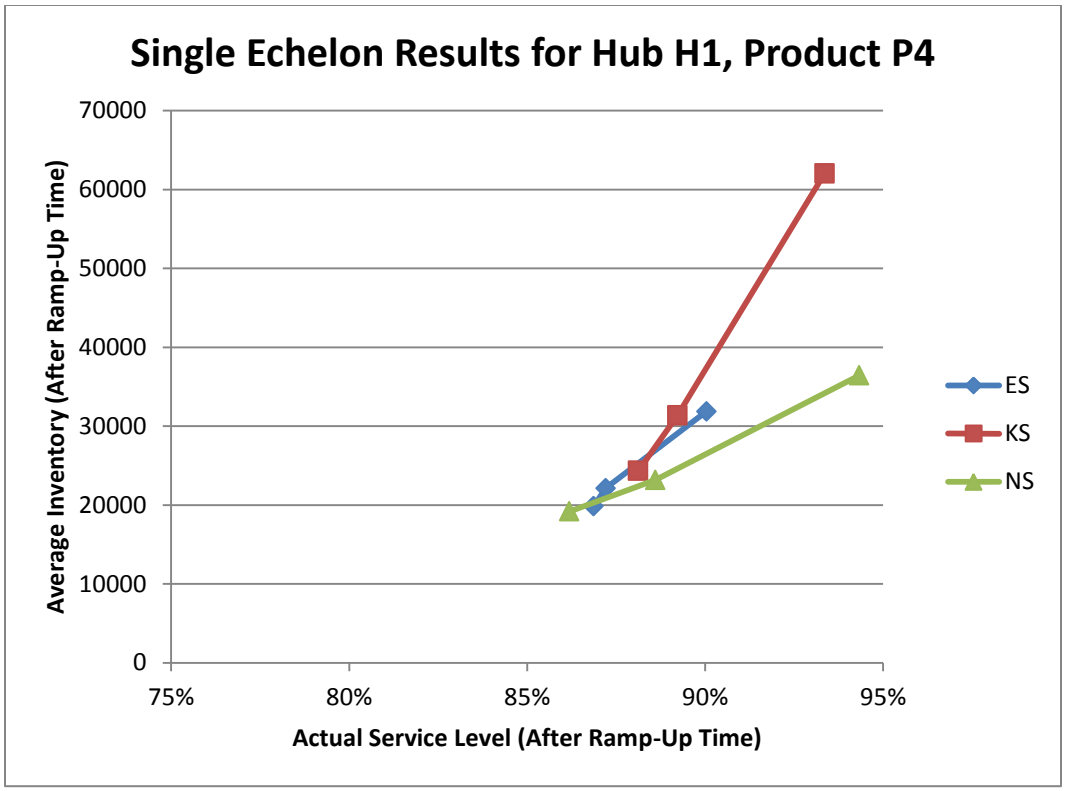

Figure 60. Deterministic, 2 Week Shipping

Since shipping in the real world is not so deterministic, a log-normal shipping distribution was selected. This means that most of the time, each package makes it through the shipping in 2 weeks. Rarely, the package will make it through in 1 week, and very rarely, the package will make it through in 0 or 3 weeks, based on a pseudo-random algorithm. The chart in Figure 61 shows the result after setting up the model the same way as above, with only the shipping element changing to a log-normal distribution. Less average stock is recorded, but the service level has also taken a hit. The same general shape as Figure 60 appears though, so no-smoothing is still the best smoothing technique to use for this configuration.

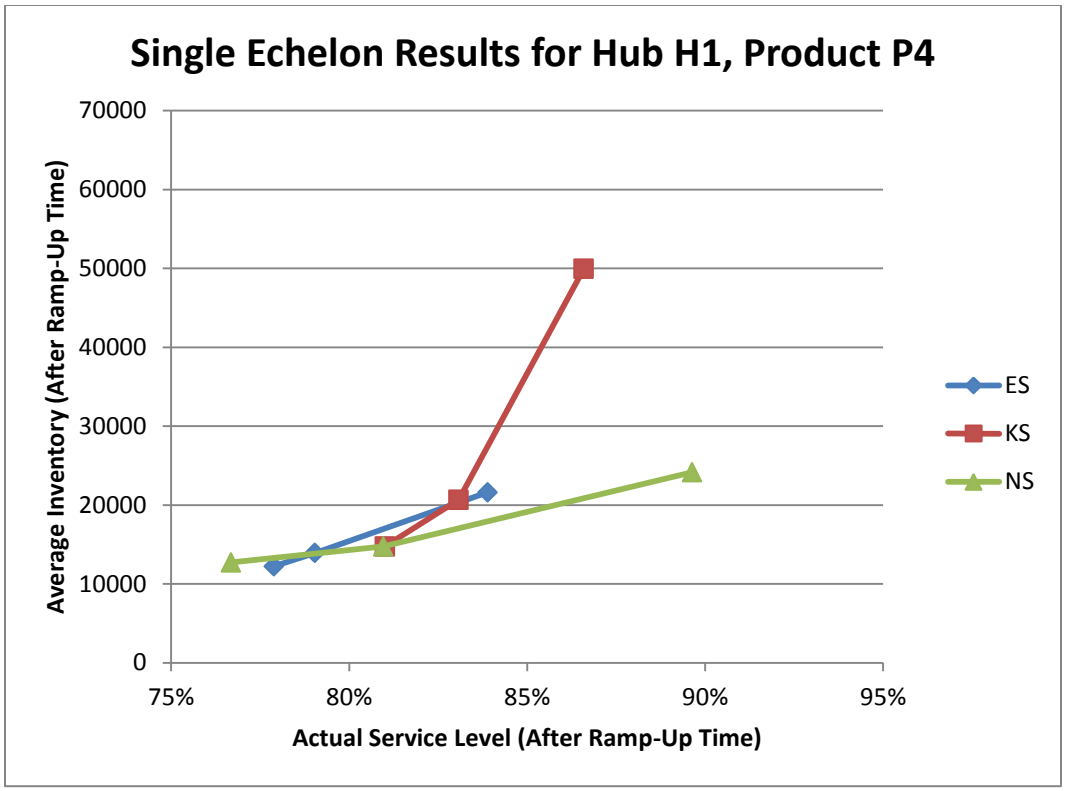

Figure 61. Log-Normal Shipping, 2 Week Mean, 0 Week Min

6.3.1.4 Simulation Daily Step/Optimization Weekly Step

The OSF platform was tested using a more "true to life" setup. Having the simulation run from day-to-day matches a real world shipping schedule where shipments arrive at a single time each day. Optimizations and forecasts are still re-evaluated once each week. In this setup, the KIB's disaggregation components are tested.

Having a strict definition of a time step is not necessary, but labeling a unit of time in a more formal way allows for better usability. In order to formalize a time step metric, the TimeUnit enumeration as shown in Figure 62 was designed. The internal Unit enumeration defines a set of base units from a picosecond all the way up to a millennium. A value is assigned to each instance of Unit which corresponds to how many seconds are in that unit. Setting a base unit of time in the Unit

enumeration allows for simple conversion from one unit value to another. The TimeUnit enumeration contains values such as WEEKLY and DAILY. In each TimeUnit instance, there are 3 attributes associated with it: the string "name" is a distinct name for the instance that is used for labeling, the Unit "unit" is the base unit, and the integer "ticksPerUnit" sets how many simulation ticks make up the length of time of the base unit. In other words, WEEKLY is set to 1 tick per week and DAILY is set to 1 tick per day. Conversion from one time unit to another is provided in the given set of operations.

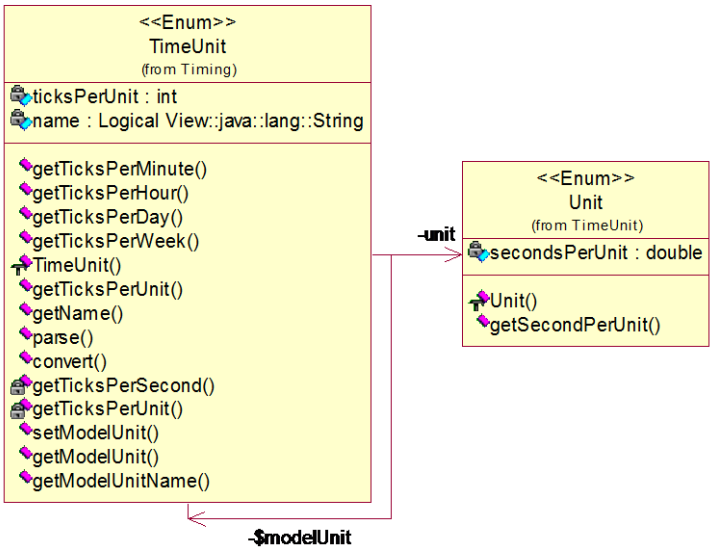

Figure 62. TimeUnit Class

The chart in Figure 63 shows the data obtained after running a daily simulation with a weekly optimization using a deterministic, 14-day shipping model. Comparing this to the chart in Figure 60, more stock is recorded across the board. The kernel and exponential smoothing techniques do worse overall since the outputting service level does not change much as the amount of stock needed goes up. However, the no-smoothing technique does significantly better and is still the best technique for hub H1 and product P4.

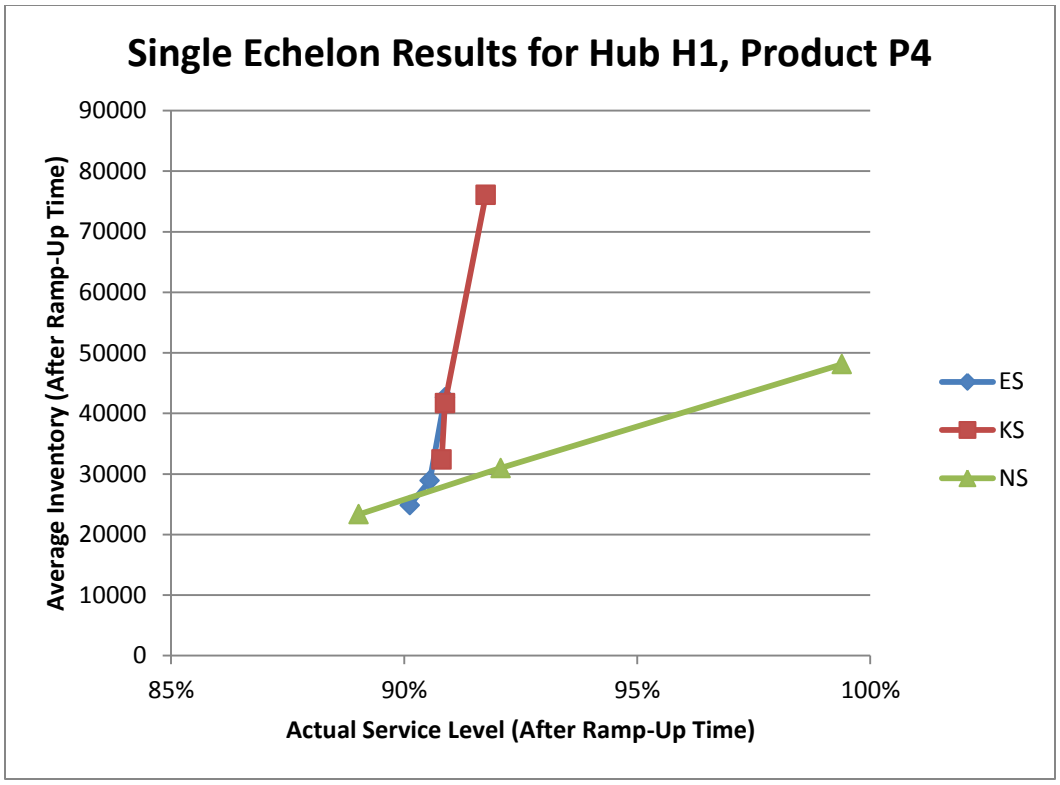

Figure 63. Deterministic, 14 Day Shipping

Now that the simulation runs on a higher granularity, the shipping buckets can be broken down into smaller pieces. Because of this, the log-normal shipping distribution can be setup to better match a real world situation. For the next run, a log-normal shipping component was setup with a 10-day mean and an 8-day min. The chart in Figure 64 shows the results with this setup. With these results, a slightly greater average inventory was recorded. This is because the ISM and LP still assumes an average of 2 weeks for shipping. Since these run on a weekly granularity, the shipping value is rounded up to the nearest week in order to optimize. This results in shipments on average arriving a few days sooner than they are needed. The shape overall is similar to the one above and no-smoothing is once again the best smoothing technique to use.

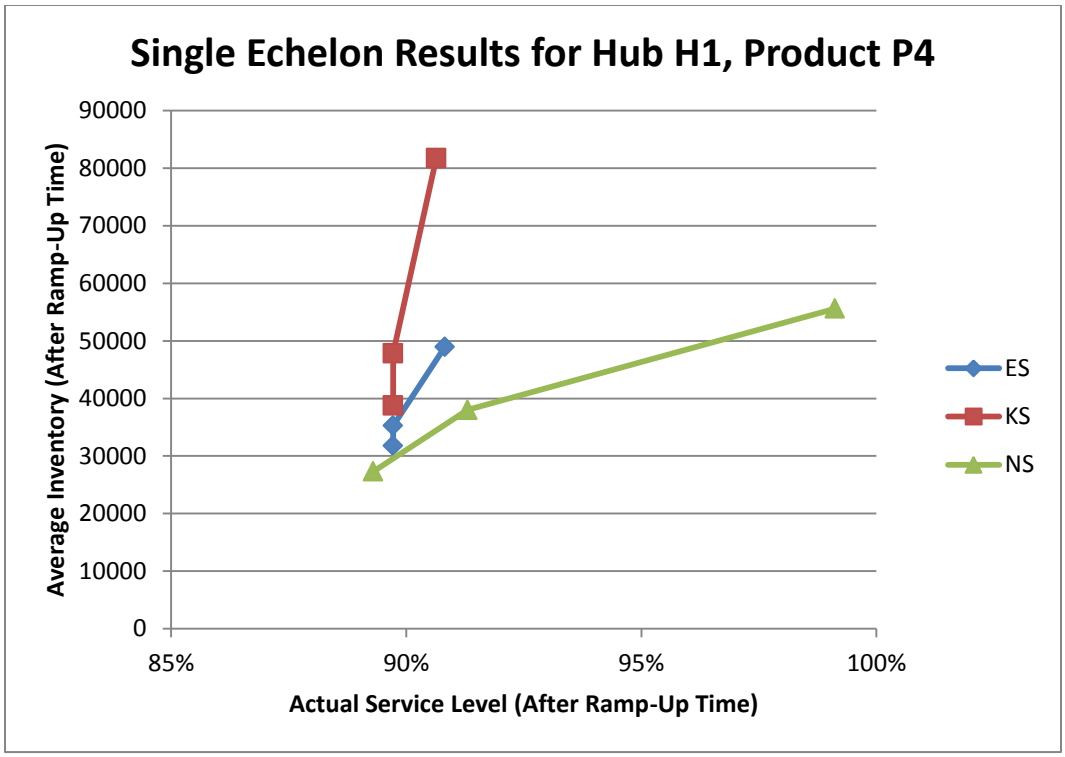

Figure 64. Log-Normal Shipping, 10 Day Mean, 8 Day Min

6.3.2 Multi-Echelon Results

Hub H1 and product P4 was again selected for the multi-echelon experiment set. The model in Figure 65 shows the double-echelon model that was setup. A single CW element was used to keep the overall model simple. This CW needs to also be able to handle product P4 to send to hub H1.

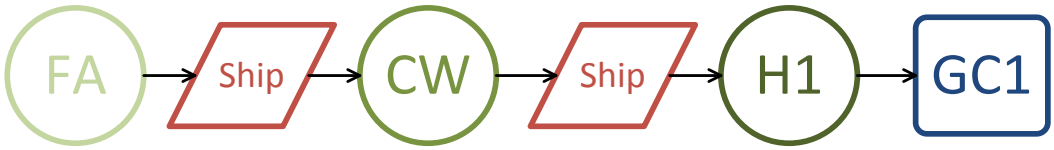

Figure 65. Double-Echelon Model

6.3.2.1 Computation of Upper Echelon Safety Stock

In order for the upper echelon to compute a safety stock, a set of adjusted demand values for the upper echelon need to be computed. This is done by applying a delay function on the demand data at GC1 by the shipping time to H1. An error for the current period's demand is computed using the following formula:

$$Error_{CW} = SS_{H1} - (BOH_{H1} + Intransit_{H1} - fill_{GC1})$$

- SS_{H1} is the safety stock computed for the downstream echelon
- BOH_{H1} is the amount of product that is stored in the inventory of the downstream echelon for the current period
- $Intransit_{H1}$ is the amount of product that is on its way to the inventory of the downstream echelon for the current period
- $fill_{GC1}$ is the sum of demand at GC1 for the time indexes up to the time that it takes for product to reach H1

The error computed is added to the forecast value of the current period to obtain the actual demand for the current period. This value is recorded for historical data in order to compute a bias at future points.

6.3.2.2 Simulation Weekly Step/Optimization Weekly Step

For testing the double echelon model, a shipping time of 0 weeks was selected between factory to component warehouse and 2 weeks between component warehouse and hub. All shipping times in this experiment are constant and deterministic. The charts in figures 66, 67, and 68 show the results double echelon experiment using a multi-echelon ISM. Figure 66 shows an average inventory at the component warehouse; Figure 67 shows an average inventory at the hub (H1); and Figure 68 shows the total average inventory held between the hub and component warehouse in the Y-axis. Overall, kernel smoothing is the best technique for this double-echelon model. Unlike some other modeling concepts for this simplistic model, inventory is kept at the component warehouse instead of releasing the entire stock of inventor immediately to the hub at each time period.

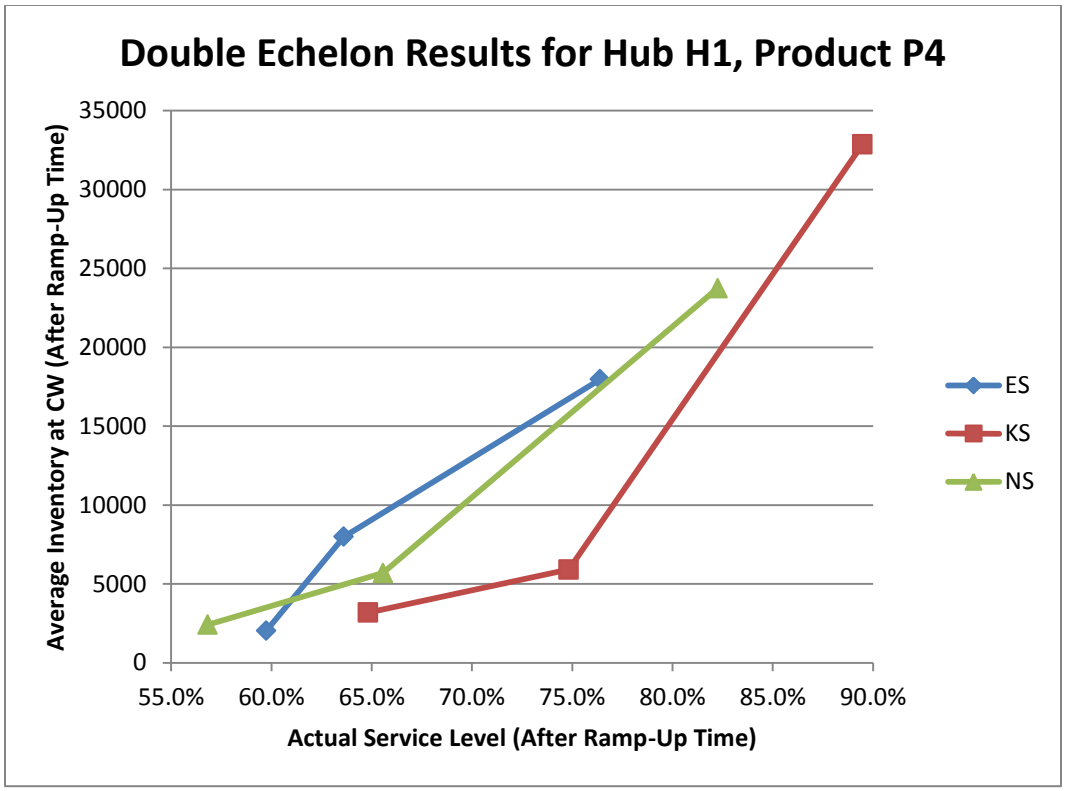

Figure 66. Double Echelon Result: Average Inventory at CW for Service Level

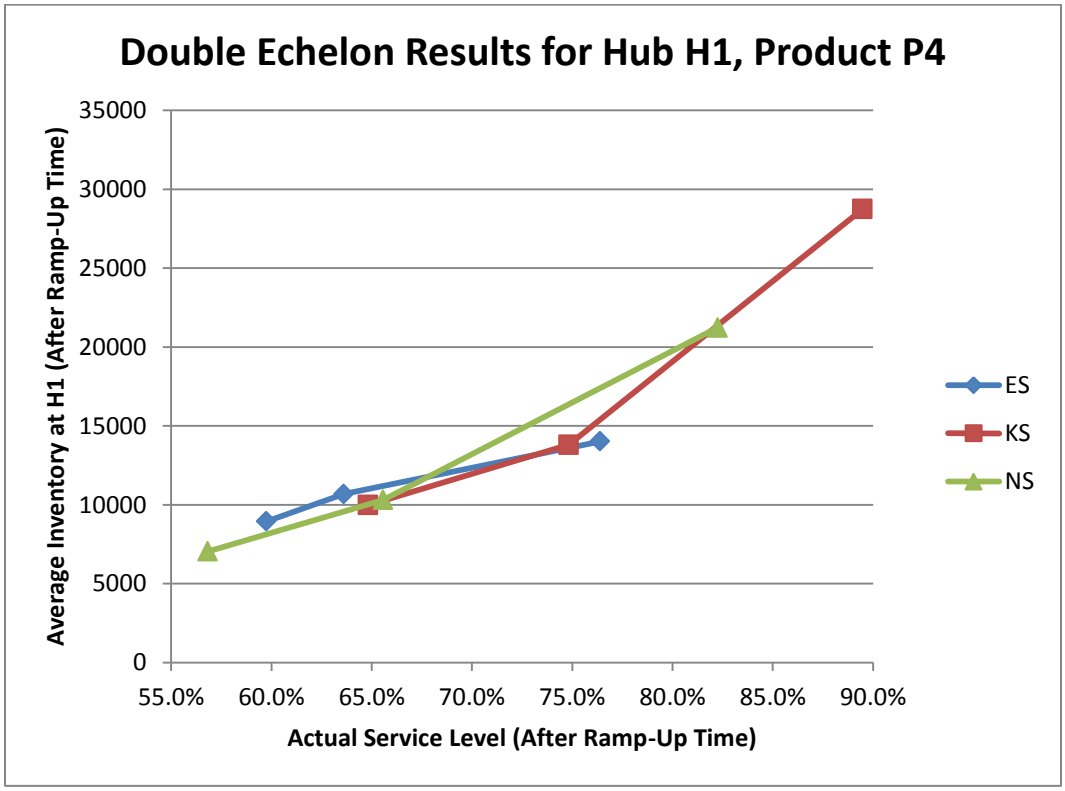

Figure 67. Double Echelon Result: Average Inventory at H1 for Service Level

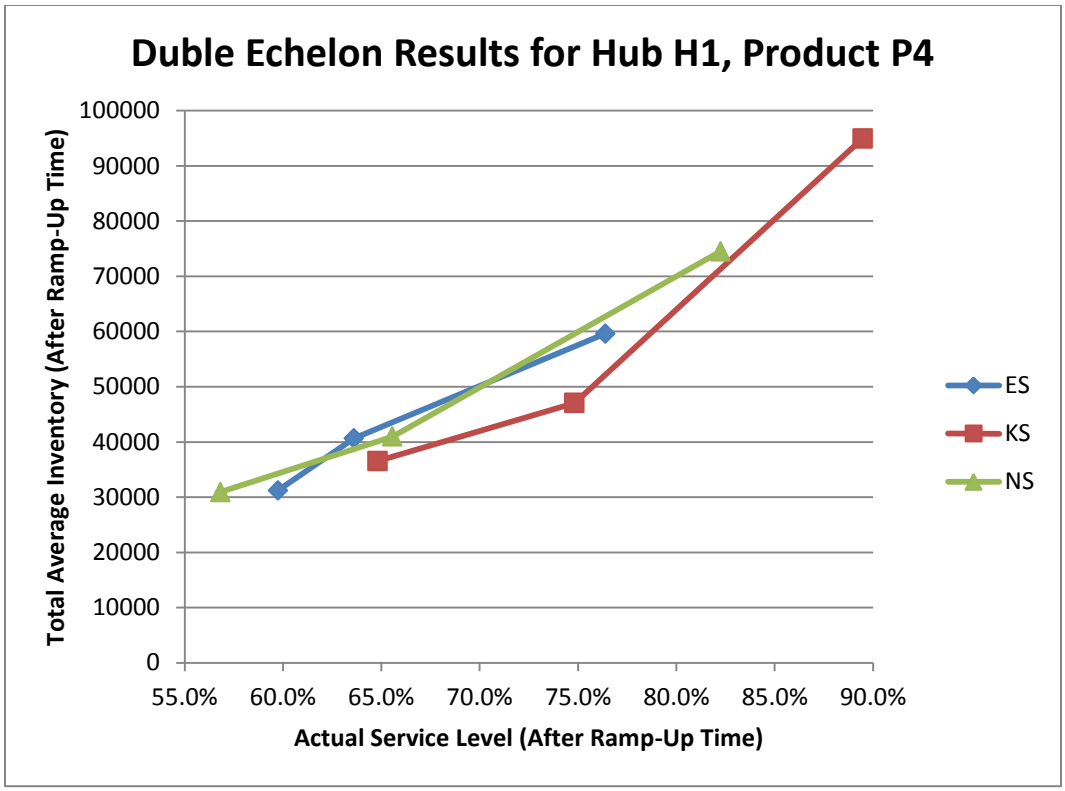

Figure 68. Double Echelon Result: Global Average Inventory for Service Level

Communication between the three model components through the redesigned KIB functions as desired for a double echelon model. Table 4 shows the XML file content breakdown for the double-echelon model. Comparing this to the average lines and average elements per file for a single echelon model in Table 3, a double-echelon model still produces a manageable amount of XML code per file. The average lines and average elements per file is relatively the same as the amount for a single echelon model with the original design of the KIB, but if the original design was used to create this model, the XML file would be around 1550 lines. This would make the KIB definition difficult to navigate and manage. The XML file structure in the redesigned version of the KIB may also be manipulated easily in order to minimize the average amount of code per file even more.

Table 4. Double Echelon XML File Content Breakdown

	Redesigned XML: Double Echelon
Number of Files:	7
Total Number of Lines:	1550
Total Number of Elements:	1155
Average Lines Per File:	221
Average Elements Per File:	165

7 CONCLUSIONS

This project is grounded on creating a multi-echelon simulation with multi-echelon forecast biasing and optimization. Having a supply chain simulation that can quickly and accurately optimize and predict the release of precisely enough stock to meet demand is a highly desired software tool among many corporations. Through the help of several people at ASU and Intel since 2003, as well as the work laid out here, the OSF platform has been established to solve this problem. This platform has the functionality to solve single- and double-echelon supply chain models containing multiple products in multiple inventory elements.

To create a multi-echelon model that is scalable for future design, the original version of the platform needed to be rebuilt from the ground up, starting with the KIB. The KIB, being the backbone of the system, has been redesigned in a way that allows a designer to quickly implement new configurations and allows for better usability, reusability, and scalability.

The designer may now reuse code over multiple configurations by utilizing the KIB_Paths schema design that splits up a KIB definition across multiple files. Components of the KIB are broken down by the definition of modules, control, and transformations through the schemas defined as KIB_Modules, KIB_Control, and KIB_Transformations respectively. Constraining an XML file to define these specific sets of data keeps an organizational pattern that allows for better usability.

A user may now conceptualize a KIB model in the same way a model is designed in each component. The original design has been changed to put modules within models instead of defining interfaces within modules. This not only allows the

designer to create a KIB model more quickly, but multiple models can be defined for a single interface.

Every element of the KIB is broken down to their atomic components to allow for better usability and scalability. When using the auto code completion in IDEs such as Eclipse, defining the KIB in the correct structure can be done in a more guided way. Although the resulting XML files for the definition of the KIB have grown to about twice the size of the same definition given in the old design, having the definition broken down across multiple files reduces the number of lines per file to more manageable chunks.

7.1 Future Work

The next step in the scalability of the KIB is to create a user interface which will allow a designer to better visualize the KIB design. To begin with, a GUI that shows what a previously defined KIB model looks like will help with verification. From here, the interface could be expanded to a clickable model design that allows the user to create new components and connect them together. Different views would need to be designed to zoom into details and zoom out to see the bigger picture. From here, more constraints can be handled that the schema cannot track such as addressing modules within a mapping.

Only initial work has been done for the OSF platform to run a multiple echelon model. The ISM running multiple echelons need to be qualified to ensure that the formalism is correctly matched with definitions of a multi-echelon ISM that Intel and other supply network companies use. The ISM can then be enhanced to handle more complex models for shipping from one to many or many to one inventory elements.

The experiment configuration for this project was used to select functionality for a model to run. As this platform is built upon, more front end work will need to be done to not only select functionality, but to build the structure of a model from a GUI as well. This gives a user who is less familiar with code design the ability to configure and run a model with ease. This is work that is left for future development. Since each model within the OSF platform was designed to be loosely coupled, integrating elements of data together in a way that will confine the definition, while still maintaining loosely coupled components is not a simple change.

REFERENCES

"Extensible Markup Language (XML) 1.0 (Fifth Edition)." World Wide Web Consortium (W3C). http://www.w3.org/TR/REC-xml/.

Godding, Gary. 2008. *A Multi-Modeling Approach Using Simulation and Optimization for Supply-Chain Network System.* PhD diss., Arizona State University.

Godding, Gary, Hessam S. Sarjoughian, and Karl G. Kempf. 2004. *Multi-Formalism Modeling Approach for Semiconductor Supply/Demand Networks.* (paper presented at Society for Modeling & Simulation International: Winter Simulation Conference, Washington DC, December 2004).

Godding, Gary, Hessam S. Sarjoughian, and Karl Kempf. 2007. *Application of Combined Discrete-event Simulation and Optimization Models in Semiconductor Enterprise Manufacturing Systems.* (paper presented at Society for Modeling & Simulation International: Winter Simulation Conference, Washington DC, December 2007).

Graves, Stephen and Willems, Sean. 2000. Optimizing Strategic Safety Stock Placement in Supply Chains. *Manufacturing & Service Operations Management* 2(1): 68-83.

Huang, Dongping. 2008. *Composable Modeling and Distributed Simulation Framework for Discrete Supply-Chain Systems with Predictive Control.* PhD diss., Arizona State University.

Huang, Dongping, Hessam S. Sarjoughian, Gary Godding, Daniel E. Rivera, and Karl G. Kempf. 2006. *Experiment Analysis of Hybrid Discrete Event Simulation with Model Predictive Control for Semiconductor Supply Chain Systems.* (paper presented at Society for Modeling & Simulation International: Winter Simulation Conference, Monterey, CA, December 2006).

Huang, Dongping, Hessam S. Sarjoughian, Welin Wang, Gary Godding, Daniel E. Rivera, Karl Kempf, Hans D. Mittelmann. 2009. Simulation of Semiconductor Manufacturing Supply-Chain Systems with DEVS, MPC, and KIB. *IEEE Transactions on Semiconductor Manufacturing* 22(1): 165-174.

Koch, Markus, Tolujew, Juri, and Schenk, Michael. 2012. *Approaching Complexity in Modeling and Simulation of Logistics Systems (WIP).* (paper presented at The Society for Modeling & Simulation International: Spring Simulation Conference, Orlando, FL, 2012).

Mayer, Gary R. 2009. *Composing Hybrid Discrete Event System and Cellular Automata Models.* PhD diss., Arizona State University.

Mayer, Gary R., Hessam S. Sarjoughian. 2009. Composable Cellular Automata. *Simulation Transactions* 85(11-12): 735-749.

Sarjoughian, Hessam S. 2006. *Model Composability.* (paper presented at The Society for Modeling & Simulation International: Winter Simulation Conference, Monterey, CA, 2006).

Sarjoughian, Hessam S. and Yu Chen. 2011. *Standardizing DEVS Models: An Endogenous Standpoint.* (paper presented at The Society for Modeling & Simulation International: Symposium on Theory of Modeling, Boston, MA, 2011)

Sarjoughian, Hessam S., Gary Mayer. 2010. Modeling Interactions among Heterogeneous Models. *Discrete Event Simulation and Modeling,* Edited by G. Wainer and P. Mosterman (CRC Press), 111-137.

Sarjoughian, Hessam, and Jeff Plummer. "Design and implementation of a bridge between RAP and DEVS." Computer Science and Engineering, Arizona State University, Tempe, AZ (2002).

Sarjoughian, Hessam S., James Smith, and Gary Godding. (In-Preperation). *Optimization, Simulation, and Forecasting: A Platform for Evaluating Long-Term Supply Chain Dynamics Under Demand Uncertainty*

Schwartz, Jay D., Manuel R. Arahal, Daniel E. Rivera, and Kirk D. Smith. 2009. Control-relevant demand forecasting for tactical decision-making in semiconductor manufacturing supply chain management. *IEEE Transactions on Semiconductor Manufacturing*22(1): 154-163.

Schwartz, Jay D. and Daniel E. Rivera. 2010. A process control approach to tactical inventory management in production-inventory systems. *Int. J. Production Economics*125(1): 111-124.

"W3C XML Schema." World Wide Web Consortium (W3C). http://www.w3.org/XML/Schema.

Wang, Wenlin, Daniel E. Rivera, and Hans D. Mittelmann. 2009. Inner and outer loop optimization in semiconductor manufacturing supply chain management. *Computational Management Science*6(4)

"What is SCOR?." Supply Chain Council. http://supply-chain.org/scor.

APPENDIX A

ABBREVIATIONS AND DEFINITIONS

ABBREVIATIONS AND DEFINITIONS

- ACD: Actual Customer Demand
- DEVS: Discrete Event System
- FCCD: Forecasted Customer Demand
- HFC: Historic Forecast
- IDE: Integrated Development Environment
- ISM: Inventory Strategy Module
- KIB: Knowledge Interchange Broker
- LP: Linear Program
- SIM: Simulator/Simulation
- XML: Extensible Markup Language

APPENDIX B

TRANSFORMATION DEFINITION

Except for the transformation labeled as "NONE," each transformation type is classified as a group transformation, value transformation, or both. A group transformation transforms data as a whole while a value transformation transforms data from a single source to a single target. Some group transformations work with other group transformations and some do not.

NONE Transformation

The NONE transformation requires that the target module contains all the data elements of the source module. All the target data elements must also be of the same type as the source data elements. Any attributes for this transformation are ignored and no other transformation should be defined for a mapping if a NONE transformation is defined.

Group Transformations and Priority

1. SET_TO_VALUES
    - This transform takes set of values that has index field, and maps them to an ordered array list to the target model. The order is sequence to send values to target by time period. Each target value maps to a value to be passed in one time period. The index element must be defined for this transformation.
2. VALUES_TO_SET
    - Values in a source array are mapped to set lines. The index element must be defined for this transformation.
3. FIELD_VALUE_TO_VARIABLE
    - Preform transformations only on data that match what is defined in the Field element(s).
4. DisaggregateIntoEqualBuckets
    - Value is divided equally into multiple time period buckets.
5. AllToOneValue, AllCurrentToOneValue, and Aggregate
    - All record values to one target value
6. (No group transformation defined)
    - Data is transformed in a 1:1 manor

Value Transformations

- NewestValue
    - In a list of records as given by model, get only the data record that has been received most recently within the current time period
- OldestValue
    - In a list of records as given by model, get only the data record that has been received first within the current time period
- Copy
    - Same as NewestValue
- FloatToInteger
    - Converts source data to target type
- IntegerToFloat
    - Converts source data to target type (same as FloatToInteger)
- Aggregate

    - Aggregates (sums) an array value to single target value or value set (must be numeric)
- MAX
    - Selects the target maximum value from an array (must be numeric)
- MEAN
    - Mean of array is calculated to single target value (must be numeric)
- MEDIAN
    - Selects the target median value from an array (must be numeric)
- MIN
    - Selects the target minimum value from an array (must be numeric)
- SET_TO_VALUES
    - Converts source data to target type
- VALUES_TO_SET
    - Converts source data to target type
- FieldValueToVariable
    - Target is Array: Converts an indexed array to the array target type
    - Target is Non-array: Converts source data to target type
- VariableToFieldValue
    - Sets to static field value
- ASSIGN_FIELD_VALUES
    - Not much different to VariableToFieldValue
- DisaggregateIntoEqualBuckets
    - Divides numeric value by the number of buckets