A MULTI-MODELING APPROACH USING SIMULATION AND OPTIMIZATION FOR SUPPLY-CHAIN NETWORK SYSTEMS

by

Gary Wade Godding

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

ARIZONA STATE UNIVERSITY

August 2008

A MULTI-MODELING APPROACH USING SIMULATION AND

OPTIMIZATION FOR SUPPLY-CHAIN NETWORK SYSTEMS

by

Gary Wade Godding

has been approved

May 2008

Graduate Supervisory Committee:

Hessam Sarjoughian, Chair
James Collofello
Esma Gel
Karl Kempf
Arunabha Sen

ACCEPTED BY THE GRADUATE COLLEGE

ABSTRACT

Enabling two large complex models that use different types of execution algorithms to work together is a difficult problem. The semantics between the models must be matched and the execution of the algorithms have to be coordinated in a way that the data I/O timing is correct, the syntax and semantics of the data I/O to each algorithm is accurate, the synchronization of control is maintained, and the algorithms concurrent execution is managed.

In this work, an approach is developed that uses a Knowledge Interchange Broker (KIB) to enable composition of general optimization and discrete process models consistent with their respective execution algorithms. The KIB provides a model specification that matches data and control semantics across two distinct classes of optimization and discrete process models. The KIB has a sequential execution algorithm in which two independent execution algorithms are synchronized.

A domain where this kind of problem is seen is in the development of computational models of real world discrete manufacturing supply chain network systems. Modeling these systems require that the planning systems, manufacturing process flows, and their interactions to be concisely described. Planning systems generally use optimization algorithms for calculating future instructions to command what the manufacturing processes build. Manufacturing processes are commonly modeled using discrete event simulations. For supply chain network systems, the planning and manufacturing models can be very large. The KIB is an enabler for correctly combining these models into semantically consistent multi-models.

Two common types of models used for planning and manufacturing are Linear Programming (LP) and Discrete Event Simulation (DES). In this work, a KIB has been developed and demonstrated on a set of theoretical representative semiconductor supply-chain network problems using LP and DES. The approach has then been successfully applied to integrating planning and simulation models of real-world problems seen at Intel Corporation's multi-billon dollar supply-chain network.

ACKNOWLEDGMENTS

I would like to acknowledge Dr. Sarjoughian and Dr. Kempf for their support and encouragement of carrying out this research. From Intel Corporation I would like acknowledge Dave Burba, Ben Wang, and Michael O'Brien for their ongoing support throughout the five years it took to complete and Kirk Smith for his role in developing the production MPC models. From ASU I would like to acknowledge Randy Singh for his simulation work on the common control bus and Donping Huang for her related KIB research work. And finally I would like to acknowledge Intel Corporation for sponsoring this research at ACIMS, ASU for one year.

TABLE OF CONTENTS

Page

CHAPTER Page

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

1 INTRODUCTION

Many computational systems are constructed from subsystems that have differing capabilities and characteristics. Subsystems are logical partitions that can focus on a particular functionality or specialized computational problem. One clear division that can be made when developing computational models of the real world is the separation of physical and logical processes. The system being studied frequently has some physical behavior that develops over time (projection subsystem), but requires direction from a planner (decision subsystem) whereby it can select among possible actions to help achieve a desired overarching system goal (see Figure 1). The decision and projection subsystems can be viewed as two distinct modules in which each has its own structure and behavior. However, for these modules to interact, they must overlap in terms of knowledge contained in each and their dependency on one another for input and output.

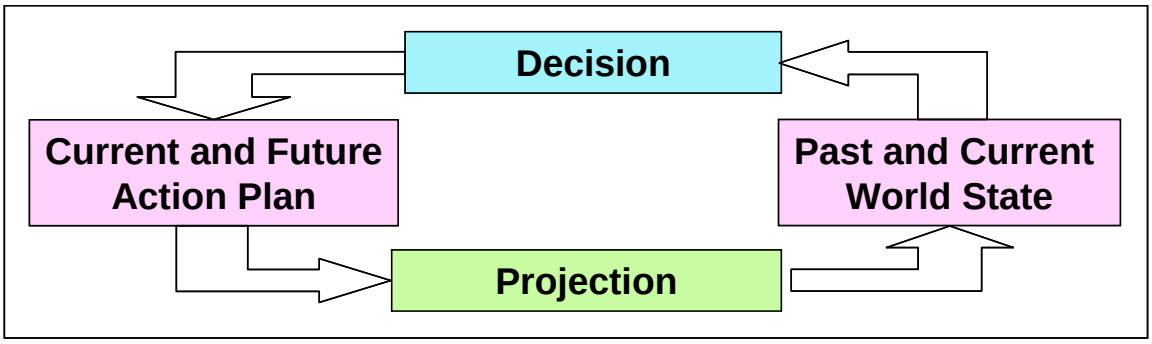


Figure 1. High Level Modeling Strategy

Two extreme options can be employed to address these differences. On one hand, a system design can force the required functionality into a single monolithic model. The result is often difficult to develop, use, and maintain due to the confounding of the decision and projection processes. On the other hand, the models and algorithms for

projection and decision can be designed and implemented separately and then used together in an integrated fashion. The major challenge with this approach is how to integrate the operation of the two modules. Extensive literature exists on computational models for decision making and for system projection. However, descriptions of ad-hoc approaches dominate the integration of the two (Fishwick 1995; Hung and Leachman 1996; Godding and Kempf 2001; Wang 2006; Wang, Rivera et al. 2007). Alternatively, software engineering techniques may be used. From a more general and formal modeling perspective, the integration requires model composability (our particular focus here) and module interoperability as shown in Figure 2. Composability concerns utilizing different models that are semantically consistent with one another. Interoperability focuses on supporting interactions between algorithms runtime.

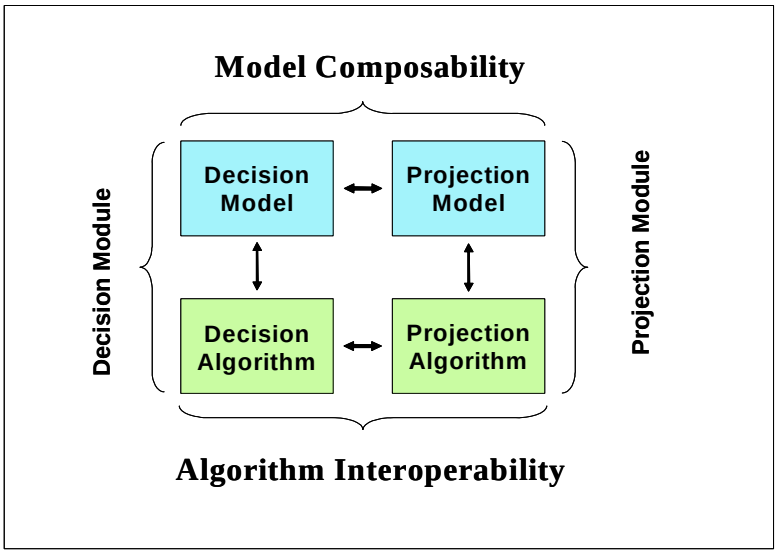


Figure 2. Composability and Interoperability

To explore these issues without loss of generality, we have selected a semiconductor supply chain network as our application domain (Figure 3).

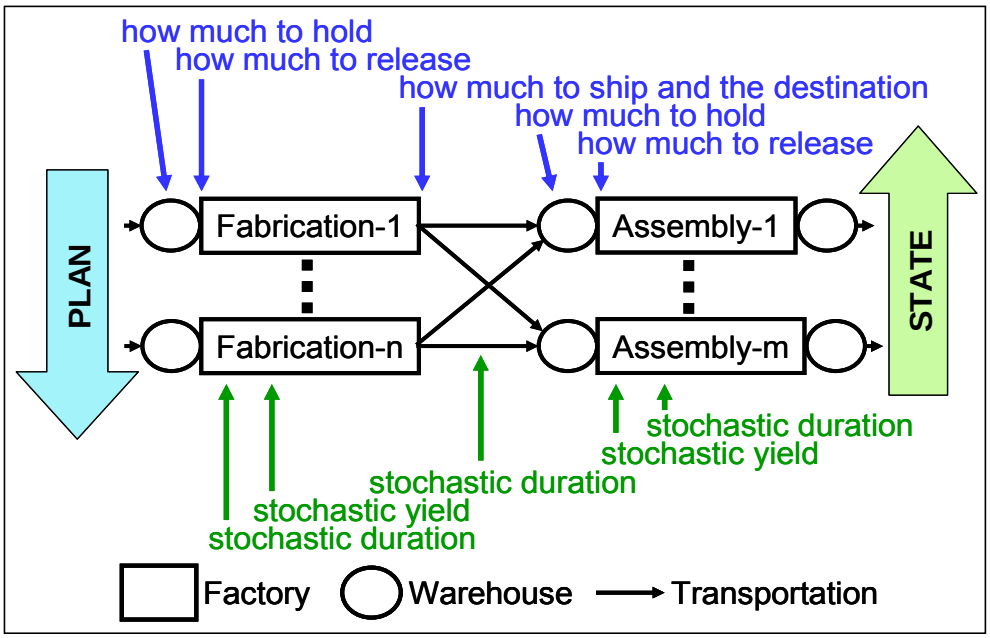


Figure 3. Semiconductor Supply Chain Network Sample

The *decision module* is charged with building a plan for current and future activities that include releasing materials into factories, shipping materials between locations, and moving materials to and from warehouses based on a profitability goal. The *projection module* is intended to represent a model of material movement through the supply/demand network processes consisting of factories, warehouses, and transportation links of the real system including as much of the stochasticity of the real world as is possible and appropriate. The decision module needs the state of the world, resulting from the past application of the previous plan, inputted before it can project the plan for the future. The execution module needs this plan of action to project forward in

time and produce the future states of the world. Clearly these models require different algorithms.

To ground our exposition of composability and interoperability, we have chosen specific decision and execution approaches for experimentation. From the range of possible mathematical and heuristic decision algorithms, Linear Programming (LP) is our choice and Discrete Event Simulation (DES) has been selected from the range of possible execution projection approaches. The focus of our research is not the use of LPs and DES's for Semiconductor Supply Chain Network, but rather the general and specific aspects of the model's composability in this application domain. We have described our work using LP and DES in the Semiconductor Supply Chain Network domain in other publications (Kempf, Knutson et al. 2001; Kempf 2004) and references therein.

Given the complexity of operating a Semiconductor Supply Chain Network, the test-bed partially described here has far-reaching practical implications. The ability to refine decision processes and financial goals as well as the number, topology, and properties of the physical entities in the network without the added time and cost of continuously reprogramming interfaces is a major improvement over existing methods.

1.1 Problem Description

The integration of decision and projection processes presents some challenging problems. The underlying objectives of theses two types of operations are fundamentally different. Decision systems are concerned with finding a good answer when many exist. Projection processes are focused on how the states of a system will evolve over time. Different types of modeling formalisms and algorithms can be found that support the

modeling and execution of either decision or projection operations in an efficient and scalable way. However, there is not an algorithm that provides good support for both. A decision algorithm employs techniques to efficiently search large solution spaces for good answers relative to some metric whereas projection algorithms iterate forward through time reproducing the manner in which states of a system would evolve. Two potential approaches for modeling systems that require both types of computational algorithms would be 1) develop a new universal algorithm and supporting modeling language, or 2) use a methodology that enables the integration of existing approaches. Since the two models, and therefore their algorithms, are focused on different objectives, we believe it would not be practical to pursue a universal algorithm. An approach that supported the integration of existing methods would enable the use of algorithms that have resulted from years of research. It would also enable the evaluation of how different models and algorithms work in a given situation. For example, whether rule-based or mathematical optimization works best for a particular decision problem could be evaluated.

A methodology that requires the integration of different types of modeling formalisms and algorithms must consider model composability and algorithm interoperability. Model composability is concerned with how different components of a model can work together in a semantically consistent way. For example, given decision and projection models, this would require that goal-oriented decision models work with time-based state-oriented projection models. More specifically, if mathematical optimization and discrete event simulation are used for decision and projection

representations of a Semiconductor Supply Chain Network respectively, the optimization models would be composed of an objective function, constraints, and initial data states while the projection models would consist of event I/O, time advance, and state transition functions. Proper model composability would ensure semantically consistent mapping between the initial state and results of the optimization I/O to the time based simulation I/O. Model composability would assure that a semantically consistent representation of the Semiconductor Supply Chain Network is maintained across the optimization and simulation models.

Every distinct type of modeling formalism requires an algorithm that can correctly execute all models specifiable by that formalism. Therefore, when models are specified using multiple formalisms that require different algorithms, it is necessary to account for interoperability between them. In particular, interoperability must handle data and control synchronization. In the case of the Semiconductor Supply Chain Network optimization and simulation multi-models, a mathematical solver and a simulation engine would have to correctly interact with each other. Since significant progress has been made in supporting algorithm interoperability (IEEE 2000; IEEE 2001) our focus will be on how to support model composability.

1.2 Background

Early work on the Semiconductor Supply Chain Network problem had been carried out by looking at the use of an expert system for configuring planning heuristics and running them against discrete event simulations (Godding and Kempf 2001). In this effort an approach was used to connect a commercial discrete event simulator to an

expert system using sockets. This enabled interoperability across the systems; however, it did not allow flexible integration. The discrete event simulation and expert system models had to follow the coordination protocol hard coded into the bridge, and match the incoming and outgoing messages.

Another area of research involved investigating the use of agent models to control a discrete event simulation of a vehicle. The environment created a bridge between a Discrete Event System Specification (DEVS)-based discrete event simulation and an agent modeling tool known as InterRap (Müller 1996) using RAP (reactive action planner). The modeling environment created for this effort was known as DEVS/RAP (Sarjoughian and Plummer 2002). Through this work the idea of a knowledge interchange broker (KIB) originated. The KIB matched the differences between distinct formalisms and enabled the creation of semantically consistent multi-formalism models using heterogeneous execution algorithms.

These two efforts were built upon to create the LP/DEVS KIB described in this dissertation. The Semiconductor Supply Chain Network problem described previously required the use of different formalism for planning and manufacturing models. For the Semiconductor Supply Chain Network models there is also the need to aggregate and disaggregate data across time horizons and into different granularities to match the real world models. There is also a desire to experiment with different planning frequencies to discover the most efficient way to operate. These requirements led to the development of a KIB where data transformations and modes of sequential control can be configured in an integration model. That is where the main contributions of this work lie.

A secondary contribution is the development of a methodology to decompose Semiconductor Supply Chain Network models into manufacturing and planning components. The methodology uses the supply chain network topology, the product routes, and the bill of materials (BOMS) to identify the points in which product differentiation should be simulated versus where they should be modeled as an explicit external control decision.

1.3 Summary of Contributions

The key contributions of this dissertation are:

- The creation of a methodology for developing LP/DEVS multi-formalism models using a Knowledge Interchange Broker (KIB)-$KIB_{DEVS/LP}$.
    - A type of multi-formalism (also known as poly-formalism) is developed where existing research is leveraged through enabling the composition of different modeling formalisms that are known to be well-suited for solving the various parts of a problem. The $KIB_{DEVS/LP}$ enabled the use of the native DEVS and LP languages along with their supporting protocols. This was a pioneering effort in multi-formalism modeling using heterogeneous execution algorithms.
    - A $KIB_{DEVS/LP}$ is developed to support a simulation test-bed for semiconductor supply chain network planning and manufacturing models. The test-bed demonstrated the composition of the OPL-Studio, DEVSJAVA, and $KIB_{DEVS/LP}$ environments using a representative semiconductor supply chain network models.

- The application of the $KIB_{DEVS/LP}$ to modeling real world scenarios.
    - The $KIB_{DEVS/LP}$ enabled the development of a new controller technology for managing logistics problems seen at Intel Corporation. The $KIB_{DEVS/LP}$ facilitated the development and validation of the control technology against simulation, which significantly advanced the state-of-the-art of real world supply chain MPC control.
    - The $KIB_{DEVS/LP}$ enabled control models to be built at a scale never seen before in production.

1.4 Dissertation Organization

The dissertation has been organized into seven sections. A brief description of each follows:

In Chapter 1, the area of research we are going after is described. An introduction is given to model composability and interoperability. An overview of how the planning and physical parts of a semiconductor manufacturing supply chain network can be viewed as two separate systems. The idea of using a methodological approach for the integration of decision and manufacturing models in Semiconductor Manufacturing supply chain networks is described. The contributions of the research is outlined and description of the remainder of the dissertation.

In Chapter 2, we give a background of related work. Different types of multi-formalism modeling are currently being researched for a variety of problems. However, the type of multi-formalism modeling using distinct execution algorithms has, until recently, been unique to this body of work. This work pioneered the approach applying it

to linear programming optimizers with DEVS discrete event simulations. Considerable research exists for integrating simulations with optimizations, however, how to support model composition has been overlooked. Also, in this chapter, we will examine some related work to modeling the types of supply chain network problems under consideration.

In Chapter 3, a detailed description of using a KIB for multi-formalism modeling is given. The importance of separating the domain models conceptually and mapping them to formalisms is described. Then, how to enable formalism composability for DEVS and LP is shown. An approach using a modeling language and protocol to enable the multi-formalism composability is outlined. And finally, an example is provided using an XML implementation.

In Chapter 4, an approach to Semiconductor Supply Chain Network multi-modeling for separating the decision models from the physical manufacturing and logistics facilities is provided. Mapping models of the supply chain network topology and product routing flows are created. From these mappings we can distinguish between the points where product changes are established by explicit decisions versus where product changes are determined by the state of the physical processes.

In Chapter 5, the case studies and experiments completed using the multi-formalism methodology is presented. First, a set of experiments were run against a theoretical problem set. These experiments enabled the development and validation of the KIB environment. It also provided proof that the multi-formalism methodology works. A second set of experiments and studies were run on real world models seen at

Intel Corporation. These experiments showed the benefits of the KIB on real world problems. It also showed scalability of multi-modeling to the large problems seen in the industry today.

In Chapter 6, the design of the software environment and the KIB are explained. The approach taken for application integration is explained. The architecture developed to enable the multi-formalism model composibility is described. The detailed KIB architecture and software design is showcased.

In Chapter 7, the conclusions and future work are discussed.

2 BACKGROUND

2.1 Modeling Methodology

Two important concepts that provide the basis of our modeling methodology are 1) use well suited models for solving the problem at hand, and 2) separate the models from their underlying computational algorithms. At a lower level, these concepts have been researched and demonstrated for software programming (Dijkstra 1976). The use of different formalisms suggests that different modeling approaches work better for some types of problems than others. The separation of the model from its execution algorithm enables the use of efficient, provably correct computer algorithms. For modeling and simulation theory this is analogous to separation of the model from the underlying simulation algorithm (Zeigler, Praehofer et al. 2000). This kind of separation has been used within system-theoretic worldview and logical processes worldview (Fujimoto 2000).

The separation of a model from its execution protocol (i.e., algorithm) has also been considered in distributed settings. This separation focuses on a framework where logical processes and system-theoretic concepts are used in supporting distributed simulation. Unfortunately, while the separation of the model and simulation protocol is necessary for model composability, it is not sufficient (Dahmann, Salisbury et al. 1999; Kasputis and Ng 2000; Sarjoughian and Zeigler 2000). Nonetheless, this kind of separation is the fundamental starting point to our multi-formalism modeling approach.

The ability to use different types of models to solve varying facets of a problem is not a new concept. In the modeling and simulation literature, a variety of general

purpose terms have been used. Examples include multi-modeling (Fishwick 1995), multi-faceted modeling (Zeigler and Oren 1986) and multi-paradigm modeling (Mosterman and Vangheluwe 2004). While multi-modeling and multi-faceted modeling literature identifies the need for using different types of models for large systems, they do not explicitly address how to compose them. Multi-paradigm modeling looks at methods of combining different kinds of models but does not address the problem of combining models with very different formalisms such as optimization and simulation.

2.2 Multi-Paradigm Modeling

Within multi-paradigm modeling, three orthogonal types have been identified: model abstraction, multi-formalism modeling, and meta-modeling. Multi-paradigm defines model abstraction as the process of finding a type of model to solve a problem. For instance, the use of an optimization model for solving a planning problem is a form of abstraction. This research is not focused on model abstraction. It is leveraging the fact that other disciplines have done considerable research on finding abstract models that will elegantly and efficiently solve their specific problems. For example, methods of abstracting planning problems into linear programs have been an area of operations research (Rardin 2000). The use of linear programming for solving planning problems is studied since efficient implementations of solvers exist, which can find optimal solutions on large numbers of variables. Using a linear program requires the practitioner to abstract the problem into a set of linear equations. Leveraging existing research is an important consideration when dealing with complex systems across different fields of study. An entire field of study can exist on the use of a particular modeling formalism

and well known textbook approaches are formulated for creating those types of abstract models.

Multi-formalism modeling is defined as the use of different formalisms for solving a problem. A *formalism* has a language and a protocol that can correctly execute models written in that language. Picking a formalism that can easily support the model abstraction to solve the problem is desirable. For example, an operations researcher using optimization models to solve planning problems would be more efficient using a framework that supports an optimization language and solver. Similarly, a process engineer interested in studying the stochastic behavior of a manufacturing problem could choose discrete events as a model abstraction and the DEVS formalism for creating the models. To observe how the manufacturing models behave against the planning solver the optimization models could be combined with the DEVS models to create a multi-formalism model. In this research the objective is to find an efficient, correct, and flexible methodology for composing optimization and DES multi-formalism models.

The third type of multi-paradigm modeling is defined as meta-modeling. Meta-modeling is the ability to represent a model equivalently in a higher level formalism. Meta-modeling has been used to combine similar formalisms into a single formalism by use of model transformation. This type of meta-modeling use similar sub-formalisms of dynamic systems such as state-charts and Petri-nets (Vangheluwe and de Lara 2002). The Unified Modeling Language (UML) is an example of a very general meta-modeling language for object-oriented software engineering. UML enables the modeling of logical classes and the realization of the classes as objects. Meta-modeling abstracts a modeling

formalism into another modeling formalism. For fundamentally different formalisms like LP and DEVS, abstracting one into the other would not help in providing an environment that could efficiently execute the composed meta-model since their protocols are fundamentally different. This research does not consider meta-modeling as a viable solution for efficiently executing complex multi-models.

When partitioning models into different components, the issues of model composability and algorithm interoperability must be considered. Composability research has centered on the ability to create components of models that can be re-used in many different types of situations (Davis and Anderson 2004). An approach to model composability, therefore, must ensure syntactic and semantic integrity of the parts and their combinations. Composite models with appropriate syntax and semantic underpinnings can then be ensured to correctly execute and interoperate.

Interoperability is the capability for different computer programs (written in different programming languages) to communicate correctly via a protocol that manages message syntax and synchronization. Interoperability does not ensure that each connected program performs in a semantically consistent manner. Instead it ensures that the connected programs pass data via consistent control protocol and are synchronized at the process execution level. If the programs being connected are engines for executing models, and the models have been composed in a manner that is semantically consistent, existing interoperability techniques can be used to enable the correct execution of the models. For software engineering, general middleware technologies and infrastructures exist such as the Common Object Request Broker Architecture (CORBA 2005) and .NET

(Platt 2003). For simulation, the High Level Architecture (HLA) standard (IEEE 2000; IEEE 2001) has been defined that enables distributed simulations to interoperate. These interoperability protocols enable the correct execution of two different applications based on a protocol, but do not ensure that model composability has been addressed. Model composability, which is the focus in this research, must be considered for semantically consistent multi-formalism models.

2.3 Multi-Formalism Modeling Approaches

For the Semiconductor Supply Chain Network problem in this work, the LP and DES formalisms will be used to model the decision and projection models respectively. A discrete event simulation model is a mathematical representation of the world in terms of events, states, time, and an encompassing structure. The simulation algorithm is a mapping from the model into an operational form that lends itself to execution on a computer. The same is true for mathematical optimization; the models are in the form of equations and the solvers are algorithms that can be executed. Since LP and DES are two distinct formalisms with very different execution algorithms, our focus will be on multi-formalism modeling that enables the coordinated execution of each algorithm. In this work the knowledge interchange broker (KIB) theory will be built upon to enable semantically consistent composition and execution of the LP and DEVS models.

Multi-formalism modeling requiring coordination of model interactions described in different modeling formalisms has been referred to as poly-formalism (Sarjoughian 2006). This approach proposes a KIB to account for the mismatch between the model semantics and execution algorithm interoperability. Enabling the composition of

disparate formalisms using an interaction model broker was first suggested in (Sarjoughian and Plummer 2002) where the authors defined how to compose DEVS and Reactive Action Packages (RAP) formalisms (Firby and Fitzgerald 1999). The description of that research was further refined and its implementation extended in (Sarjoughian and Huang 2005). The DEVS/RAP work did not account for other kinds of modeling formalisms requiring different capabilities. Nor did it account for scale and complexity of domains such as supply-chain systems (Godding, Sarjoughian et al. 2003; Kempf 2004) or human-landscape dynamics (Mayer, Sarjoughian et al. 2006). The use of a KIB for the Semiconductor Supply Chain Network multi-formalism modeling defined in this dissertation started early 2004 (Godding, Sarjoughian et al. 2004) and is currently being applied to industry scale problems (Godding, Sarjoughian et al. 2007).

Other related research in the development of KIB theory is ongoing. Theory concerning a KIB supporting asynchronous control and parallel execution with Model Predictive Control (MPC) and DEVS formalisms has been examined using the Semiconductor Supply Chain Network application domain as an exemplar (Huang, Sarjoughian et al. 2007; Huang 2008) and new kinds of interaction models for agent-based and cellular automata formalisms (Mayer and Sarjoughian 2007) are being researched.

Prior to the KIB, alternate approaches could be found to enable models written in different formalisms to work together. These approaches are illustrated in Figure 4. The first method is the most commonly seen. A connection between the two applications is created using off-the-shelf interoperability tools. This form of integration addresses

composability in an ad-hoc way (see Figure 4a). The simulation engine is connected to the optimizer that enables the systems to interoperate. However, model composability must be addressed within each of the applications. This results in development of inflexible solutions. The applications are integrated in such a way that makes the implemented models work. If a change is required in the integration, both applications must be updated.

Many commercial packages offer this type of connection via middleware, or data connection through a database. Research has shown the value of MPC for supply chain control by demonstrating it with a MatLab Simlink simulation (Wang 2006; Wang, Rivera et al. 2007). However, since the work did not consider model composability concepts, the integrated MPC and simulation model resulted in an implementation which lacked conceptual integrity, robustness, scalability, and performance. Applications working together in a related area will create standards that define a message structure and protocol for communicating (Gartland, Godding et al. June 30, 2000). This resulting standard is then considered when composing the models. These implementations work well when the relations and interface between the two models can be easily formulated in terms of simple inputs and outputs that conform to a chosen interoperability protocol. They also work well in mature environments where there is little change. However, this approach is not very flexible for experimentation, research, and bringing in new capabilities. This flexibility is a key requirement for the type of multi-formalism modeling being developed in this work.

In early related work a framework was built to enable an expert system work with a supply chain network DES (Godding and Kempf 2001). A broker was designed that enabled connectivity and messages to be sent between the two. The design closely followed that of the broker pattern (Buschmann, Meunier et al. 1996) having a server and API's, or proxies, on each of the applications. This design worked well for transmitting messages and enabling the concurrent execution of algorithms. It provided an implementation in which the data could be parsed at each of the applications; however, model composability concerns were pushed into each of the respective models. While developing each of the models, the names of the variables, the syntax of the messages, and the granularity of the data had to be matched to the other via interoperability concepts. Differences in the formalisms also had to be addressed. For example, with a DEVS Java simulation, ports are used as the originating and destination points for messages. A port is not a construct found in an expert system. Conversely, the name of a rule that fired in an expert system is not something known to the simulation. If an expert system rule was refined and renamed, the simulation model would also need to be updated to be able to comprehend the change. Both of the models must also be composed to work with each other's execution protocols. For discrete event and expert systems, this requires the coordination of asynchronous events with the firing of rules. A model composability approach that allows independent changes of the different models is desired. This would allow an existing model to be reused with different implementations of the other. The KIB enables this by inserting a third model between the other two. Mismatches in the original models are compensated by the KIB model.

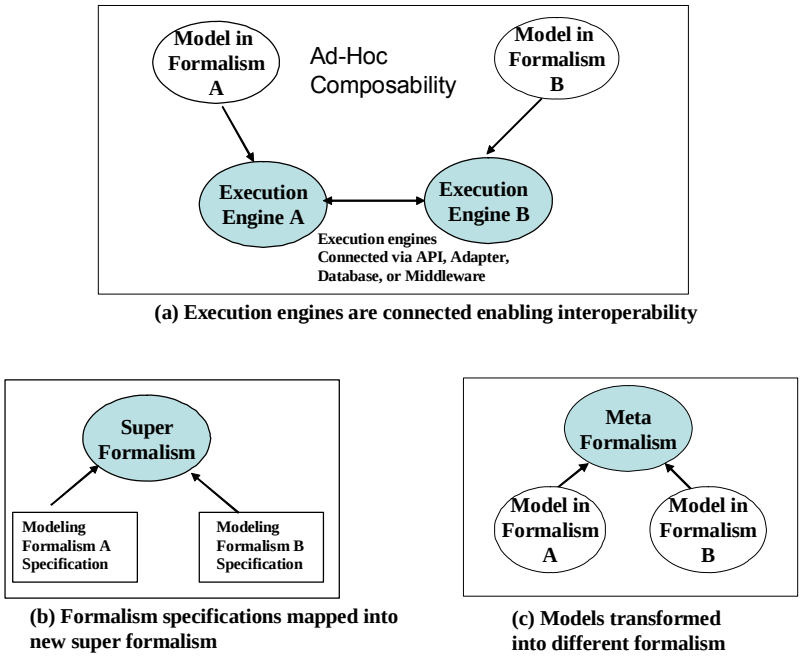


Figure 4. Approaches for Integrating Models in Different Formalisms

The next multi-formalism approach is to map the formalisms into a new super formalism (see Figure 4b). If a supporting algorithm is created, only a single execution engine is needed. For example, a formalism and supporting algorithm exists for representing combined discrete event and continuous models (Prähofer 1991). This approach is not practical for LP and DES models. The two formalisms and their underlying algorithms are fundamentally different and non-compatible.

The last approach, shown in Figure 4c, transforms the formalisms into a meta-formalism that can include the capabilities of the sub-formalisms. If the structure and behavior of the sub-formalisms can be mapped into an equivalent representation of the meta-formalism, the aggregate model can be validated against the properties of the meta-

formalism. Thus, only the algorithm of the meta-formalism would be required to execute the models. This approach has been applied to combining closely related formalisms in terms of how components and their interactions are formalized (de Lara and Vangheluwe 2002). Transformation of models into different formalisms can also be beneficial for validating different properties (Vangheluwe and Lara 2004), However, this approach is not suitable for LP and DES given their inherent differences. The meta-modeling approach would not work well because the LP and DES do not share a common basis that can be used to describe their structures and behaviors. A DES is a state-based event model which use ports for coupling and has a concept of time. LP models consist of an objective function and constraints. Transforming these two types of models into a single representation would be impractical.

2.4 Semiconductor Supply Chain Network Modeling

Semiconductor supply demand networks are large complex systems consisting of interdependent flows of material, control, and capital (Kempf 2004). Material flows are generated when the physical product is manufactured and transported. Flow of control determines how decisions will achieve the desired material and cash flows. Cash flows are produced via revenue from sale of product or expenditures for the cost of manufacturing, storage, and transportation. A practical approach for modeling the Semiconductor Supply Chain Network is to partition the system into different modules and use a framework to enable cross communication of interrelated information flows (Godding and Kempf 2001; Godding, Sarjoughian et al. 2007). There are many benefits to modeling the information and physical flows in a way that they can communicate and

influence each other, such as enabling the analysis of how one impacts the other (Godding, Sarjoughian et al. 2003).

Due to the size and complexity of a supply chain network, data type and granularity are key factors. For the level of decisions being modeled, the approach for modeling the physical entities outlined in (Knutson, Fowler et al. 2001) has been utilized. For modeling controls flows to and from the physical models a bus strategy connecting all the entities is used (Singh, Sarjoughian et al. 2004). With this level of modeling, an interface between control and process has been developed to enable their separation and integration with a KIB. This interface consists of status and control messages with respect to time. The status messages would include inventory levels, work in progress, what is being delivered, and the customer demand. The control messages instruct how much material to release from an inventory.

Partitioning the Semiconductor Supply Chain Network into decision and process models simplifies each of the components; however, each can still be quite complex. The decision module needs to calculate start instructions for complex stochastic processes based on stochastic input signals from the market demand. Process modules must model complex product flows based on stochastic process. The interface between the decision and process modules must be able to transform the data and communicate control between the two (Godding, Sarjoughian et al. 2004).

For experiments using different planning frequencies, the interface between the decision and process models must allow flexibility in how often one module runs in relation to the other. This implies that data flowing between the two can be aggregated or

disaggregated over discrete time periods. Decision and process models are typically developed at different abstraction levels. The process model needs to capture the important events that impact actual flow of material through each entity, whereas the decision module is interested in the current overall state across the entities of a process. Process models are well suited for the use of object-oriented approaches, whereas planning data sets could be the Cartesian product of data elements across many physical objects. This implies that data needs to be transformed across sets of different cardinality. For example, a decision model may need to know how much material is currently available across all inventories and process lines, whereas the process model considers each process and inventory as a separate entity. Sets of data originating from specific physical objects need to be transformed into different views for the decision model.

For this work, linear programming is being used for the decision models. A linear program (LP) is a form of mathematical optimization that has been applied successfully to a wide range of planning applications (Hopp and Spearman 1996; Chopra and Meindl 2001). The purpose of LP is to find the best answer when many exist. Linear programming employs search techniques to find an answer from a set of many different possibilities. Models described in linear programming consist of an *objective function*, *a set of constraints, a set of cost variables, and a set of decision variables* (Wu and Coppins 1981). The LP model relationships must all be linear.

In Figure 5, the standard form for the LP formalism is shown (Moré and Wright 1993). This can be expressed as: minimize c***x** subject to the constraints A***x** =b, and ***x**

≥0. The linear program is solved for the decision variables $\boldsymbol{x}$. An LP modeler would develop an objective function with cost values and a set of constraint equations with constant coefficients.

The coefficients for the cost vector c, constraint matrix A, and the constant vector b are all supplied as initial state values. Some or all of these could be supplied from DEVS outputs depending on the type of problems under consideration. The inputs to an LP consist of an initial state populated into the coefficients for the cost vector c, constraint matrix A, and the constant vector b. The outputs of the LP model would be the set of decision variables $\boldsymbol{x}$.

The protocol for an LP is to find the best answer for a given initial state. Different algorithms exist that an LP formulation can be directly mapped to and then solved for. Two of the most popular algorithms are simplex and interior point. While the algorithms are not part of the formalism, they provide an efficient methodology for finding the answer to the set of equations. The formalism is the mathematical symbols and the meaning behind them (i.e., the syntax and semantics). The syntax is the algebraic specification. The semantics is the protocol that defines the meaning of the syntax.

$\min\{c^* x: A^* x = b, x \geq 0)$
where:
$x \in \Re^{n \times 1}$
$c \in \Re^{n \times 1}$
$b \in \Re^{m \times 1}$
$A \in \Re^{m \times n}$

c is a vector of cost coefficients
x is a vector of decision variables (unknowns)
b is a vector of known constants
A is the constraint matrix

Figure 5. LP Formalism

The process models must be able to model stochastic data flows. DES has been demonstrated as a good choice for modeling complex manufacturing processes (Law and Kelton 1999). Many prominent discrete event modeling paradigms, such as the Discrete Event System Specification (DEVS) (Zeigler, Praehofer et al. 2000), can be used to model dynamic systems. DEVS allows modelers to describe discrete as well as continuous dynamics in terms of discrete-event models. Complex models can be hierarchically constructed from *atomic* and *coupled* models using well-defined interfaces and couplings. This formalism uses mathematical set theory and provides a framework to support model development with well-defined structural and behavioral specifications and model simulation. The DEVS framework has been extended with object-oriented abstraction, encapsulation, and modularity and hierarchy concepts and constructs (Zeigler and Sarjoughian 1997).

An atomic model specifies input variables and ports, output variables and ports, state variables, internal and external state transitions, confluence function, and time advance functions. This type of model is a stand-alone component capable of autonomous and reactive behavior with well-defined concepts of causality and timing. They can also handle multiple inputs and generate multiple outputs.

A coupled model description specifies its constituents (*atomic* and *coupled models*) and their interactions via *ports* and *couplings* (Wymore 1993). A coupled model can be composed from a finite number of atomic and other coupled models hierarchically. Due to its inherent component-based support for model composition, this framework lends itself to simple, efficient software environments such as DEVSJAVA (ACIMS 2002). Atomic and coupled models have sound causality, concurrency, and timing properties that are supported by various simulation protocols in distributed or stand-alone computational settings.

Figure 6 and Figure 7 show the generic parallel DEVS atomic and coupled model specifications (Zeigler, Praehofer et al. 2000). Atomic and coupled models provide basic components for describing component-based hierarchical models. In an atomic model, input and output ports and messages ((InPort, X), (OutPort, Y)) are used to specify the structure. The behavior of an atomic model is specified in terms of the state variables (S) and functions. A model can have autonomous and reactive behaviors by defining internal transition function (δ_{int}) external transition function (δ_{ext}), and confluent function (δ_{conf}).

The output function (λ) allows the models to send out messages. The time advanced function (ta) captures the timing of the atomic model. The confluent function can be used for modeling simultaneous internal events and external events.

Atomic Model = <X, S, Y, δ_{int}, δ_{ext}, δ_{conf}, λ, ta>

where:

- X is the set of input values
- S is a set of states
- Y is the set of output values
- δ_{int} is the internal state function
- δ_{ext} is the external state function
- δ_{conf} is the confluent function
- λ is the output function
- ta is the time advance function

Figure 6. Parallel DEVS Atomic Model Formalism

A coupled model is composed of one or more atomic or coupled models. The structural specification of a coupled model includes input and output ports (with messages), a set of components, and its coupling information. A coupled model does not have direct behaviors. Its behavior is based on the message exchanges between itself and its components as well as message exchanges among the coupled model components. The components of a coupled model can be connected using three types of couplings referred to as external input coupling (EIC), external output coupling (EOC), and internal coupling (IC). Figure 8 illustrates a DEVS coupled model with the EIC, EOC, and IC couplings labeled.

Coupled Model = $\langle X, Y, D, \{ M_d \mid d \in D \}, EIC, EOC, IC \rangle$

where

- $X = \{(p,v) \mid p \in IPorts, \ v \in Xp \}$ is the set of input ports and values
- $Y = \{(p,v) \mid p \in OPorts, \ v \in Yp \}$ is the set of output ports and values
- D is the set of the component names
- $d \in D$, $M_d = \langle X_d , Y_d , S , \delta_{ext} , \delta_{int} , \delta_{conf} , \lambda , ta \rangle$ is a parallel model with
- $Xd = \{(p,v) \mid p \in IPortsd, \ v \in Xp \}$
- $Yd = \{(p,v) \mid p \in OPortsd, \ v \in Yp \}$

external input couplings:

- $EIC \subseteq \{ ((N, ip_N), (d, ip_d)) \mid ip_N \in IPorts, d \in D, ip_d \in Iports_d \}$ connect external inputs to component inputs

external output couplings:

- $EOC \subseteq \{ ((d, op_d), (N, op_N)) \mid op_N \in OPorts, d \in D, op_d \in Oports_d \}$ connect component outputs to external outputs
- internal couplings:
- $IC \subseteq \{ ((a, op_a), (b, ip_b)) \mid a, b \in D, op_a \in OPorts_a, ip_b \in Iports_b \}$ connect component outputs to component inputs
- Note: no direct feedback loops are allowed,
- $((d, op_d), (e, ip_d)) \in IC$ implies $d \neq e$.

Figure 7. Parallel DEVS Coupled Model Formalism

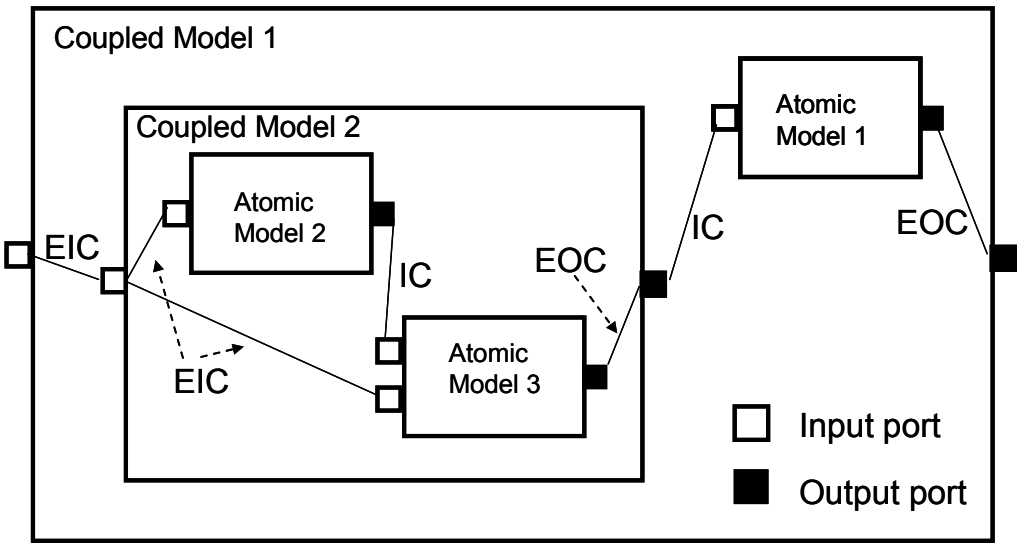


Figure 8. Coupled Models in DEVS

Pictorially, Figure 9 describes the semantics of the DEVS simulation specification. An external event x_j arrives at the input port while the atomic model is in state s_0. The model enters a new state s_1 which is generated by the external state transition function. The state s_1 is dependant on the time elapsed since s_0, the s_0 state values, and the external event x_j. Next, an internal event occurs when the ta for state s_1 has elapsed. An output event y_k is sent to the output port with the values associated with state s_1. The internal transition function then immediately puts the model into state s_2.

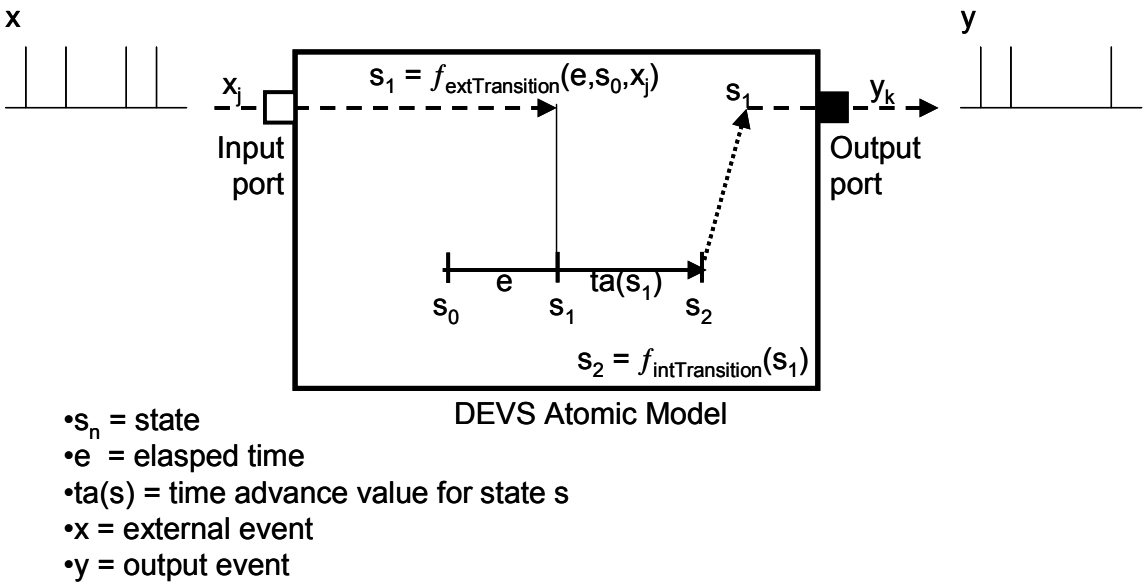


Figure 9. DEVS Semantics

While the DEVS formalism may be used for optimization, its theory is not well suited to support linear programming or other optimization modeling approaches in which general mathematical equations specify constraints among decision variables (Godding, Sarjoughian et al. 2003). Instead, DEVS (and more generally, systems theory) is concerned with describing the structure and behavior of a system in terms of components and simulating those components for a period of time.

Given these complexities of modeling the Semiconductor Supply Chain Network, a feasible multi-formalism modeling approach would need to support the methodological development of complex interdependent decision and process models, a flexible interface to enable the modeling of different types of data transforms, a flexible way to model the interaction frequency, and the use of high performance execution engines.

2.5 Summary

This research has been focused on developing a methodology where the interactions between inherently different classes of models can be described as a model. To carry out this research, it was important to have two disparate modeling formalisms and a rich application domain. Modeling semiconductor manufacturing supply-chain systems are demanding for existing simulation and optimization approaches and tools.

The multi-modeling approach used for semiconductor supply chains in this work has been a valuable tool in evaluating how a planning algorithm performs against a large set of expected values that evolve over a simulation. It has also worked well for tuning a controller to the dynamics seen in semiconductor supply networks. This research was not concerned with how to develop optimization or simulation models, but rather how to leverage existing approaches in these areas and then develop a science to integrate these models to bring about a semantically consistent execution.

Consequently, in this work, we look at composing LP and DEVS formalisms in semantically consistent manner that enables correct multi-model behavior when executed. The science of the KIB theory has been expanded and demonstrated on LP and DEVS supply chain network models. Other related work shows KIB theory to be powerful for different kinds of modeling formalisms and application domains. Based on the advances described in this work for composing optimization and simulation formalisms, the development of a KIB can be extended to other research. For example, simulation optimization is concerned with combining the two to create a new class of optimization algorithms. The work presented here can offer basic concepts and capabilities for

modeling the types of interactions required for the combined execution of the models. However, how to develop simulation and optimization models and how they coordinate would be the work of domain experts. The KIB could facilitate the modeling of the interactions between the two to support the simulation optimization research.

3 APPROACH

An approach is required to describe different parts of a system in well-suited modeling formalisms. Such a case can be made for optimization and discrete event simulation for planning and manufacturing problems. LPs are a type of optimization that have been applied successfully to solve many types of operations research problems. There are several well known scalable algorithms that can easily be realized on a computer. DEVS is another well known formalism for describing discrete event simulations. DEVS has shown itself to be good for modeling dynamic systems. Each of these formalisms works well for its intended use. For example, DES does not fit well for solving optimization problems and LP is ill suited for simulating component-based process flows. These formalisms both have a distinct specification language and supporting protocols for defining the semantics. The protocols can be implemented in efficient computing algorithms. A common optimization algorithm for LPs is the simplex solver and an efficient algorithm exists for the DEVS abstract simulator.

In our case, where we want to develop models using both LP's for planning algorithms and DEVS for manufacturing models, we need an approach that supports the composition of the differing model specifications and the interoperability of their protocols. Figure 10 illustrates the approach we take through the use of a knowledge interchange broker (KIB) (Godding, Sarjoughian et al. 2004; Sarjoughian and Huang 2005; Sarjoughian, Huang et al. 2005; Huang and Sarjoughian 2006; Godding, Sarjoughian et al. 2007; Huang, Sarjoughian et al. 2007; Mayer and Sarjoughian 2007; Huang 2008). Two basic concepts shown in Figure 10 are:

- KIB supports both model composability and execution interoperability through its own specification and protocol. The arrows between the model specifications and execution protocols indicate there is a direct relation between the two. Each specification defines the class of models that can be described and the protocol defines a scheme under which the models can be executed.
    - *Model composability* is supported via a KIB model specification. This specification enables a modeler to stipulate how data is mapped and transformed between the LP and DEVS model specifications. The KIB model specification enables semantic consistency across the LP and DEVS model specifications.
    - *Execution interoperability* is supported through the KIB execution protocol. This execution protocol ensures correct data and control synchronization between the LP and DEVS protocols.
- Data mappings and transformations supported in the KIB must be further specified to handle modeling and execution of complex application domains such as semiconductor supply/demand network.

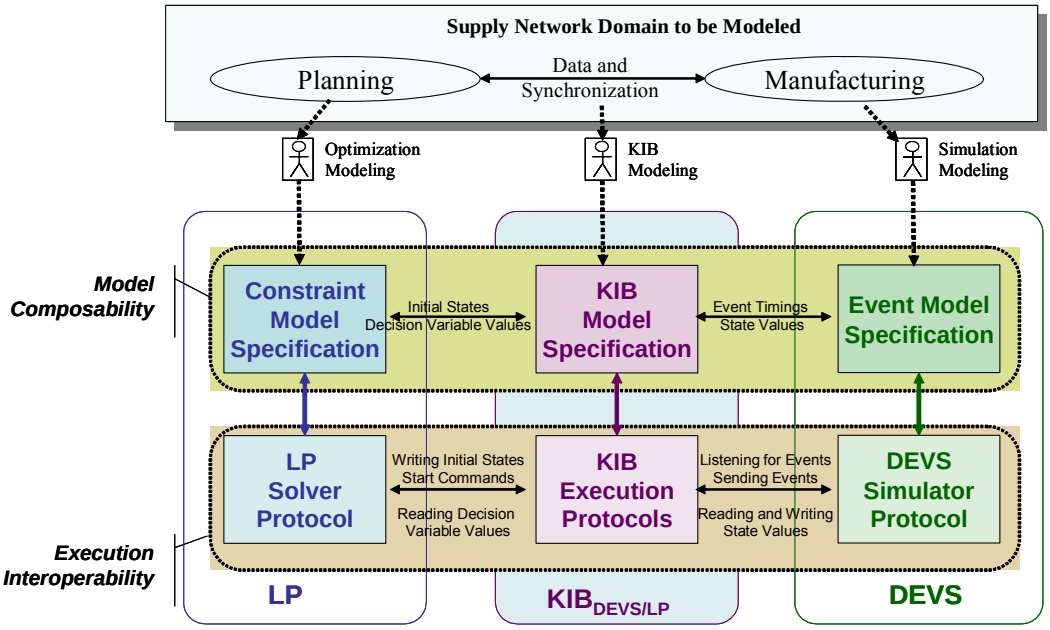


Figure 10. LP/DEVS KIB

3.1 Supply Chain Network Domain Model

A conceptual representation for partitioning a supply chain network system is illustrated in Figure 11. There is the planning and manufacturing systems, the data that flows between them, and the manner in which they are synchronized.

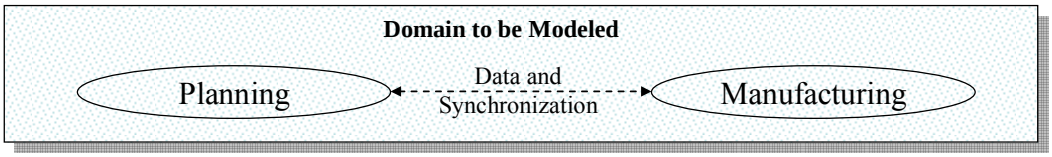


Figure 11. Supply Chain Network Partitioning

The planning system model includes the decision algorithms that will determine how much material manufacturing should build in a predefined time period. This time period usually coincides with some time interval such as work shifts, days, and weeks.

The manufacturing system model represents a physical segment of the supply chain network. The level of detail to model is in part determined by what level of planning is being modeled. For example, it could be modeled at a facility level for how much to start into a factory, or at manufacturing segments between points where material is stored for inventory planning. In both the planning and manufacturing case, the level of detail to specify is determined by the modelers and the actual problems they need to study.

An important piece to consider when partitioning these models is to determine the inputs and outputs for each type and how their execution will synchronize with the others. We need to consider how to map the output data from one model into the input of the other and if there is a transformation required to address a mismatch in the granularity or representation of the data. The frequency one model runs in relation to the other should also be determined. Specifically these things need to be addressed:

1. The data flows between the models and the data granularity.
2. How the execution of the models will be synchronized.

3.1.1 Supply chain network Domain Model Data Example

A minimal supply chain network shown in Figure 12 illustrates a possible integration for a Manufacturing Segment with an Assembly Starts Planner. The assembly starts planner looks at how much material is in the manufacturing segment and calculates what to begin for the next interval or period. In this example, the quantity of work in progress (WIP) in the assembly line is sent from Manufacturing Segment to the

Assembly Starts Planner as an input. The WIP quantity data from the Manufacturing Segment needs to undergo a transformation defined as function $g(m_{o,t})$ where $m_{o,t}$ is the manufacturing output at the beginning of period t. The Assembly Starts Planner will use the transformed WIP data to generate a schedule of what to start in the assembly line. The Assembly Starts Planner output needs to undergo the transformation $f(p_{o,t})$ before being input into the Manufacturing Segment. The transformations are formulated below:

- $m_{i,t} = f(p_{o,t})$ where
  - $m_{i,t}$ = manufacturing instructions received at end of period t for what to do in period t+1.
  - $p_{o,t}$ = planning output at the end of period t.

- $p_{i,t} = g(m_{o,t})$ where
  - $p_{i,t}$ = planning input at the end of period t.
  - $m_{o,t}$ = manufacturing output at the end of period t.

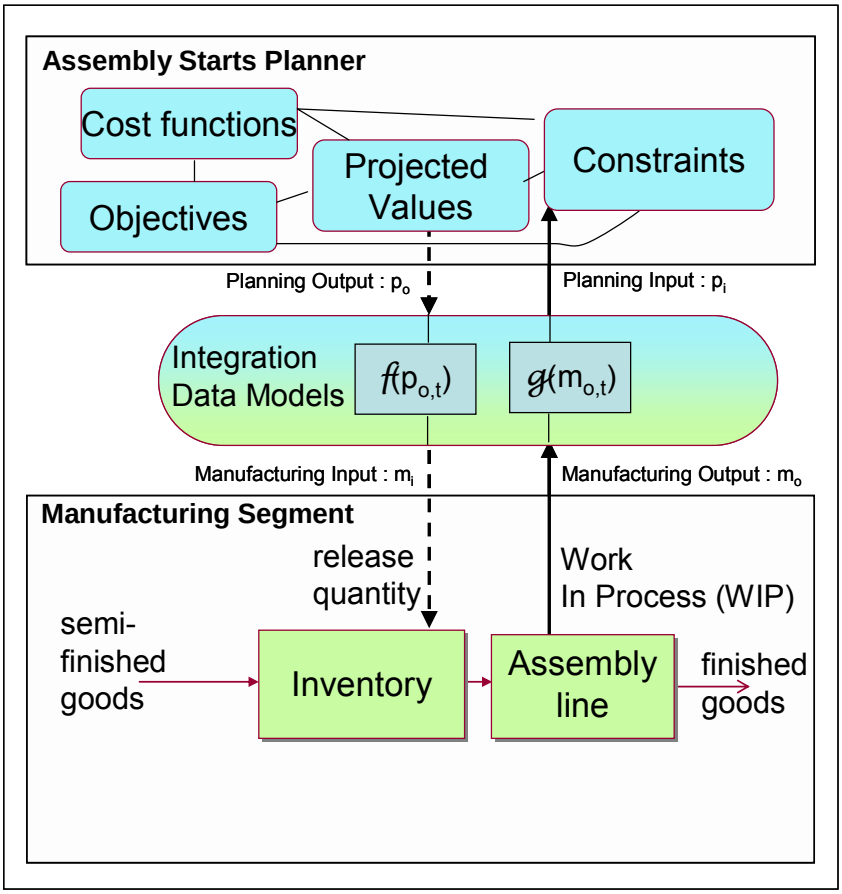


Figure 12. Domain Mapping and Transform Specification

This simple model illustrates how data mapping and transforms could be specified. This model could be expanded such that the functions have multiple inputs and/or outputs and also take into account historical values. The timing aspect will be addressed in the next section.

3.1.2 Supply chain network Domain Model Synchronization Example

Now we must consider how the models can be synchronized. A straightforward approach for synchronizing the planning and manufacturing components is shown in Figure 13. In this scheme, first the manufacturing runs for some period of time. At the end of the manufacturing period, the output data from the manufacturing model is input to the function $g(m_{o,t})$. Second, the function $g(m_{o,t})$ transforms the data as input to the planning policy. Third, the planning policy creates a new set of instructions for the manufacturing process. These instructions are then sent to the $f(p_{o,t})$ function which stores and transforms the data as input to the manufacturing process. Even though the same time instant t is used for $g(m_{o,t})$ and $f(p_{o,t})$, the executions for g and f are ordered. This cycle would continue for the duration of the model execution.

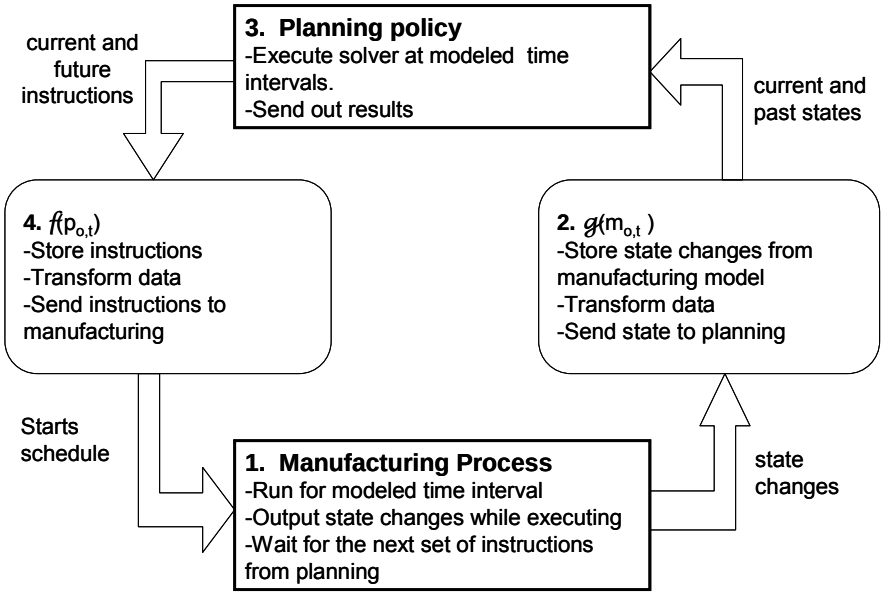


Figure 13. Supply Chain Network Simple Model of Synchronization

The synchronization requirement can have a significant impact on the complexity of the KIB. The types of synchronization needed between the different models can be considered to constrain what must be supported by the KIB specification. In the sequential case, a protocol shown in Figure 13 would only need to be supported. This sequential protocol can have a fixed execution frequency, though it can easily be changed to allow different execution frequencies. It is also possible to have an asynchronous execution scheme.

The requirements specified in the domain model largely determine what must be supported in the KIB. In our problem description in Chapter 2, we identified the need of experiments using different planning frequencies (hourly, weekly, ...) for a manufacturing process frequency. This means the KIB specification must support the change of frequency of one model execution in relation to the other. It also creates the requirement that time be introduced into the transformation functions.

The experiments show that the ability to specify the models to execute at different frequencies is very powerful. For example, a manufacturing simulation can be created that reports its hourly state. Experiments could be run to evaluate how a 12 hour schedule performs versus a daily schedule by specifying the planning policy to execute once every 12 or 24 hour intervals, respectively.

3.2 Supply chain network DEVS/LP KIB

The KIB enables a conceptual model to be partitioned and then modeled into distinct pieces. In our example shown in Figure 10, the supply chain network is partitioned into planning and manufacturing parts and a data and synchronization model

that connects the two. Each of these components needs to be mapped into their respective formalism. This is an area in which the KIB can be very powerful. Experts from each of the areas (planning and manufacturing) can be leveraged to reuse existing models or create new ones using best known approaches. These models can then be composed together into a semantically consistent multi-model using the KIB integration model. The KIB facilitates experimentation and simulation of different ways of doing this. Some actual industry case studies showing this are in the experiments section (Chapter 5).

As alluded above, to support this type of multi-formalism model composability, a number of capabilities must be provided. The capabilities required to support the LP / DEVS KIB will be described.

3.2.1 Formalism Composability with LP and DEVS

Modeling theory must be considered at the formalism level to enable the creation of composable models. The LP and DEVS formalisms are distinct and in general are targeted for special classes of problems. LP models are developed to find a good or optimal solution when many exist. DEVS models are developed to simulate behavior of a system composed of many independent parts with well-defined interactions over some time period.

The LP specification uses a mathematical algebraic modeling language that is comprised of an objective function, a set of constraints, and a set of variables. The LP is a static model in which a fixed initial state is used as the starting point for finding an

optimal solution. The search for the optimal solution is directed by an objective function and set of constraints. The solution is populated into decision variables that can be read after the search is complete.

The DEVS specification defines a set of states, input and output events, state transition functions, an output function, and a time advance function to describe the dynamics of a system. The DEVS specification explicitly accounts for time. The state values are available through external events and can be observed at any instant during the simulation execution.

To enable composability across these two types of models, we must ensure that the semantics of data and synchronization are consistent. This requires the mapping and transformation of algebraic LP data to and from the DEVS dynamic system specification language. For synchronization, the static solves of an LP must be coordinated with the dynamic execution of a simulator in a way that is semantically consistent with the intended behavior of the multi-model.

3.2.2 DEVS/LP KIB Formalism

A KIB for composing DEVS/LP formalisms would need to enable a consistent composition of both the specifications and execution protocols. This type of an approach would need to consider the execution protocols of the formalisms, how to transform the data between the two, and then enable the correct coordination between each of the execution protocols and data transformations. Figure 14 illustrates the DEVS and LP

data protocols, the data transformation, and the KIB synchronization that would need to be supported.

The LP reads an initial state of values, performs a solve function, and outputs results. The DEVS model has an internal control loop that manages the state transition, time advance, and output functions. The DEVS input and output are received via external events from coupled models. The KIB would need to coordinate the execution of LP solves with input and output events from the DEVS model and execute the data transformation functions at the correct time instances. The KIB would also need to continue the execution and coordination of the models for the duration of the composed multi-model. This implies that multiple runs of the LP may need to be coordinated with one single run of the DEVS simulation. For example, if the composed multi-model was set up to run a supply chain network for one week with a new schedule generated at the beginning of each day, the LP would need to run seven times to create a schedule for each individual day, while the DEVS model would need to simulate seven days of manufacturing. The KIB must coordinate the LP solves such that the initial state supplied is what the DEVS state is at the logical simulation time corresponding to the beginning of each day.

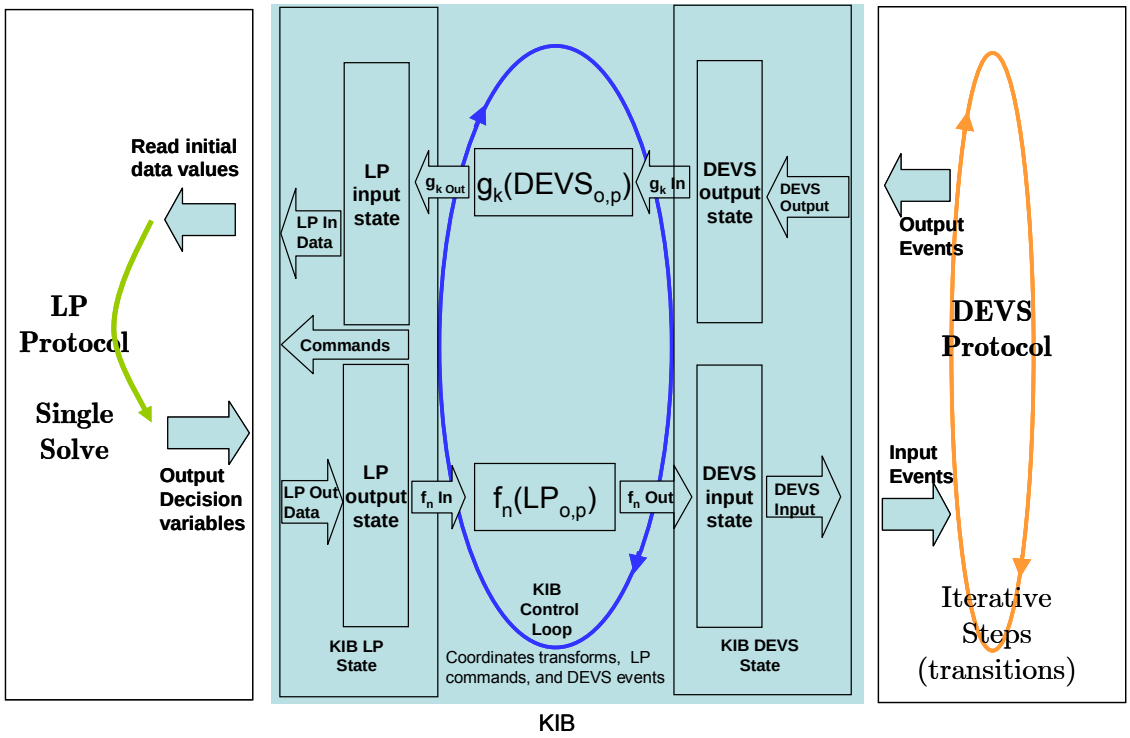


Figure 14. LP/DEVS KIB Interfaces and Protocols

Any given KIB approach to enable multi-formalism composition must address the following four items: data mapping, synchronization, timing, and concurrency between the models (Sarjoughian and Huang 2005). For the DEVS/LP KIB formalism in this research, they have been addressed as follows.

Data mapping is accomplished through state interfaces and transformation functions. The I/O for each of the modeling formalisms are mapped to and from a KIB interface state model. The transformation functions operate on one interface state model to generate content for the other interface state model. Each of the models interfaces and its associated mappings to and from the data transform functions can be modeled in the KIB specification.

Synchronization is supported through sequential execution of LP, KIB, and DEVS and is handled by the sync variable. Everything has to happen on time period boundaries, which is determined by when the sync variable changes value. To support interval-based sequential execution, the KIB model uses time intervals or periods. The time interval boundaries are determined by when a data variable changes value. This variable is called the sync variable. It is configurable and can be set to any variable that is output from the DEVS model. For this KIB, the DEVS is the only logical model to provide the sync variable. When the sync variable changes, a state transition in the KIB control model is called. At the start of each period, the data transformations, LP solves, and sending of DEVS events can occur.

Timing refers to data mappings and transformation consuming logical time. The data mappings just mentioned occur instantaneously (i.e., zero logical time) as compared with the DEVS state transition functions that consume some finite logical time period.

Concurrency refers to the KIB model executing in parallel with the LP and DEVS models. The KIB does not perform any of the actions concurrently since it serializes data transformations, DEVS events, and LP solves. Data transformations and LP solves are "blocking synchronous calls". The DEVS event handler for external KIB events has also been implemented as a blocking call. The simulation cannot continue until the KIB returns response to the event.

3.2.2.1 Model Specification

A LP/DEVS model specification enables defining the LP and DEVS interface inputs and outputs, the data transformation functions, and their synchronization (see Figure 14). The LP/DEVS model specification can be partitioned into five different areas. They are:

1. The specification of the LP input and output interface model
2. The specification of the DEVS input and output interface model
3. The specification of the LP➔DEVS transformations
4. The specification of the DEVS➔LP transformations
5. The specification of a KIB synchronization model.

3.2.2.1.1 Interface Model Specifications

The interface model specification enables the definition of which input and output data can be read and written for each interface. The KIB has a state model for each that can store current and previous values. The interface specification allows the modeler to specify which data elements to read and write from each model, what the data structure is, and how much historical data state to track (e.g., the modeler may want to keep track of the last seven values of material leaving the warehouse if they are going to configure a transformation that aggregates the last seven days into a weekly value). The KIB creates a set of state values for each data element (e.g. array entries or data fields from data objects) for current and historical values.

The LP input interface data model specifies the names and structure of the LP input data variables. The modeler could choose data being supplied from the DEVS➔LP transformations or they can hard code values in the KIB specification. For LP output, the

model would provide the names and structures of the decision variables the modeler needs to supply to the LP➔DEVS transformations.

For LP solvers these types of interfaces would typically be named matrices specifications. Each of the matrices would have a name and dimension. Each value in the matrices would map to a set of time ordered LP interface state values. If the KIB is interfacing to an optimization implementation that supports the OPL (Hentenryck 1999) language, the modeler may choose to specify the interface using data structures supported by this language.

The LP input and output interfaces can be described as:

$S_{LPOutputInterface} \subseteq x$

where $S_{LPOutputInterface}$ is the set of LP output data configured to go into KIB LP interface state model. The LP output data can be a set of the values x. The decision variable values (Figure 5).

$S_{LPInputInterface} \subseteq c$ or b

where $S_{LPInputInterface}$ is the set of KIB output data configured to go into LP solver. The input would be either c: the cost coefficients, or b: the set of constant values (Figure 5).

The DEVS interface specification enables the modeler to define which events to send and receive from the simulation. The specification also enables the data structure to be modeled to match the data object structure contained within the events. Each data element within the data object can be a state value in the DEVS interface state within the KIB. The set of allowable states that could be configured in the DEVS interface model would be:

$S_{DevsOutputInterface} \subseteq Y$

where $S_{DevsOutputInterface}$ is the set of DEVS output data configured to go into KIB DEVS interface state model. This output would be contained within the output events Y from the DEVS simulation (Figure 6).

$S_{DevsInputInterface} \subseteq X$

where $S_{DevsInputInterface}$ is the set of KIB output data configured to go into DEVS Semiconductor Supply Chain Network simulation. The data could be elements of X: the allowable input values to the DEVS simulation (Figure 6).

3.2.2.1.2 Transformation Specifications

The transform specifications enable the modeler to define how to map and transform data between the models. The transforms can read the state from one of the KIB interface state models (LP interface state or DEVS interface state in Figure 14) and write it to the other interface state model. The modeler can specify a data transform function and the input and output mappings to that transform. In Figure 14, the output of the DEVS interface state is made available to the LP interface state via function $g_k(DEVS_{O,t})$. Conversely, the LP output state is made available to the DEVS input state via function $f_n(LP_{O,t})$.

The LP➔DEVS transformations are defined as:

$\boldsymbol{fn_{out}} = f_n(LP_{O,t})$:
$LP_{O,t} \subseteq S_{LPOutputInterfaceStates}$ **and** $\boldsymbol{fn_{out}} \subseteq S_{DEVSInputInterfaceStates}$

The DEVS➔LP transformations are defined as:

$\boldsymbol{gk_{out}} = g_k(DEVS_{O,t})$:
$DEVS_{O,t} \subseteq S_{DEVSOutputInterfaceState}$ **and** $gk_{out} \subseteq S_{LPInputInterfaceStates}$

3.2.2.1.3 KIB Synchronization Model Specification

The KIB synchronization specification enables the LP or DEVS model frequency to be a multiple of the other, (e.g., the simulation model can run 24 cycles to every 1 LP cycle if the simulation is running hourly and the LP is running daily). A single LP iteration is one execution of the solver. For the DEVS model, the KIB requires a definition of what constitutes a DEVS iteration. For the KIB, this is accomplished by requiring the modeler to specify a DEVS synchronization data variable. The KIB will increment its DEVS cycle count when this variable is updated by an external DEVS event. The KIB maintains two cycle counters, the LP cycle counter and the DEVS cycle counter. They are denoted as:

$KIB_{DEVSCycleCount}$
$KIB_{LPCycleCount}$

$KIB_{LPCycleCount}$ will increment on each completed LP solve
$KIB_{DEVSCycleCount}$ will increment each time the $KIB_{DEVSSyncEvent}$ occurs
The $KIB_{DEVSSyncEvent}$ will occur when the synchronization data element changes.

The KIB model specification requires a synchronization data element be defined. The KIB synchronization element must be an element of the DEVS output data elements defined in the DEVS interface specification.

The KIB synchronization specification allows the modeler to define when the LP solver is called or when events are sent to DEVS using the $KIB_{LPCycleCount}$ and $KIB_{DEVSCycleCount}$ variables. The specification also allows the configuration of when data transform functions are called using the cycle count variables.

The execution frequency can be configured with the cycle count variables. (e.g. if there was a transform that should only update the LP data weekly, and the simulation was running hourly, you could model the data transform to execute once every 168 (hours in 7 days) cycle counts.

3.2.2.2 Protocol Specification

The protocol captures the behavior and semantics consistent with the LP/DEVS Model Specification. Figure 15 illustrates a system design for a Semiconductor Supply Chain Network LP/DEVS KIB. We will use this illustration to describe the protocol formulation.

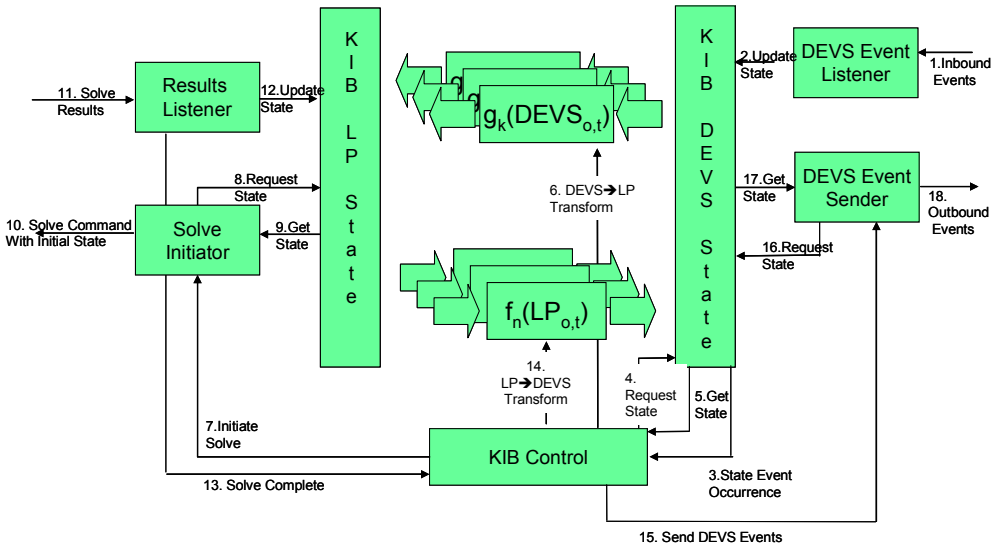


Figure 15. Semiconductor Supply Chain Network LP/DEVS KIB System Design

Input and output arrive at the KIB from the LP and DEVS models. The KIB sends and receives data from the LP via the Solve initiator and Results Listener shown in

Figure 15. Input and output to the DEVS model is via the Event Sender and Event Listener. For both models, the receiving components (i.e. Results Listener and Event Listener) can update the internal state of the KIB when events or results occur. The sending components (i.e. Solve Initiator and Event Sender) are initiated from the KIB control component. The KIB control component can initiate four different types of actions. They are LP solve, DEVS event, DEVS➔LP transformation function, and LP➔DEVS transformation function. The LP/DEVS KIB Model Specification is used to define how the KIB coordinates these actions.

3.2.2.3 KIB Control Model

The Semiconductor Supply Chain Network KIB control has an internal cycle counter that increments when the synchronization data element updates. The data element is updated using push pull logic. When an event is received from the DEVS model, the DEVS event listener updates the KIB DEVS state, which in turn notifies the KIB control component. The KIB control then pulls the new value from the KIB DEVS state.

The KIB synchronization model must define a synchronization data element, which can be any data element configured in the LP or DEVS interface specifications. It is up to the modeler to make sure the appropriate data signal is configured. For our DEVS Semiconductor Supply Chain Network models, we have a clock event that outputs at the end of each day. This is what has been configured in the KIB as a synchronization event. When the synchronization data element updates, the KIB control model will

increment its cycle counter and start a new control cycle. The control cycle is configurable to which actions take place and their order of execution. In Figure 16, the different types of control loops are shown.

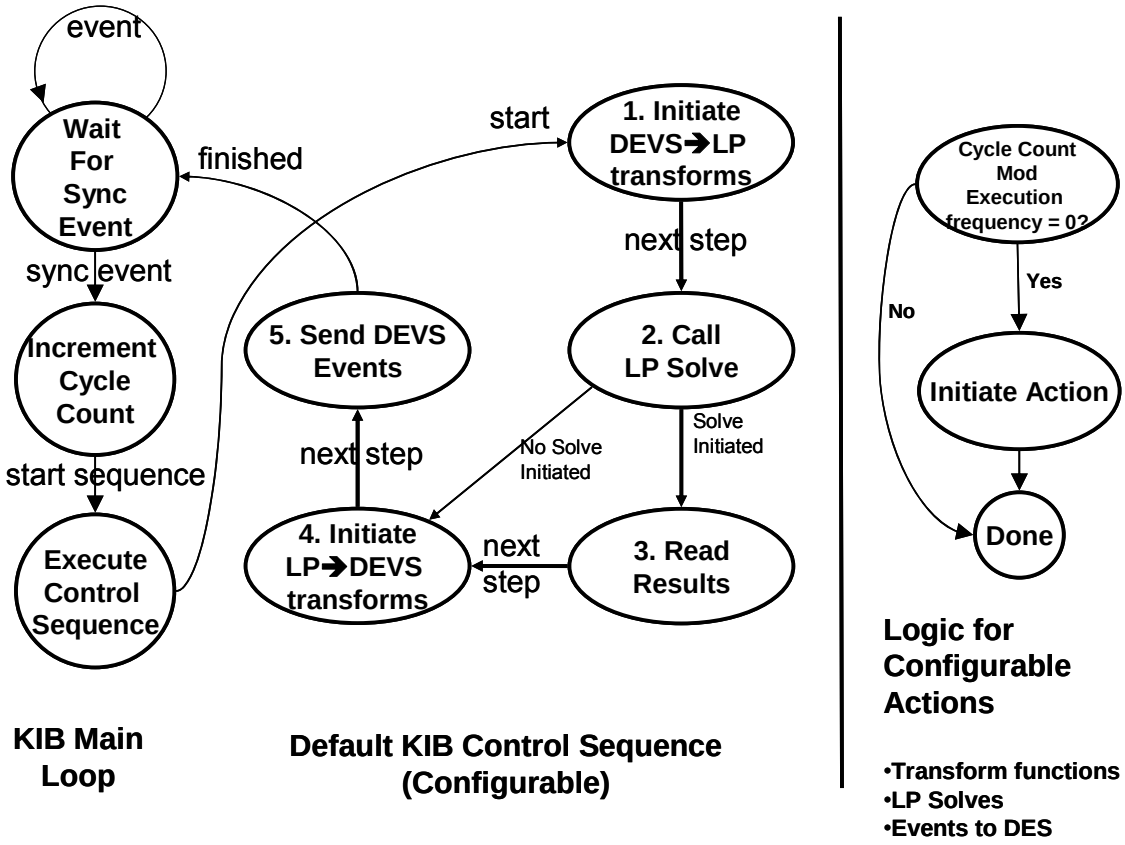


Figure 16. Default KIB Control Logic

First, there is the main control loop. This loop waits for a synchronization event to occur, then the cycle count is incremented, after which the control sequence is started. After the control sequence completes, it goes back to the *wait for sync event*. The *wait for sync* event would be supported by arrows (1,2,3,4,5) in Figure 15. An external DEVS

event would update the KIB DEVS state (arrows 1 and 2). The KIB DEVS state would notify the KIB control component of a state change (arrow 3). The KB control would check if the data element to update the cycle count changed (arrows 4 and 5).

The control sequence is started at the end of the KIB main control loop. This is default which could be overwritten in the KIB control model specification (see Figure 15 and Figure 16). The ordering of when to execute solves, initiate transform functions, and send events are configurable by the KIB models.

The numbering in Figure 15 corresponds to the default control loop. In this scenario, first, the DEV➔LP transformation functions are initiated (arrow 6). There may be multiple occurrences of these functions, each of which can be configured to execute on every nth cycle. Second, the LP solve is initiated (arrow 7); it may be initiated every time or at some lower frequency multiple. If no solve is initiated, the control loop jumps to step 4, otherwise it proceeds to step 3. If a solve was initiated, the initial state from the KIB LP state model needs to be read (arrow 8,9), followed by a call to the LP solver (arrow 10). Then the control loop waits for the results (arrow 11) and reads them into the KIB state model (arrow 12). The KIB control is notified that the solve is complete (arrow 13). Next, the LP➔DEVS transformation functions are initiated (arrow 14). And finally DEVS events are sent out (arrows 15,16,17,18).

The data transformation functions and LP solves are configurable actions. The logic for configurable actions is shown in Figure 16. For the Semiconductor Supply Chain Network DEVS/LP KIB, they can be configured to execute on every cycle, or some lower frequency multiple. (e.g. If cycles are occurring every hour and updates are

only needed once a day, the configurable action can be modified to execute once every 24 cycles).

The LP/DEVS KIB protocol for the class of Semiconductor Supply Chain Network models is defined as follows:

Initialization

1. *At initialization, the DEVS model reports the initial state of the supply chain network topology.*
2. *The KIB executes the DEVS➔LP function(s) for writing the initial DEVS output state into the LP initial state.*
3. *The KIB initiates an LP solve.*
4. *The LP generates the initial future schedules for the DEVS model to use.*
5. *The KIB executes the LP➔DEVS function(s) to write the LP schedule to the DEVS model.*

Control Loop

6. *KIB waits for DEVS to update its output state (arrows 1 and 2 in Figure 15).*
7. *The KIB examines DEVS output state and DEVS/LP control model frequency to determine if any DEVS➔LP should be executed. It executes all functions that are scheduled (arrows 3,4,5,and 6 in Figure 15).*
8. *The KIB checks if an LP solve should be initiated by performing the configurable action check illustrated in Figure 16. If true, the KIB initiates an LP solve and waits for the results before going to the next step (arrows 7,8,9,10,11,12,and 13 in Figure 15), otherwise, proceed to next step.*
9. *KIB looks at simulation output state and control model to determine which LP➔DEVS functions to execute (arrow 14 in Figure 15).*
10. *KIB sends events to the simulation which will enable it to complete another interval (arrows 15,16,17,18 in Figure 15).*
11. *Go to beginning of control loop – step 6.*

What makes this protocol specific to our Semiconductor Supply Chain Network models is:

- The initialization protocol requiring the DEVS model to report the initial state before starting anything. Other types of problem domains may need the solver to run first.
- Control is only being synchronized on the DEVS state. The control could be generalized based on the LP state synchronization.
- The LP solve and transformation functions run at time boundaries on a DEVS iteration.

3.2.2.3.1 Protocol Execution

Two types of external occurrences can initiate internal KIB actions, an external event from the DEVS model or the completion of a solve from the LP model. The algorithm for the occurrence of an external DEVS event is:

On DEVS event

1. Execute DEVS state update transition function.
    - *This function will update the DEVS state if the event was modeled in the model specification file (arrows 1,2 in Figure 15).*
2. Update time advance.
    - *Check if state updates meet model configuration requirements to update KIB time advance (arrows 3,4,5 in Figure 15).*
3. Execute DEVS➔LP data transformation function.
    - *This function will iterate through all the DEVS➔LP transforms configured in the model specification and execute it if their time advance = current KIB control cycle (arrow 6 in Figure 15).*
4. Execute check solve function.

Check if time advance for solve = current KIB cycle
If TimeAdvance = current KIB cycle

- *Execute the LP initiate solve function (arrows 7,8,9 in Figure 15).*
    - This function populates an LP coefficient model using current states from the KIB LP State model.
    - Send command to solver with LP coefficients (10 in Figure 15).

Else

- *Execute LP➔DEVS transforms transition*
    - This function will iterate through all the LP➔DEVS transforms configured in the model specification and execute it if their time advance = current KIB cycle (arrow 14 in Figure 15).
- *Send update events to DEVS*
    - Will create events scheduled to be sent at the current time to the DEVS model using the current KIB DEVS state (arrows 15,16,17 in Figure 15).
    - Send the events to the DEVS model at the proper time (arrow 18 in Figure 15).

The algorithm for when an LP solve completes:

On LP solve complete event (13 in Figure 15).

1. Execute LP➔DEVS transform transition function.
    - *This function will iterate through all the LP➔DEVS transforms configured in the model specification and execute it if their time advance = current KIB cycle time (arrow 14 in Figure 15).*
2. Send update events to DEVS
    - *Will create events scheduled to be sent at the current time to the DEVS model using the current states from the KIB DEVS state model (arrows 15,16,17 in Figure 15).*
    - *Send the events to the DEVS model (arrow 18 in Figure 15).*

A software design class diagram showing basic software components and their relationships highlighting the interactions among the major KIB components can be found in Chapter 6.

While developing the DEVS/LP protocol, some parts may be modified based on the LP planning and DEVS manufacturing models. For initialization, it will be assumed an initial state already exists. This choice is implementation specific and therefore may be redefined based on different requirements. For configuration of KIB, it will be assumed that either model state can be used to initiate KIB control actions.

It is also assumed that the DEVS simulation will wait for KIB to send events. In order for DEVS models to support this, they would need to have an atomic or a coupled model that listens for external events outside of DEVS. This would require the implementation of a DEVS model that can communicate externally from the DEVS implementation. The external communications would need to be enabled using interoperability techniques for interfacing with the KIB.

Likewise, interoperability techniques would be required for connecting a KIB implementation to an LP solver. The right and left boundaries of Figure 15 is where connections to the KIB implementation via interoperability techniques would be required.

3.3 DEVS/LP Model Specification

A Semiconductor Supply Chain Network LP/DEVS KIB enabled the composition specification of LP/DEVS models and experiments described previously. The formalism follows the methodology described in the previous section.

The specification consists of:

1. *Model interfaces in terms of their native data structures.*
2. *Interface relationships using mapping and transform functions.*
3. *Model synchronization in terms of how each will be executed in relation to the other.*

3.3.1 Mapping and Transform Example

For this section we will use the example problem shown in Figure 17. A small semiconductor manufacturing topology consisting of two fabrication factories, two assembly warehouses, and two semiconductor Assembly Test (AT) sites are being controlled by a wafer shipping decision system. The factories can ship their products to the two assembly warehouses. Material from the warehouses can be released into semiconductor assembly tests. The amount and timing of the product released from the assembly warehouses is determined by schedules of what to start from the associated assembly test site.

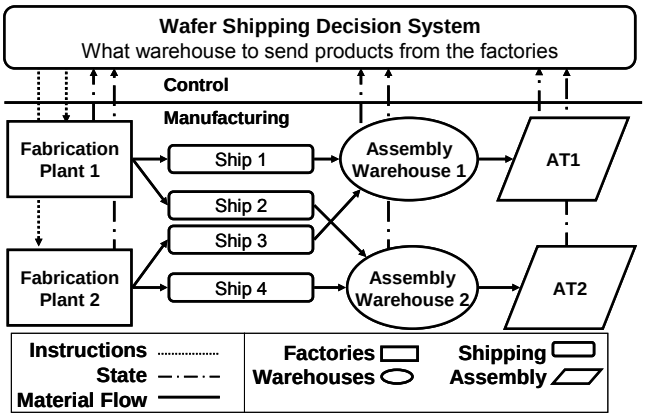


Figure 17. Example Problem

A controller is connected to make decisions on routing material leaving the fabrication plants. The objective of the controller is to keep the warehouse inventory within upper and lower control limits. Input states to the controller are the product that was shipped, warehouse inventory levels, forecasted builds of the fabrication plants, and forecasted starts from the assembly test sites. The output of the controller are commands that dictate the quantity of each product to be shipped from a given fabrication factory to the assembly test warehouses. The output commands need to specify what product to release, where it should be shipped, and the quantity.

The manufacturing topology can report how much WIP is in the fabrication plants, how much material is in shipment and how much inventory is in the warehouses, as well as the planned starts for the assembly warehouses. The data system reports the material states in terms of lots. A lot is a batch of units that are processed together in the manufacturing flow. A lot has a certain quantity of units of a particular product. We can define a lot structure as follows:

```
Lot Structure
   Name: Unique identifier
   ProductName: String
   Quantity: Integer
```

The controller would need to output a matrix of quantities for each factory that specifies how much to release of each product and where it should be shipped. For the model shown in Figure 17, there would be two output matrices, one for each factory.

An example of how this matrix could look is shown below (1). The matrix is a $3 \times n$ vector that specifies a product number, a destination warehouse number, and a quantity where n is the number of product and destination combinations.

$$\begin{matrix} \text{Product} \\ \text{Destination} \end{matrix} \begin{bmatrix} P_1 & \cdots & P_n \\ D_1 & \cdots & D_n \\ Q_1 & \cdots & Q_n \end{bmatrix} \quad (1)$$

The product number is mapped to one of the products built by the factory. The destination number specifies which warehouse the material should be shipped to and the quantity specifies how much.

3.3.1.1 Data Transforms

The control model requires data to be input in terms of units. For some types of data such as factory shipments, the quantity of product that leaves must be aggregated over a controller interval. For example, if the controller interval is one day, the quantities from all lots that left the factory in the previous day would be aggregated into a single value for input into the controller on the start of the current day. The value is calculated as shown in equation (2):

$$f(p,d) = \sum_{lot \in LotsOut} lot_{quantity_{p,d,t}} \quad (2)$$

where $p \in$ *products,* $d \in$ *destinations,* $previousControlTime < t \leq currentControlTime$.

If the controller interval is daily and the manufacturing model runs hourly, the controller instructions need to be disaggregated. For the mapping of the controller release to the simulation, let's assume that the value needs to be divided equally over each of the simulation time intervals. For example, if the simulation is running at an hourly granularity and the controller is generating instructions once a day, the controller instruction would be divided by 24. In general, the disaggregation could be more complex, but for illustration this simplified algorithm will be used. The equation for the equally divided disaggregation is:

$$g(p,d) = cr_{p,d,q} * \left(\frac{cf}{sf} \right) \quad (3)$$

where g(p,d) is factory release quantity for product p going to destination d, cr is controller release quantity, cf is controller frequency, and sf is simulation frequency.

3.3.1.2 Mapping between Vectors and Events

We must now consider how the data and control will be transferred between the two formalisms. For the discrete event simulation, we must read and write events to a running simulation. For the controller, we must populate input variables, initiate a solver run, and then read the output variables.

Suppose the simulation is running at hourly granularity and the controller is generating instructions once a day using the format shown in (1). Also assume that each factory can build two products and there are two possible destinations for each product. The LP modeler could write the equations in a manner that would format the output as shown in (4). The top row specifies product numbers, the middle row denotes the

destinations, and the bottom row indicates the quantity to release. Each column defines three values: product, destination, and quantity.

$$\text{Factory1Ships} = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 2 & 1 & 2 \\ 1000 & 1000 & 500 & 250 \end{bmatrix} \quad (4)$$

The controller output instruction needs to be mapped to simulation input events, which have the following structure:

```
ReleaseEvent(
   SimulationPort,
   Data(product, destination, quantity))
```

The simulation port specifies where the event should be directed. The data has three elements: what product this event is for, which destination warehouse the product should be shipped to, and the quantity to transport.

To accomplish the mapping and transforms, we need to first consider which entity the controller output matrix is for. Since the name of the matrix is `Factory1Ships`, we can assume it is for `factory1`. This must be explicitly mapped in the KIB integration model. The controller output matrices with name Factory1Ships will be mapped to DES events going to the release input port for factory1. Next, each column of the matrix needs to be transformed into simulation input events. Each matrix element [column i, row j] is a positive scalar value (e.g., Factory1Ships [0,2]=1000). We will

work through the first column in matrix (4). This column shows that 1000 units of product 1 should be sent to warehouse 1. Let's assume that both the controller and simulation use the same number scheme for products and destinations. Since the controller is running once a day and the simulator hourly, we will need to generate 24 simulation events for each hour of the simulation on the day under consideration. The quantity will need to be transformed using equation (3). The transformed quantity value for the first column in (4) would be 1000 * 1/24. The simulation data event values in the 24 generated events would be: data(1,1,1000/24). The mapping between structures would look like:

```
ReleaseEvent(Factory1ReleasePort,
Data(Factory1Ships[0,0],
     Factory1Ships[1,0],
     g(Factory1Ships[0,0],
       Fatory1Ships(1,0])))
```

Referring back to section 3.3, the specification consists of model interfaces, interface relationships, and the synchronization model.

3.3.2 Model Interfaces

This part of the specification enables the modeler to identify the names and data structures of the available input and output data to each of the models. All the data structures required for the interfaces need to be specified.

3.3.2.1 DEVS interface specification

Data of interest from a DEVS model is contained within external events received from atomic or coupled models. Depending on the implementation of the simulator and interface, the states could be read and written during a running simulation or the written states could be populated as an initial state to the simulation, the simulation run for one time period, and output states read at the end of the time period. Either way, the same elements must be modeled.

It is also important to specify at what time instant the state is read or written. For the Semiconductor Supply Chain Network LP/DEVS formalism, the timing of when to read or write a state is determined by the KIB Control model (see Figure 16). This will be discussed in more detail below.

For our formalism we need to map all the data associated with a DEVS external event. An external event is tied to an input or output port on an atomic or coupled model, has a name, can contain data, and happens at a specific time instant. Table 1 shows how these data elements are mapped into the KIB Data Model. The mappings are configurable in the model specification that the KIB provides. The DEVS port and event

names are mapped to a single message name in the KIB. If the port and event names differ, then a multi-part message name is required.

DEVS Data Elements	KIB Data Model
Type of Port (input or output)	Direction: Input or Output
Model (atomic or coupled)	Module Name
Port Name	Data Name
Event Name	
Time of occurrence	Timestamp
Data Object Structure	Data type specification

Table 1. DEVS Data Element Mappings to KIB Elements

The mapping into the KIB specifies all the data elements that are available to be populated with values while the models are being executed. When the models are executed, the external events received from the DEVS model has each of its data elements mapped into an instance of a KIB DEVS state value. The control will determine how many historical records are stored for recurring events of the same type in the KIB DEVS state.

3.3.2.2 LP interface specification

An LP model has an initial state populated with data values and a set of variables that can be read as output or written to as input. Inputs, outputs, and state variables can be multi-dimensional arrays with values constrained to positive real numbers.

The mapping between the LP data elements to the KIB data model is shown in Table 2. The data can be input for the LP (initial state) or output (decision variables). The LP has no mapping to the KIB module name since it is not a component-based

specification. All LP data variables could be placed into a single module or logically grouped by the modeler. The name of the vector or matrix maps to the KIB data name. The time of the solve maps to the KIB timestamp field and the matrix or vector definition is stated in the data type specification.

LP Data Elements	KIB Data Model
Type of Parameter: Initial state(input) or decision variable (output)	Direction: Input or Output
N/A	Module Name
Matrix or Vector name	Data Name
Solve Time	Timestamp
Matrix or vector bounds definition	Data type specification

Table 2. LP Data Element Mappings to KIB Elements

3.3.3 Interface Relationships

Three different items must be modeled in the interface relationship:

1. Mapping: specifies which LP input states map to the DEVS output states and vise versa.
2. Transforms: specifies what function to apply to the data being written from one model output to the other's input.
3. Timing: specifies when the data should be read from one model output, transformed, and written to the other model input.

Mapping: KIB interface output variables can be assigned to transformation function inputs. Similarly, transformation function outputs can be assigned to KIB interface inputs.

The timestamp on the output is written by the KIB control when an event arrives or when a solve completes. The timestamp on the input data (this is input to the external model from the KIB) is written when the KIB sends an event or calls the solver. A timestamp from the output can be configured to be written to the input. If an output timestamp is mapped to the input timestamp, this means the input should occur simultaneously with the output.

For mapping, the data variables are addressed by the values in the *Module Name* and *Data Name* fields. Data values in the *Module Name*, *Data Name*, *Timestamp*, and *Data fields* can also be mapped to transformation inputs (see Table 1 and Table 2). Some rules for mappings:

- Output variables must always be mapped to transformation function inputs.
- Transformation function outputs can only be mapped to input variables.
- Transformation functions can have 1 to *n* inputs.
- Transformation functions can have 1 to *n* outputs.

Transformation: The transformation logic enables the interface to transform one model-type data representation to another model-type data representation. Data can be scaled, have primitive type conversions applied, be selected from multiple values, and be aggregated or disaggregated.

The set of available values from the source model for the transformation are all the current and historical data values in the KIB state associated with the source model.

Transform Types:

- Scaling: multiplier applied to value
- Type Conversions: Integers to Floats, Floats to Integers, Strings to Number, Numbers to Strings
- Value Select: Minimum, Maximum, Oldest, Newest
- Aggregation: Mean, Median, Sum
- Disaggregation: Division, Netting (only 2 for Semiconductor Supply Chain Network)

Scaling: Many times, a planning algorithm may be modeled using calculations in kilo units, or mega units. In this case, if the data originating from the simulation is in terms of actual units, the quantities will need to be divided by 10^3 or 10^4 respectively. Correspondingly, the data from the planning algorithm will need to be multiplied as well.

Type Conversions: The LP will always require positive real number input. However, the DES can use Reals, Integers, or Strings. Type conversions may be required.

Selections: A single value is selected from a set of possible values. Some common types of selections are minimum value, maximum value, oldest, and newest. Selections can have a time dependent attribute. Figure 18 shows a case in which the indexed value to reference in an array is dependent on current logical time. The example shown could happen if the LP generates a schedule once a day for the next three eight-hour shifts. The corresponding values are then passed into the DEVS model when the logical time for the next shift start occurs.

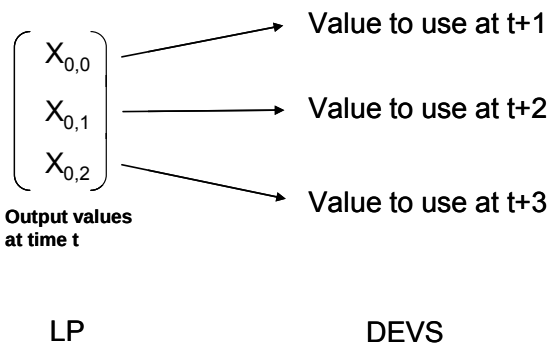


Figure 18. Time Dependant Mapping

Aggregation: Aggregation is a class of transforms that calculates a single value from a set of multiple values. Figure 19 depicts different ways in which data can be aggregated, including across current values, across historical values, or both. The number of historical values available is dependent upon how often the model is being synchronized in relation to the other model. For example, if the simulation is giving hourly updates and the LP is giving daily plans, there will be 24 simulation data records available on every LP run. It should be noted that although arbitrary operations can be carried out on matrices in terms of time and data, only a subset of these operations are logically well defined. In particular, aggregation across time periods must be ensured to be consistent with the KIB protocol.

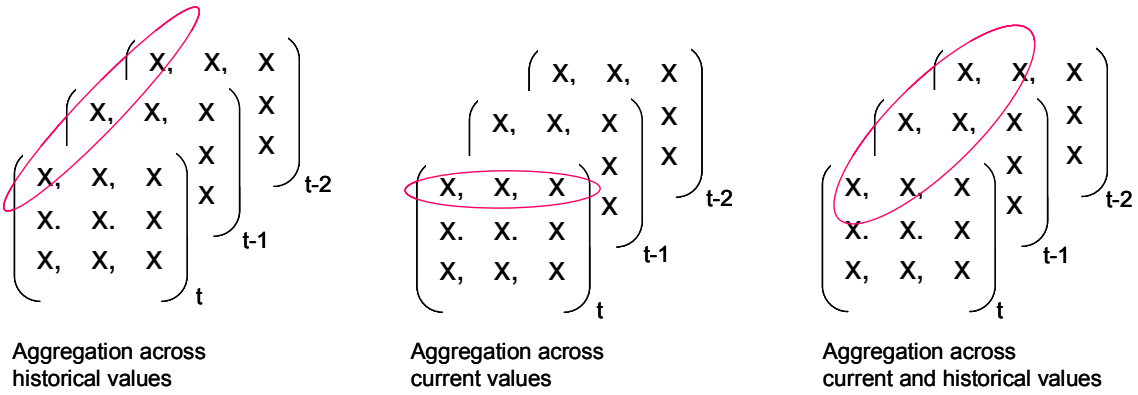


Figure 19. Types of Data Aggregation

Disaggregation: A common application for disaggregation in our Semiconductor Supply Chain Network domain is how to break down a planning schedule into lower granularity buckets. For example, how would a weekly schedule value be divided into daily starts schedules? Disaggregation algorithms can be modeled to do this. A very simple algorithm would divide the weekly schedule by seven. In reality, this type of schedule disaggregation could cause problems if manufacturing was building multiple products and the setup time for switching to a particular product was long.

Other approaches known as netting algorithms can be used. Netting simply adds or subtracts the over and under builds from the previous time period. Whether the netting is put in the interface function or in the models themselves should be a modeler's choice.

Disaggregation could also be tied closely with timing. Take the example in which the LP is creating a new schedule once a week, the simulation is modeled to run on daily schedules and the actual execution of the simulation is at an hourly granularity. The LP

schedule would need to be divided by seven to get a daily value, then that daily value would need to be sent to the simulation once every 24 hours.

Timing: The timing of when to execute the data transform functions is configurable. The default timing is to execute the transform once every DEVS/LP cycle. However, the execution can be configured to only run one time every *n* cycles, where the value of *n* can be modeled.

When the synchronization frequency is changed, it will impact when the data between the models will be sent and the aggregation/disaggregation data transformations executed. For example, if running the controller daily and simulation hourly, the quantity of material in all the lots that have left the factory in the last 24 hours need to be summed up. If the control frequency is changed to once a shift (8 hours), then we would only need to sum the material over the last shift. Conversely for the factory release commands, in the daily control cycle, 24 events would need to be sent. In the once a shift control scenario, only eight events were sent per control cycle. Figure 20 illustrates the data transform considerations of running a daily solve against an hourly simulation.

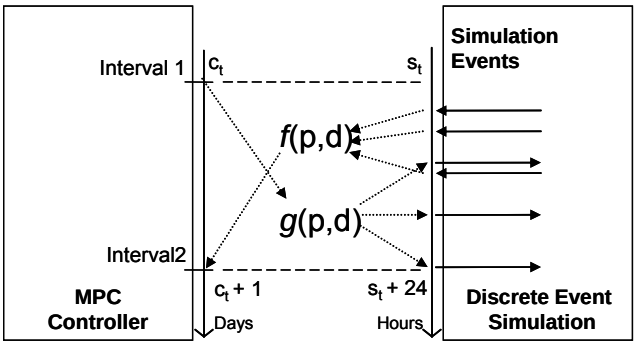


Figure 20. Transforms over Different Time Scales

3.3.4 Synchronization Model

The synchronization model allows the specification of when to run the LP model in relation to the DEVS model. The models can run in frequency multiples of each other. For example, if the simulation is running hourly, the LP could also run hourly, once every 2 hours, once every 3 hours, and so on.

Synchronization is supported through a data value received from one or the other models. The specification allows the definition of which variable to use, and when to execute each model based on changes of this data variable. This scheme requires that one of the data values received by either the LP or DEVS model be used for execution synchronization.

For example, if the DEVS model had a data variable named *iteration number*, the KIB could be configured to update its cycle counter every time the DEVS *iteration number* changed. The KIB could also be modeled to execute either of the models on multiples based on this data variable being updated. If we wanted the LP to generate a new plan daily, it could be set to run once every 24 times the DEVS *iteration number* changed.

Any of the data transformation functions can be modeled to run on multiples of the KIB cycle count. DEVS➔LP transformation functions can be executed on every KIB cycle or on some multiple of that cycle.

3.3.5 Transformation Functions For the LP/DEVS KIB

The transformation functions have been categorized into two types. There are data mapping transformations and data value transformation functions. The data mapping transformation functions enable complex data types (arrays, objects, sets) to be mapped to different complex or simple (integer, float, string) data types. The data value transformation functions can change the values within the data elements.

The inputs and outputs to data mapping transformation functions can be single or multi valued. By default, data mapping transformation functions will not change data values; it will only copy the data from one data structure type to another. However, data mapping transformation and data value transformation functions can be combined in one interface relationship. For example, the data mapping transformation function that copies an array to an unordered set could include a data value transformation function that scales each value before writing it to the set data structure.

In the following two sections, we will describe the mapping and data transformation functions needed to implement our DEVS/LP KIB.

3.3.6 Data Mapping Transformation Functions

UnorderedSetToArray: This mapping copies an unordered set of tuples to an ordered array. One of the data fields in the set must be specified as an index field. The transform uses the index data values to carry out the ordering.

ArrayToUnorderedSet: An array of values is copied to a set of unordered tuples. The array index value can be copied to a set tuple field if the array is part of the structure,

then the other fields in the structure will be copied as well. For example, if you have a structure that contains a field for a product name and another field that contains an array of *n* values, a set can be created with *n* tuples where each contains the product name, one of the array values, and the index for that array value. The starting value for the index can is customizable.

ArrayValueToVariable: This mapping copies a specific array value to a single variable. The array index needs to be specified. Also, if the array is part of a structure, a key field with its matching data value can be indicated. An example of such would be a structure that contains a product name and an array of values. The array entry 3 of $product_X$ could be specified so it will be written to this variable.

VariableToArrayValue: This mapping copies a variable value to a specific array entry. An index number needs to be denoted. Also, a key entry can be specified for arrays that are contained within a structure.

SetFieldValueToVariable: This mapping copies a specific field value to a single variable. A key field needs be specified to determine which tuple from a set to use. For example, if a set of tuples each contain a *product name* and *quantity* value, and the quantity for $product_x$ needed to be copied to $variable_{x,}$, the *product name* needs to defined as key to distinguish from other *product name*, *quantity* tuples.

VariableToSetFieldValue: This transform copies a variable to specific field in a set tuple. The field name in the tuple must be specified along with a key value.

Copy: The values of the fields from one model are copied into variables in the other model. The names do not need to match, merely the data type. For example, integers can only be copied to integer fields.

Copy Exactly: The structure and data is copied to an identical structure in the other model. Names of variables and their values are maintained in the transfer.

3.3.7 Data Value Transformation Functions

This section lists the data transformations provided by the KIB modeling language.

FloatToInteger: This transforms the data value from float to integer. A rounding algorithm of floor, ceiling, or round must be specified.

IntegerToFloat: Converts an integer value to a float.

AssignValue: Assigns a value that is configured in the KIB model to a data field. This is a static value that cannot change during the execution.

Aggregations (Mean, Median, Min, Max, Sum): These transforms aggregate multiple values into a single value. The aggregation can be for all values in the current time period or for multiple time periods. If data values are in arrays, the aggregation can return an array where the entries are aggregated from multiple arrays.

Disaggregation: Different types of disaggregation can be supported. A general purpose disaggregation is to divide the source value into equal target values. The design of the KIB enables extensions for customized disaggregation algorithms.

Scale: Multiplication or division operations can be specified to scale the data values.

3.4 Control Schemes

The KIB is required to support a synchronization model that enables the KIB control to run in multiples of the simulation. Experiments were designed with daily, shiftly, and hourly control with a simulation that could run at an hourly granularity. The KIB configuration model allows the execution of either model to run in multiples of the other.

To provide this capability, the KIB must coordinate the timing of simulation input/output events with the solver execution. It also needs to execute aggregation/disaggregation transforms across the correct time intervals. The ability to adjust the aggregation/disaggregation of data based on execution frequency is a key enabler of experimentation at different control frequencies. For example, if trying daily control against hourly data requires the aggregation of all events that occurred over the last 24 hours of logical simulation time. If trying shiftly (eight-hour shifts) control, then only aggregate over the last eight hours for logical simulation time.

A modulo function is used to model when solves and data transforms are to occur. A frequency number can be configured against one of the cycle counts. If the module returns zero, then the action will take place. The specification is as follows:

Execution Frequency for $X = X_{ef}$

Control Algorithm:

If (X_{ef} Modulo CycleCount = zero)
Take action
Else
Do nothing.

Where Cycle Count in ($KIB_{DEVSCycleCount,}$ $KIB_{LPCycleCount)}$

3.5 KIB Specification Implementation in XML

An implementation of the KIB specification was created using XML. The XML language provides the capability to create a structured language that is easy to parse and store in computer memory using off-the-shelf software tools. A KIB implementation was created that could read this XML language, which was used to support the case studies described in Chapter 5.

We will illustrate the LP/DEVS KIB XML modeling language using the Supply chain network model illustrated in Figure 17 and the example described in 3.3.1. The model in Figure 17 depicts status messages being sent from the fabrication, assembly warehouse, and AT entities to the planning model. The planning model sends commands back down to the fabrication entities.

Our XML implementation requires that an LP/DEVS KIB model be a "well formed" XML document. Any KIB implementation can then use standard XML tools to parse the contents of the model. Since the model must be a well formed XML document, it requires a root element. We have defined the root element as: <KIBMODEL>. The opening and closing tags of this model would be:

```
<KIBMODEL>
….
</KIBMODEL>
```

The LP/DEVS KIB modeling language provides a hierarchy that allows the model to be decomposed into modules. Within each of the modules, the interfaces for each of the models can be specified along with the relationships between them. The module tags give the modeler an option to separate the KIB model into logical parts. The opening tag for a module specification is:

```
<MODULE_SPECIFICATION name = “entity_name”>
```

A KIB model requires a minimum of two modules, one for specifying the data interfaces, mappings, and transforms and another to specify the control synchronization. However, the modeler may want to use many more modules to keep the model easy to understand. A logical set of modules to model for the example Supply chain network problem Figure 17 is shown below in Example 1.

```
<KIBMODEL>
<MODULE_SPECIFICATION Name="FABRICATION_1">…</MODULE_SPECIFICATION>
<MODULE_SPECIFICATION Name="FABRICATION_2">…</MODULE_SPECIFICATION>
<MODULE_SPECIFICATION Name="ASY_WH_1">…</MODULE_SPECIFICATION>
<MODULE_SPECIFICATION Name="ASY_WH_2">…</MODULE_SPECIFICATION>
<MODULE_SPECIFICATION Name="AT1">…</MODULE_SPECIFICATION>
<MODULE_SPECIFICATION Name="AT2">…</MODULE_SPECIFICATION>
<MODULE_SPECIFICATION Name="Sync">…</MODULE_SPECIFICATION>
</KIBMODEL>
```

Example 1. High Level Module Specification for the KIB Model

In this example, we have created seven different modules. There is a module corresponding to each of the entities that send and receive messages to the planning

model and an additional “sync” module. We will define the data interfaces and transformations for the entity modules. For the sync entity, we will define how the LP and DEVS models are synchronized.

At the next level there are interface specifications and interface relationships. How these are specified within KIB modules are shown in Example 2. There are the DEVS and LP interfaces along with the interface relationship. The XML tags are shown in Example 2.

```
<MODULE_SPECIFICATION Name="ASY_WH1">
<LPINTERFACE>…</LPINTERFACE>
<DEVSINTERFACE>…</DEVSINTERFACE>
<INTERFACE_RELATIONSHIP>…<INTERFACE_RELATIONSHIP>
</MODULE_SPECIFICATION>
```

Example 2. Specification for a Module

The DEVSINTERFACE and LPINTERFACE defines the input and output data available to the KIB from the DEVS and LP models. The interface relationship defines the mappings and transforms available. An example of an input and output DEVS interface for the FABRICATION_1 specifications is shown in Example 3.

```
<MODULE_SPECIFICATION Name="FABRICATION_1">
<DEVSINTERFACE>
<DataInput Name="Release_FABRICATION1">
      <Type>
            Collection,Record,
            Key:String:product,
            Int:Quantity,
            Key:String:Destination
      </Type>
  </DataInput>
…
<DataOutput Name="Outs_FABRICATION1">
      <Type>
            Collection,Record,
```

```
            Key:String:product,
            Key:String:Destination,
            Int:Quantity
        </Type>
 </DataOutput>
 </DEVSINTERFACE>
…
</MODULE_SPECIFICATION>
```

Example 3. DEVS Input and Output Variable Definitions

The variables have a name and a type string. The name is used to map the data variable to the corresponding DEVS event. The KIB implementation needs to provide a way to map this name to an input or output event in the DEVS model. The implementation we have created requires the name to have the syntax: EventName_Port. So the input in Example 3 maps to an event named Release and goes to the Fabrication1 port.

The type string specifies the data structure for the variable name. The modeler would match this name and type to the actual data name and type coming from the DEVS model interface.

For this implementation, the type definition is not specified in XML; rather, it is specified as a token separated string. Tokens are separated by the comma character, and attributes of tokens are separated by the colon character. The type definition specifies the data structure of values being exchanged; the actual values are filled in when the models are executed. The names of the fields must match the I/O names from the associated DEVS model.

In Example 3, the type definition for the output variable *Outs_Fabrication1* specifies it is a collection of records. The collection tag defines that multiple instances of this record can be received for the same time interval. The record contains two string fields with names of *product* and *destination;* both are defined as "Key" fields. The key tag specifies the fields to use for determining the distinct instances of values. In this case, if two received values have different product names or destination names, they are different types of messages coming from the simulation. If two messages are received that have the same product and destination name, then it is the same type of message with ordered values. The last message received has the most recent value. For example, if two messages were received for product1 with the destination going to ASY_WH_1, then these are both considered two different values for the same status message. The definition also includes an Integer field with a name of *quantity*.

A similar definition for the LP interface is shown in Example 4. This example shows an input variable named FABRICATION_Actual_Released and output named FABRICATION_Releases. The type definitions show both the inputs and outputs having arrays with dimensions of three rows and four columns. This is a configuration of the interface model for the mapping and transform example described in section 3.3.1.

```
<LPINTERFACE>
      <Data Input name ="FABRICATION_Actual_Released">
                  <Type>Float:ActualOuts[3,4]</Type>
      </DataInput>
      …
      <DataOutput Name="FABRICATION_Releases">
            <Type>Float:FactoryShips[3,4]</Type>
       </DataOutput>
 </LPINTERFACE>
```

Example 4. LP Input and Output KIB Variable Definitions

Next the Interface relationships define the mapping and transforms. Example 5 shows the DEVS to LP mapping for the *actual out* messages from the fabrication plants. This transformation corresponds to equation (2) in section 3.3.1.1. The quantities of all lots leaving the factory need to be summed across the time period to input into the LP model.

The DEVS➔LP mapping first defines the variables to use from the LP and DEVS interfaces using the XML tags <LPNAME> and <DEVSNAME>. This mapping must contain an output variable from the DEVS model and an input to the LP.

The data transformation specification first defines which transform to use and the input and output mappings to and from the transform. In Example 5, the transform is to sum all values. The input to the transform is the quantity data field from the DEVS variable name Outs_Fabrication1 in which the destination fields match the value of 'ASY_WH1' and the product field matches 'Product1'. If multiple records are received with matching product and destination values, the quantities are summed. The output of the transformation is written to the LP data input array variable ActualOuts. The quantity is written to row 3, column 1. The transform also writes the value 1 to row 1, column 1 which corresponds to the product number. The value 1 is also written to row 2, column 1 which corresponds to destination 1. A second transform is configured to map the simulation output to the LL input array values corresponding to product 1 destination 2. Similar mappings would be configured for product 2.

```
<DEVSLPMAP>
      <LPNAME>FABRICATION_Actual_Released</LPNAME>
      <DEVSNAME>Outs_FABRICATION1</DEVSNAME>
      <DATA_TRANSFORMATION> SUM:ALLVALUES,

      quantity:Field:Destination=ASY_WH1:Field:Product=Product1
      ActualOuts[3,1]:ActualOuts[1,1]=1:ActualOuts[2,1]=1

      </DATA_TRANSFORMATION>
</DEVSLPMAP>

…

<DEVSLPMAP>
      <LPNAME>FABRICATION_Actual_Released</LPNAME>
      <DEVSNAME>Outs_FABRICATION1</DEVSNAME>
      <DATA_TRANSFORMATION> SUM:ALLVALUES,

      quantity:Field:Destination=ASY_WH2:Field:Product=Product1
      ActualOuts[3,2]:ActualOuts[1,2]=1:ActualOuts[2,2]=2
            </DATA_TRANSFORMATION>
</DEVSLPMAP>
```

Example 5. DEVS to LP Interface Relationship for Product 1

Example 6 shows the LP➔DEVS transform corresponding to equation (3) in section 3.3.1.1. A single release value is read from the LP solve and then divided into equal buckets. The divisor is determined by the frequency of the DEVS model in relation to the LP model. For the XML specification, this frequency is defined in the KIB control module, which will be described later.

The LP➔DEVS relationship shown in Example 6 first defines the input and output variables. These must be configured in the LP and DEVS interface specifications. Next the data transformation is configured. For this model, there is a disaggregation configured. Elements from the LP FactoryShips array are mapped to DEVS inputs. For product1 with the destination ASY_WH1, the row 3 column 1 entry from the

FactoryShips array is used. The corresponding configuration for ASY_WH2 is shown in the second LPDEVSMAP specification.

```
<LPDEVSMAP>

      <LPNAME>FABRICATION_Releases</LPNAME>
      <DEVSNAME>Release_FABRICATION1</DEVSNAME>
      <DATA_TRANSFORMATION>

       Disaggreagate_equal_values:Round,
       FactoryShips[3,1],
       quantity:Field:Destination=ASY_WH1:Field:Product=Product1

      </DATA_TRANSFORMATION>
</LPDEVSMAP>

…

<LPDEVSMAP>
      <LPNAME>FABRICATION_Releases</LPNAME>
      <DEVSNAME>Release_FABRICATION1</DEVSNAME>
      <DATA_TRANSFORMATION>

      Disaggreagate_equal_values:Round,
      FactoryShips[3,2],
      quantity:Field:Destination=ASY_WH2:Field:Product=Product1

      </DATA_TRANSFORMATION>
</LPDEVSMAP>
```

Example 6. LP to DEVS Interface Relationship for Product1

Example 7 shows a KIB control specification for a DEVS/LP configuration. First, the controlling model is defined. This is the model that provides the data value to synchronize the KIB execution. Second, the KIB module name and KIB variable name is defined. In Example 7, a variable named *sync* with a field named *value* from the module named *synchronization* is used. This variable must be defined in the DEVS interface since the controlling model is DEVS. Next, the execution sequence is defined. The

sequence defines the order of model solves and data transformations. Then the control type is defined. This model defines a periodic control in which 24 DEVS cycles run to each individual LP cycle. This models the hourly simulation run with daily solves.

```
<KIBCONTROL>
      <CONTROLLING_MODEL>DEVS</CONTROLLING_MODEL>
      <MODULENAME>Synchronization</MODULENAME>
      <VARIABLENAME>SYNC:value</VARIABLENAME>
      <EXECUTIONSEQUENCE>
            DEVS,DEVSLP,LPSOLVE,LPDEVS
      </EXECUTIONSEQUENCE>
       <CONTROLTYPE>Periodic:DEVSCYCLES:24</CONTROLTYPE>
 </KIBCONTROL>
```

Example 7. KIB Control Specification

4 MULTI-FORMALISM SUPPLY CHAIN NETWORK MODELING

An approach for creating the conceptual Semiconductor Supply Chain Network models will be described. Then a methodology is shown on how the conceptual model can be decomposed into a suitable planning and manufacturing multi-model for mapping into KIB LP/DEVS multi-formalism models.

The conceptual model describes representations for the supply chain network topology and product routings. The topology includes the actual physical entities that make up the production and logistics facilities within the supply chain network. These would typically be the factories, warehouses, and shipping links within the supply chain network. The product routings define how material flows through the network to become a finished good. It starts as raw material and leaves as the finished product. The different ways that products can flow through the facilities is considered the set of possible product routes.

At different steps through the product routing, the material changes into different intermediate products. Which intermediate product the material becomes can either be by an explicit decision or by the results of product quality from a stochastic manufacturing process. In semiconductor manufacturing, product output can be of varying qualities due to the complex physics involved in making the final product. Yields are generally stochastic and supply planning is done using expected values which are determined through advanced process monitoring and statistical analysis.

In our multi-modeling approach we separate the stochastic processing yields from the explicit decisions. The stochastic processing dynamics will be modeled as the manufacturing processes and the decision points will utilize planning algorithms.

When the conceptual supply chain network models are separated into planning and manufacturing components, a specification is needed to describe how the model components will interact and what data they will share. Specifically, what will the planning frequency be, what types of data will be sent, and what granularity should there be. This is not dissimilar to problems corporate enterprises must address with communications between their manufacturing and planning systems.

4.1 Supply chain network Topology

A prototyped semiconductor supply chain network topology is shown in Figure 21. The supply chain network consists of a raw silicon warehouse (siWh), two fabrication plants (Fab1, Fab2), two wafer inventories (WI1, WI2), two assembly test sites (AssemblyTest1, AssemblyTest2), two semi-finished goods inventories (SFGI1, SFGI2), two finishing lines (Finish1, Finish2), two components warehouses (CW1, CW2), one geographical warehouse (GEO), four customers (Cust1, Cust2, Cust3, Cust4) and shipping links connecting entities to the different geographies. This is a representative subset of what would be seen in an actual semiconductor manufacturing supply chain network.

The siWH provides raw silicon wafers to the fabrication plants (fabs). There is a single warehouse that can ship the wafers to either of the fabs. The fabs put the actual micro-electronic circuitry onto the wafers. Product leaving the fabs can be shipped to

either of the wafer inventories. From the wafer inventory, the product can be released into an assembly test process. In this process, the wafers are cut into die, and the die are put into packages. There can be different types of packages; this topology illustrates package type A or B. After packaging, the material goes through a testing process to determine the electrical characteristics of the product. The characteristics determine which finished products it can be configured to. After leaving the assembly test process, the material can flow into a semi-finished inventory point. The semi finished product can be released into the finish process in which the product is set to its final configuration and packaged. The finished product flows into the component warehouse where the finished goods are stored. From the components warehouse, the products can be shipped to the geographical warehouse or to customers.

The arrows in Figure 21 illustrate shipping links between entities in different geographies. This model shows that shipping is required from the siWH to the fabs, from the fabs to the assembly/test/finish facilities, and from the components warehouses to the Geo warehouse or customers. The geographical regions can be anywhere in the world. This model shows that customer 1 and customer 2 can be supplied from the Geo warehouse. Customer 3 and customer 4 can be supplied from the components warehouse number 2. This model represents a scenario in which the siWh, fabs, assembly/test sites, and customers are in different geographies around the world.

Figure 21 illustrates a product P2 going into the Fab1 and leaving as either product P4 or P5. Similarly for Fab 2, the product goes in as product P3 and can leave as

product P6 or P7. Products P4, P5, P6, and P7 are intermediate products. More details on intermediate products will be given in the next section.

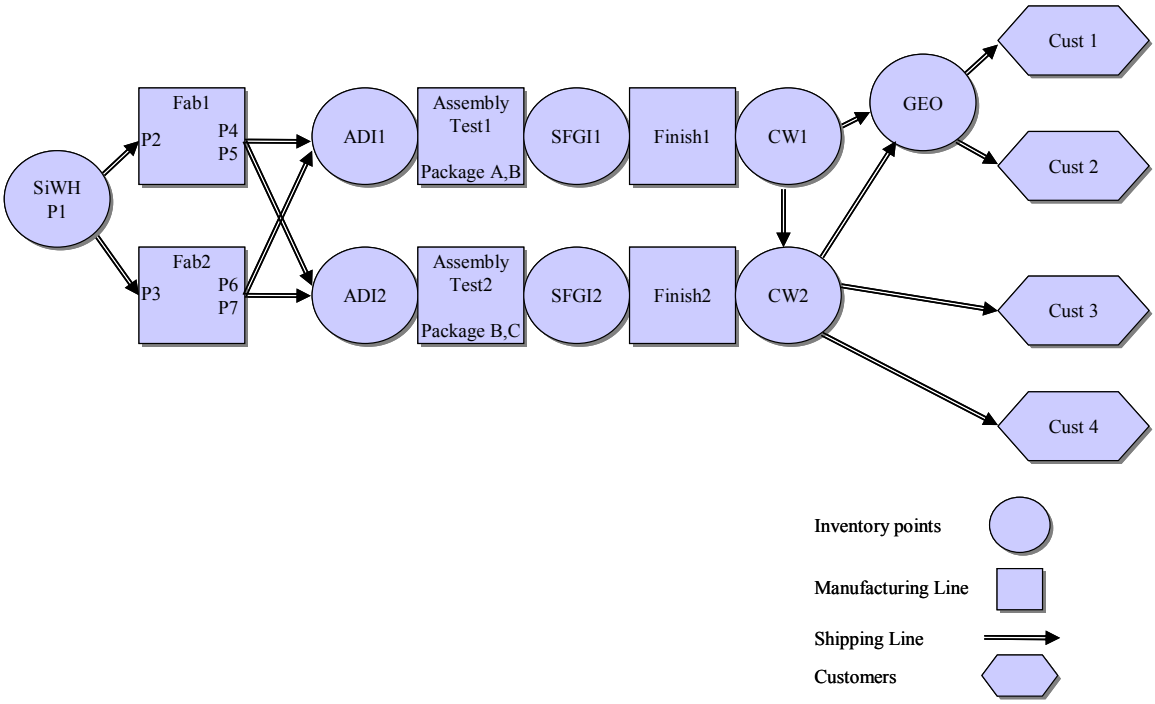


Figure 21. Supply Chain Network Topology

The delay times, capacities, and stochastic distributions for all the entities will be configurable parameters in the simulation. Some important characteristics of a semiconductor supply chain network are:

- Delay times for the fabrication plants can be 6-12 weeks
- Delay times for the assembly/test can be 2-4 weeks

- The fabrication plants and assembly test plants are usually capacity constrained. (i.e. bottlenecks).

Shipping times vary based on distance, method, and delay through customs in different countries. For example, air, sea, and ground shipping all have different delays.

4.2 Product routing

A representative product routing corresponding to the topology in Figure 21 is shown in Figure 22. This product routing has mappings for 30 different products (P1-P30). The product routing can be read starting with unfinished products at the top to finished products at the bottom of Figure 22. Products located at the top are further away from being finished than products closer to the bottom. Raw silicon (P1) is stored at the start of the network and eight finished products (P23-P30) can leave. All other products are intermediate levels created at different manufacturing or assembly steps. P1 is raw silicon that is received from a raw materials supplier. The finished goods P23-P30 are also designated as FG1-FG8. These are the finished products ready to be shipped to the end customer. Products P8-P12 have a package identifier tied to them. This is the type of package the semiconductor die is combined with during the assembly step (e.g. Laptop, Desktop, or Server package).

Horizontal lines have been added to indicate where products change names by a decision or a stochastic physical process. The solid horizontal lines in the product routing are points where the next product type is determined by a control decision from the decision layer. The dashed horizontal lines show where product split is determined by a stochastic distribution in the manufacturing or assembly process.

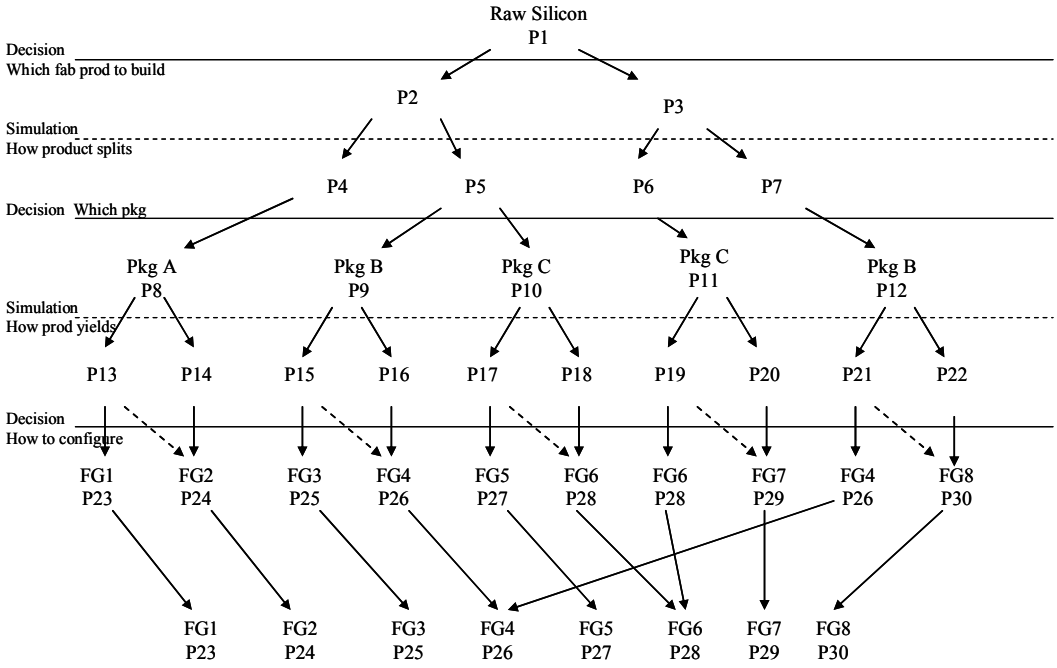


Figure 22. Product Routing

To clarify, the dashed diagonal line, as seen connecting P13 to P24 in Figure 22, indicates that P13 can be configured to different products. That is, P13 can be used to make P23 or P24. Similarly, the dashed lines for P15, P17, P19, and P21 indicate they can be configured to different products. The solid line, like the one connecting P13 to P23, is the default value assignment.

4.3 Mapping of Product Routing onto the Topology

The relation of the product routings to the supply chain network is shown in Figure 23. It can be seen from Figure 21 and Figure 22 that there are dependencies between the product routing and the supply chain network topology configuration.

Products change names as they flow through the supply chain network. Raw Silicon (P1) changes to P2 or P3 as soon as it enters a fab. Products arriving at inventory points do not change names, but can change when they are released to manufacturing, such as P4 becoming P8. The specification of what products can be built in the fabrication plant and which packages are supported in assembly test determines the different routings that the intermediate products can take, as illustrated in Figure 23. For example, Figure 23 illustrates P5 can be put into package B or C. If P5 is shipped to WI2, it can be used to make P9 or P10. If P5 is shipped to WI1, it can only be used for P9 since AssemblyTest1 does not support package C. Similarly, since P7 can only be placed in package B, it can only be used to make P12 regardless of which Assembly/Test site it is sent to.

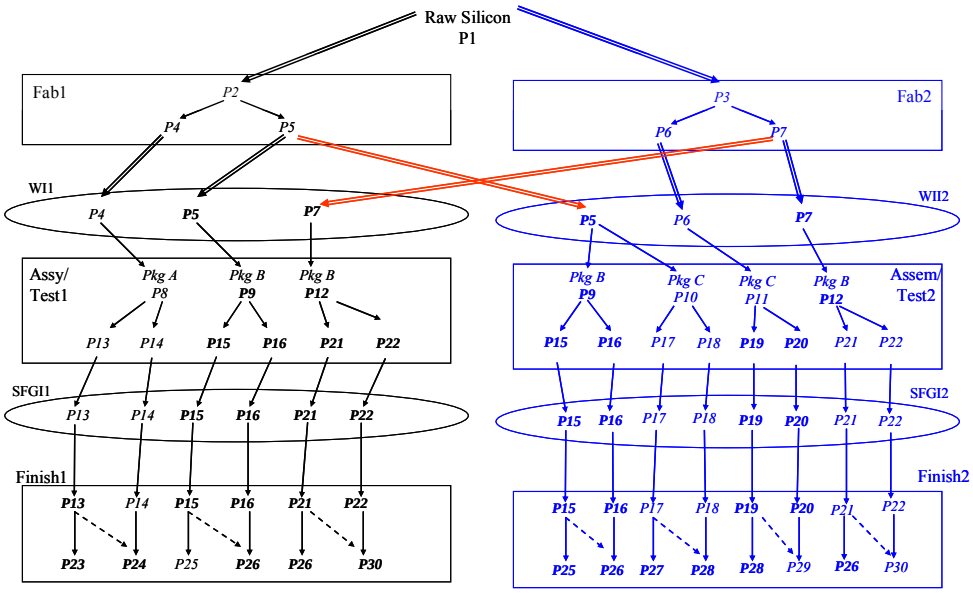


Figure 23. Product Routing Mapped onto the Topology

4.4 Control inputs

A set of decisions were shown by the horizontal solid lines in Figure 22. The mappings of these decisions to control inputs into the topology are shown in Figure 24. Five major categories of decisions are shown.

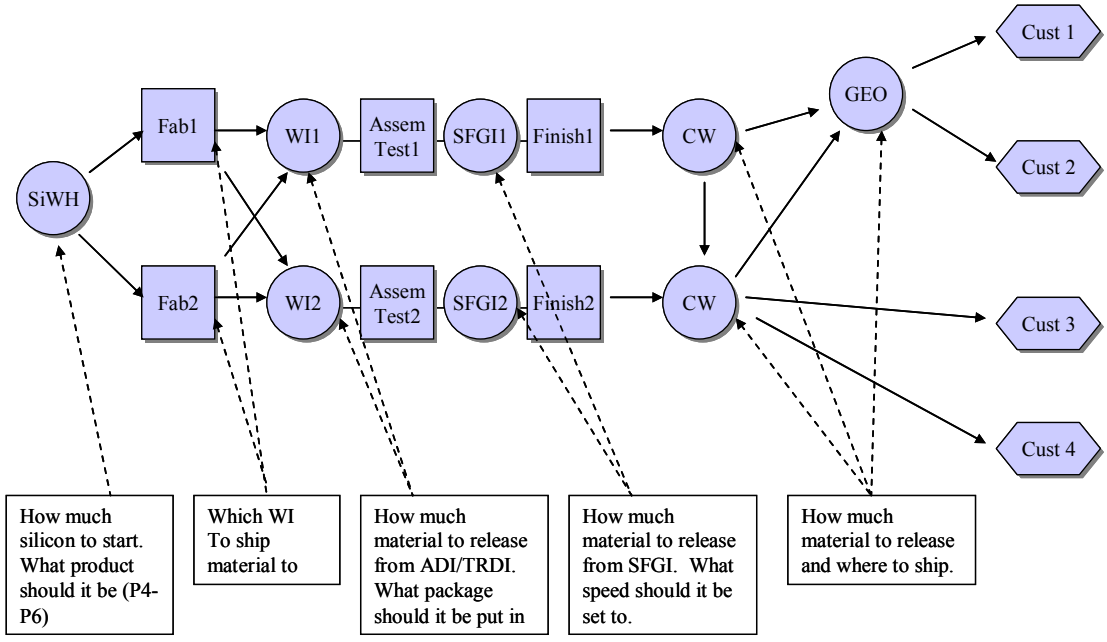


Figure 24. Supply Chain Network Control Inputs

The first set of decisions involves how much material to start and what to build in the fabs. The control input is connected to the silicon warehouse and the command sent to it would specify how much raw silicon to release to each fab and include an instruction of which product should be built.

The next set of decisions specifies where the fab should ship its material. The control will be connected to the end of the fab. This control signal will need some type of translation. The decision algorithm will not have exact knowledge of what is finishing on any given day in the fab due to the stochastic processes. The factory will need to handle cases when the decision algorithm asks more products to be shipped than what is available. For example, assume that Fab1 has built 9,000 units of P4 and 11,000 units of P5. Also, assume that the decision algorithm is expecting 10,000 units of each to be built; it sends a command to ship 5,000 units of P4 to WI1 and 5,000 units of P5 to WI2. How should the physical simulation interpret this command? There could be a set of default rules (e.g. always meet the WI1 request first and send everything else to WI2). Or the command could be converted into a percent split (e.g. Send 50% to WI1 and 50% to WI2). How this is handled will be specific to the scenario and the modeler's responsibility to specify.

In our case studies, this was handled two different ways. First, there is a dependency on how material flows in semiconductor manufacturing process. The manufacturing process modeled the flow of material as discrete batches of units also known as lots. The units in a lot are of a single product type. The quantity of units in a lot is stochastic and based on the manufacturing yields. This is because yield losses happen on a lot-by-lot basis at each processing step. Lots are not divided when they are shipped between entities. Let's assume the scenario as:

1. The average lot size is 500 units

2. The planning algorithm asked to release 500 units to WI1 and 2,000 units to WI2.
3. The factory only has four lots of sizes 300, 600, 500, and 400.

The first method for releasing lots used a round-robin algorithm. The lots are released one at a time to each entity until they run out. In the above scenario, the 300 unit-lot would be released to WI1, the 600-unit lot to WI2, the 500-unit lot to WI1, and the 400-unit lot to WI2. In this case, the results have a significant mismatch from the planning instructions. WI1 received 800 units and WI2 1,000 units. The second method allowed the planning algorithm to send a priority. If the planning algorithm sent priority for WI2, then all four lots would have been sent to WI2 resulting in an inventory of 1,800 units.

The third set of decisions shown in Figure 24 relates to how much material to release from the WI warehouses into the assembly lines and which products to build. The control input to the simulation would need to specify how much of a particular product to release, and what it should be assembled into. The fourth set of decisions shown in Figure 24 specifies how much product to release out of SFGI.

The fifth set of decisions shown in Figure 24 concerns how much material to ship from the CW and Geo warehouses. These decisions are tied to the logistics portion of the supply chain network. All products in these warehouses are finished goods and can be shipped directly to the customer. The command to the simulation would be where to ship and if the product is being shipped to customer, the order ID that is being filled. The

order ID is necessary to track customer service levels (how well are the orders being filled on time).

4.5 Summary

The product routings shown in Figure 22 identify the tree of intermediate products. Each product change is determined by an explicit decision or the result of a stochastic process. The explicit decisions identify the outputs required of the planning algorithms and the stochastic product splits identify where simulation models should be built. The product mappings combined with the topology shown in Figure 23 provide a good conceptual model of the product routing through supply chain network entities. This provides a view of the relations between the possible product routings through the supply chain network facilities.

5 CASE STUDIES

In this chapter we will show a number of case studies to exemplify the utility of the KIB for both theoretical and real world problems. A set of theoretical experiments are run to validate the simulation, LP optimization, and the KIB. After completing the theoretical experiments, the environment was used at Intel Corporation for three different sets of real world supply chain network studies with differing tangents of complexities.

We ran the theoretical set of experiments to show that the KIB works correctly, the dynamics of the simulation reproduces the expected behavior, and the linear programs can be used to generate the starts schedules. The KIB enables interesting experiments of the dynamics between the LP optimization and simulation models. The experiments also enable observation and validation of the correct behavior of the composed models.

For the second category of experiments, the environment is extended to work with planning models developed with a commercial Honeywell MPC controller. There is a significant increase in complexity for the KIB, planning, and manufacturing models. The size must scale to the topologies seen in real-world multi-geography supply chain networks. The KIB provides an environment that enables a timely implementation of MPC control technology into a discrete manufacturing supply chain network. This is an area in which MPC technology has not been previously used in a commercial setting. The KIB enabled the development and validation of the MPC commercial controller using the previously developed supply chain network simulation. It also provided the potential to obtain understanding of the dynamics of the simulated and studied problems.

The MPC controller performed to design when moved from the simulation environment into production.

The KIB and simulation environment were then used to evaluate how a Honeywell MPC controller could be utilized in the latter half of the semiconductor assembly manufacturing process to support build-to-order scenarios. These experiments required modeling the complex product mappings inherent in semiconductor assembly and test operations. The environment had to be extended to support the composition of MPC models running in conjunction with LP optimizations. The MPC controller would not scale to support the number of variables required to enable the thousands of possible product combinations, therefore an LP optimization was introduced to optimize the mapping selection. This set of experiments increased the scaling requirements for product mappings to handle the real world combinations seen by Intel Corporation. The KIB facilitated experimentation in solving these problems by providing a flexible 'model based' integration environment and enabled reuse of validated simulation and MPC models. The KIB helped avoid significant software development efforts by providing an environment that supported model composability across three different modeling formalisms.

The third set of industrial experiments introduced models requiring scalability in modeling both the topology and product mapping. The KIB environments enabled the simulation based development of the largest controller ever created by the Honeywell application suite. The KIB environment was used for early requirements gathering/analysis, controller development, and end user customer validation. The KIB

facilitated the experimental test-bed required for the development and validation of such a complex controller. The fact that the customers trusted the approach enough for user validation and that the project met its timeline targets, the value and utility of using KIB theory towards multi-modeling with disparate modeling formalisms was shown.

5.1 Theoretical Experiments:

A set of experiments that build upon each other has been formulated to show the utility of the KIB. Two base cases were devised to validate the models before different scenarios were executed. The experiments and their parameters are shown in Table 3. The first set of experiments demonstrated the behavior of the composed model with no stochastic behavior. For experiment 1a, the customer produced a sinusoidal demand signal as input. The TPT of manufacturing was 11 days and the yield was 100%. There was one customer, and the simulation and planning solver ran once every time period.

For experiments 1b and 1c, an additional customer has been added. They both use the non-stochastic setup. In experiment 1b, the planning and solver algorithm run every time period. In experiment 1c, the solver only runs once every seven simulation periods. The results of experiment 1c show the impact of planning weekly rather than daily and how the KIB enables this type of experimentation.

Experiment Number	Experiment Description	Demand Input	Finish TPT	Finish Yield	Number of Customers	*Plan / Sim run ratio*
1a	Non-Stochastic Base Case	Sine Mean 750 ± 150	11 days	100%	1	*1:1 Daily plan*
1b	Two Customers	Sine Mean 750 ± 150	11 days	100%	2	*1:1 Daily plan*
1c	Weekly Planning	Sine Mean 750 ± 150	11 days	100%	2	*1:7 Weekly plan*
2a	Stochastic Base Case	Sine Mean 750 ± 150 Uniform (±10%)	Mean Tri (9,10,12) days	Mean Tri (80%, 90%, 95%)	1	*1:1 Daily plan*
2b	Optimistic Data	Sine Mean 750 ± 150 Uniform (±10%)	Min Tri (9,10,12) days	Max Tri (80%, 90%, 95%)	1	*1:1 Daily plan*
2c	*Pessimistic Data*	*Sine Mean 750 ± 150 Uniform (±10%)*	*Max Tri (9,10,12) days*	*Min Tri (80% ,90% ,95%)*	*1*	*1:1 Daily plan*

Table 3. Theoretical Experiment Scenarios

The second group of experiments adds stochastic behavior. The customer generates a demand where each point of the sinusoidal demand may vary by uniform distribution ±10%. The TPT of manufacturing is configured as a triangular distribution with lower and upper limits of 9 and 12, and mode of 10. The yield of manufacturing is configured as a triangular distribution with min and max values of 80% and 95% and

mode of 90%. In experiment 2a, the mean value of TPT and Yield is sent to the decision solver at each time period.

Experiments 2b and 2c demonstrate that the KIB can be used to change what data is sent between the decision and physical models. There is a mismatch in how the decision and simulation models represent yield and TPT data. The decision model requires one aggregated value for the yield and TPT from each product. The simulation generates a different value for every lot that is started into the manufacturing line. There needs to be an aggregation of the value for the decision model. The KIB supports this aggregation and enables flexible experimentation. For our base stochastic experiment (#2a), we have used the mean value. For the optimistic experiment (#2b), we have reported the minimum TPT value and the maximum yield. For the pessimistic model (#2c), we have reported the reverse, which are the maximum TPT and the minimum yield.

5.1.1 Environment

The KIB has been demonstrated on a set of experiments that, first, shows consistent behavior between the composed models and, second, illustrates the capabilities of the KIB. The composed model has 3 distinct components, the DES of the physical process, the KIB for the interactions, and the LP decision control module. The physical DES models a hypothetical supply chain network topology and its material flows. The LP models the decision algorithms required for calculating starts and ships to meet customer demand. The KIB models the transformations required between the physical

and decision models. A view of the major components of each model and how they are connected is shown in Figure 25.

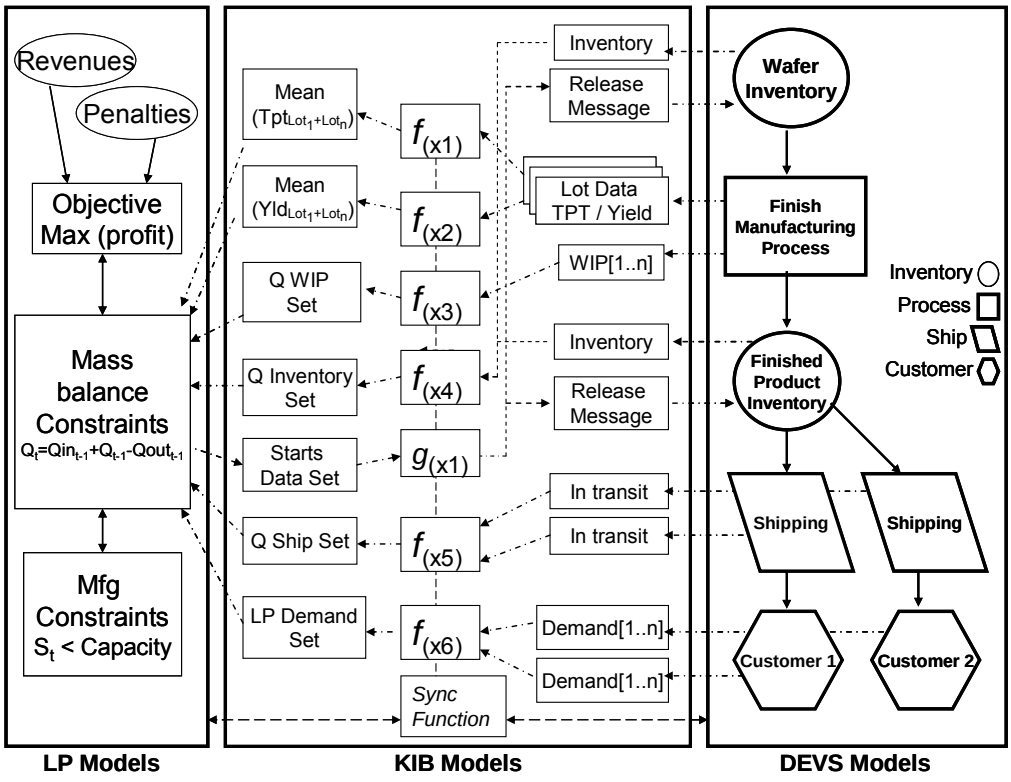


Figure 25. Environment Topology

The decision LP algorithms were developed using the ILOG OPL studio application (ILOG 2005). The DES was modeled and implemented using DEVSJAVA with semiconductor supply chain network extensions (ACIMS 2002). The KIB was implemented using JAVA and enables the modeling of transformations between LP and DES formalisms using XML.

5.1.1.1 KIB Modeling

The Semiconductor Supply Chain Network LP/DEVS formalism modeling language described in Chapter 3 provides the basis of the modeling definitions. The implementation described in Chapter 6 using XML for structuring the models has been used for this set of experiments. The KIB mappings and transforms that needed to be modeled are shown in Figure 25. The lot-based events from the simulation needed to be mapped and transformed into the array-based inputs to the LP. In turn, the LP array-based outputs had to be mapped into the lot-based events to the simulation.

The mappings to and from the models are shown in Table 4. These give details of the structures shown in Figure 25.

5.1.1.1.1 Transformations:

f(x1): TPT: Input is the set of throughput values for each lot that left the manufacturing entity in the last *n* simulation periods. The output is the mean of these values for each product. The last *n* simulation periods correspond to all values since the last time the transformation had been executed (e.g. if the simulation is running hourly and the transform is occurring daily to support daily solves, there will be values for the last 24 simulation runs).

f(x2): Yield: Input is the set of yield values for each lot that left the manufacturing entity in the *n* simulation periods. The output is the mean of these values for each product.

f(x3): WIP: Input is an array of values for each product. Each entry in the array corresponds to the material currently in a segment of the manufacturing process. Output is mapped to a set of vector inputs. Each vector corresponds to each product.

f(x4): Inventory: Input is set of lot objects. The lot objects have fields for product type and quantity. The output is mapped to a vector, each entry corresponding to a different product.

f(x5): Shipping: Input is an array of values for each product. Each entry in the array corresponds to the material currently in a segment of shipping (e.g. on the plane, in customs, etc.) Output is mapped to a set of vector inputs. Each vector corresponds to each product.

f(x6): Demand: Input is a set of vectors; each vector contains the current and future orders for a given product at a given customer. Output is a multi-dimensional array.

g(x1): Releases: Input is a multi-dimensional matrix corresponding to which product to release, how much to release, and the product's destination. The output is a set of command objects to the simulation.

LP	KIB			DEVS
	LP Structure	Mapping	DEVSJAVA Structure	
Inventory Set	Set of Vectors	DEVS➔LP Field to Set	Lot Records	Wafer Inventory
		DEVS➔LP Field to Set	List of Lot Objects	Finished Product Inventory
TPT Data	Single Vector	DEVS➔LP	List of Status Objects	Manufacturing TPT Data
Yield Data	Single Vector	DEVS➔LP	List of Status Objects	Manufacturing Yield Data
WIP	Set of Vectors	DEVS➔LP	List of Arrays	Manufacturing WIP
Shipping	Set of Vectors	DEVS➔LP	List of Arrays	In transit Data
Demand	Set of Vectors	DEVS➔LP	List of Arrays	Customer 1 Demand
		DEVS➔LP	List of Arrays	Customer 2 Demand
Starts	Multi-dimension Array	LP➔DEVS	List of Command Objects	Finished Inventory Releases

Table 4. Table of KIB Mappings

5.1.1.2 DES Modeling

The simulation models partition the supply chain network into factories, inventories, shipping, and customers. This follows the approach researched in (Godding, Sarjoughian et al. 2003); factories model capacity, yield, and cycle time and can change product names. When material arrives at a factory, it will either leave at some later time determined by the TPT configuration or it will be lost through defective yields. The TPT and yields can be configured as a constant number or it can be assigned via stochastic

distribution. Capacity is modeled as an input constraint. The simulation will never start more than what the manufacturing line has capacity to process. In these experiments, capacity has been set sufficiently high to not impact behavior. Factories can report their work in progress (WIP), what was actually output (AO), and what the TPT and yields were for the product actually built. These messages specify the values for each product. WIP can contain multiple values corresponding to different sections of the factory. For example, if WIP is configured to report material in two time buckets, the first bucket reports what is in the first half of the factory and the second reports what is in the other half.

Inventories are holding points for material. Material that arrives at an inventory will not leave until commanded by an external release message. Inventories have a one day TPT for arriving material. That is, anything that arrives on a given day will not be available for release until the following day. Material released from inventory is immediately started in the next connected entity. Release commands sent to inventory points must specify the product, quantity, and destination of the material. The release can optionally specify what the target product should be. Inventories can report the beginning on hand (BOH) inventory level and the AO for each product.

Shipping is used to model the time delays and yield loss of material. Shipping is similar to factories; however, it does not change product attributes. Shipping can report what is in transit (similar to the manner in which WIP is reported), and the AO for each product.

Customers generate orders and consume the finished product. When a customer receives product, they subtract it from their outstanding orders, filling the oldest first. Customers report what their current backlogs are and their projected orders over a predefined planning horizon. The customers have a window in which they cannot cancel orders without penalty. Outside this window they are free to change their orders. The simulations can apply a random distribution to orders outside this cancellation period.

The material flow through the simulation is lot based, as are the TPT time and yields assigned. A lot can contain one or more discrete units of material. Their sizes are determined by a configuration parameter that controls the maximum quantity allowed. For example, if an inventory is configured to have a maximum lot size of 10 and a release command is received specifying that 1,500 be sent out, then 150 lots of size 10 would be output.. If 1501 units are started, then 151 lots would be output with the last containing only one unit. If yield loss is configured to be 80%, and lot size is 10, then two units would be subtracted from each.

The TPT times for each lot are assigned when entering manufacturing. At each cycle, the process time of the lot is incremented. If the process time is equal to the TPT time, the lot is scheduled to leave at the beginning of the next time period. The lot size has an impact on the behavior of TPT. If maximum lot size is equal to 1,000 and a quantity of 1,000 is started, there is only one TPT assigned to the material being processed. However, if the size is set to 100, the random TPT will be assigned to 10 different lots, which will in effect distribute the amount of material that leaves early versus late across the 10 lots.

The time when yield loss is taken is configurable. It can occur during any step of the manufacturing process. The yield loss time impacts the data reported to the decision algorithm. If yield loss is taken in the middle of the process, the decision model will see reduced WIP in the latter half. If it is taken when the material is output, the decision model will see reduced quantity of material in the AO message. In our experiments, the yield loss occurs at the end of the process right before the material is output.

5.1.1.2.1 Simulation Messages

There simulation has a single input command, which is a release command input to the inventories. The format of the release is:

- *$Release_I$ = Quantity of material to release out of inventory at beginning of day. This material will arrive in the next entity downstream on the same day.*

All status messages are reported at the end of the day. A summary of status messages reported by the simulation available for use in the decision layer is given below:

Factories

- $WIP_F = Starts_{BeginningOfDay} + WIP_{BeginningOfDay} - Outs_{EndOfDay}$
- $AO_F = Outs_{EndOfDay}$
- $TPT_F = ActualTpT[1..NumberLots]_{Outs}$
- $Yield_F = ActualYield[1..NumberLots]_{Outs}$
- Inventories

- $BOH_I = Arrived_{BeginningOfDay-1} + BOH_{BeginningOfDay} - Outs_{BeginningOfDay}$
- $AO_I = Outs_{BeginningOfDay}$

Shipping

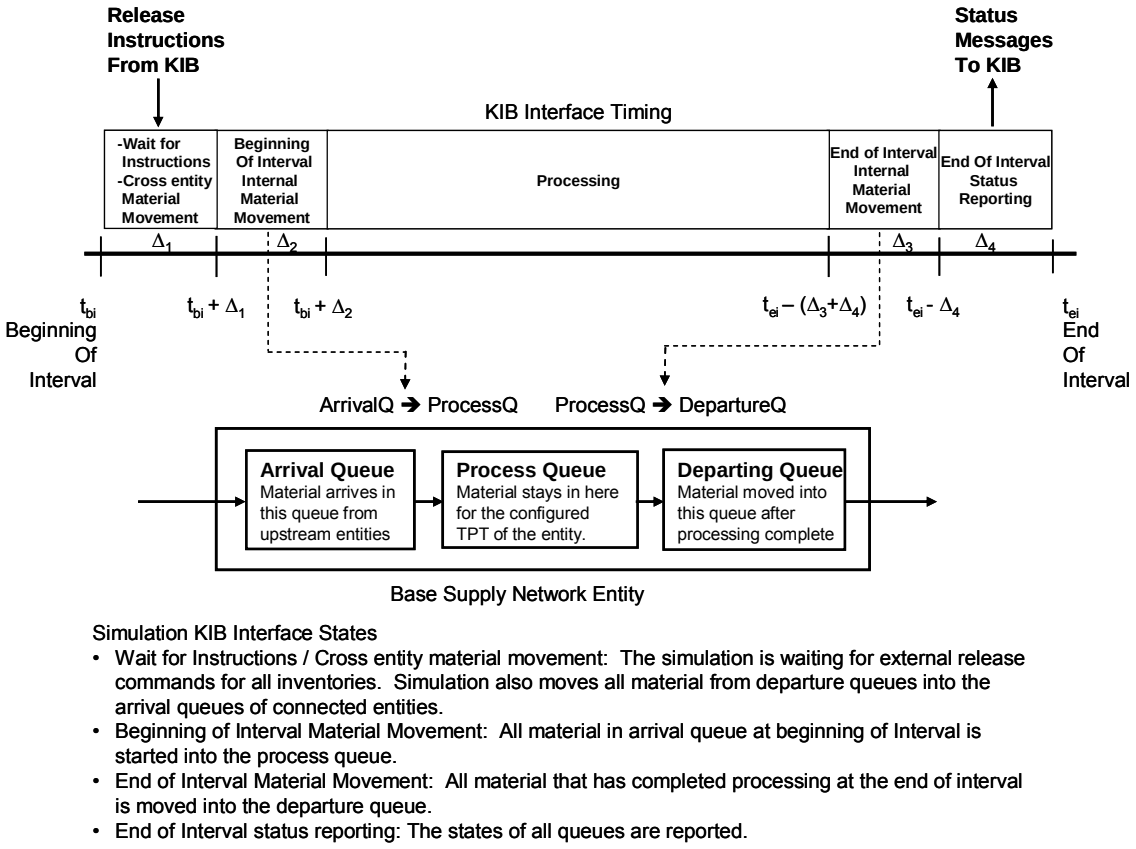
- $Intrans_S = Starts_{BeginningOfDay} + Intrans_{BeginningOfDay} - Outs_{EndOfDay}$
- $Outs_S = Outs_{EndOfDay}$

Customers

- $Backlog = UnfilledOrders_{EndOfDay}$
- $ForecastDemand = OrdersDue[t_{EndOfDay+1} .. t_{PlanningHorizon}]$

The actual outs (AO_I) message for inventories is slightly different than factories and shipping. All three entities are reporting what actually left for the day; however, for inventories it is the quantity that left at the beginning of the day instead of at the end. It corresponds to the release quantities calculated for the day from the decision model (LP). The release commands are sent to inventories at the beginning of day, which is then immediately output to the connected entity. Factories and shipping report what was actually built or shipped at the end of day.

A timing diagram of when input and output events are sent and received from the KIB are shown in Figure 26. The five important simulation interface timing states are shown in the top half of Figure 26. The bottom half shows the queuing design of the simulation base supply chain network entities. The dotted arrows show the timing relations of the interface states and material moving between the internal queues.



Simulation KIB Interface States

- Wait for Instructions / Cross entity material movement: The simulation is waiting for external release commands for all inventories. Simulation also moves all material from departure queues into the arrival queues of connected entities.
- Beginning of Interval Material Movement: All material in arrival queue at beginning of Interval is started into the process queue.
- End of Interval Material Movement: All material that has completed processing at the end of interval is moved into the departure queue.
- End of Interval status reporting: The states of all queues are reported.

Figure 26. KIB Input and Output DEVSJAVA Timing Diagram

The simulation based supply chain network entities have an internal event at the end of each interface state. After the 'End of Interval Status Reporting' event, the entity transitions back the 'Wait for Instructions / Cross entity Material Movement state'.

The first state: 'Wait for Instructions / Cross entity material movement' starts at the beginning of Interval. First, the simulation starts all material that arrived from the previous interval. Second, the simulation waits to receive all external events from the KIB for inventory release commands. Third, the material quantities specified in release

commands is moved from inventories to the connected entities. The duration of this state should be a very small delta and is defined as Δ_1. The next simulation state is 'Beginning of Day Internal Material Movement'. In this state, all material is moved from the arrival queues to the processing queue. Then the model transitions to the processing state. Material stays in this state for the duration of the entities TPT time. The TPT time can be calculated from a random distribution. The yield loss is taken during the processing time. At the end of processing, material is moved into the departure queue at the 'End of Interval Internal Material Movement' state. This state duration should also be very small and is defined as Δ_3. At the end of the day, the status of the entity is reported to the KIB. The status includes the state of all material in the three queues.

5.1.1.3 LP Decision Algorithm

The LP model has an objective function to maximize profit. Revenue is generated when orders are filled. Costs are assigned to manufacturing, inventory holding, and shipping. In addition, penalty costs are assigned for meeting orders late, shipping too much to the customer, and for daily changes in factory starts commonly referred to as thrashing. The objective function is:

Maximize(profit)

Where

Profit = revenue –costs

Revenue = orders filled

Costs = Material Cost + Inventory holding Cost + Factory Thrash penalty

The LP optimizes for a set of solutions over a predefined planning time horizon. The time horizon is determined from the demand input vector received which was generated by the customer entity in the simulation and then transformed and provided as input to the LP via the KIB. The demand vector contains entries corresponding to number orders that need to be shipped starting from the current time to some horizon in the future. For example, assume the vector for productA from customer1 is received:

Demand Vector = <100,200,100,500>

Also assume that the KIB has been modeled to send the LP daily values for demand. The demand vector would specify that 100 units are due today, 200 on the next day, and so on. The time horizon the LP would solve over would be 4, the next four days.

A set of mass balance constraints have been modeled in the LP to enforce physical constraints when the solver populates the solution set. The mass balance constraints contain a set of values for each time period out to the end of the planning horizon. If the time horizon was 4, there would be four sets of values in the solution set. Examples of mass balance equations for finish manufacturing are shown in Equation 1.

Assumptions:

The time horizon is k time periods
There are n different products

Variable Definition

- *FinishStarts[1..k,1..n]*
- *FinishWIP[1..k,1..n]*
- *FinishOuts[1..k,1..n]*
- *FinishTPT[1..n]*

- *FinishYield[1..n]*

(1) $FinishWIP_{p(t=1)}$ = f(x3) =
= KIB transformation of DEVs simulation WIP data (see Figure 25)

(2) $FinishWIP_{pt}$ = $FinishStarts_{p(t-1)}$ + $FinishWIP_{p(t-1)}$ – $FinishOuts_{p(t-1)}$:
$1 < t \leq k$, $1 \leq p \leq n$

(3) $FinishTPT_p$ = f(x1) = KIB transformation DEVS TPT data :
$1 \leq p \leq n$

(4) $FinishYield_p$ = f(x2) = KIB transformation DEVS yield data :
$1 \leq p \leq n$

*(5) $FinishOuts_{pt}$ = $FinishWIP_{p(t=1)}$ / $FinishTPT_p$ * $FinishYield_p$:*
$1 \leq t \leq FinishTPT_p$, $1 \leq p \leq n$

*(6) $FinishOuts_{pt}$ = $FinishStarts_{pt\text{-}FinishTPTpt}$ * $FinishYield_p$:*
$FinishTPT_p < t \leq k$: $1 \leq p \leq n$

Equation 1. LP Finish Manufacturing Mass Balance Constraints

The equations for FinishWIP, FinishTPT, FinishYield, and FinishOuts are shown. FinishWIP has to be modeled as two equations. The first equation (1) populates the initial state for WIP. The value is supplied by the KIB transform function f(x3) shown in Figure 25. The second equation (2) defines how the remaining values for future time periods can be populated.

The FinishTPT and FinishYield vectors will be populated by the KIB transform functions f(x1) and f(x2) for each product from the simulation. These values are constants in the LP (i.e. there are no unknowns for TPT and Yield).

For FinishOuts there are also two equations. The first equation (5) defines what the FinishOuts values will be in time periods between 1 and the TPT value for each product, which was populated in (4). This equation defines that an equal amount of material will leave FinishManufacturing for the first time periods within the TPT. For example, assume the FinishTPT value for productA is 3. Also assume that the WIP value for t = 1 supplied by the KIB = 1200. The FinishOuts value for time periods 1 through 3 would be 400 * FinishYield. For time periods beyond the TPT, equation (6) is used. This equation defines TPT to be equal to what the starts were at the time period minus the TPT time. Following through with the previous example, the FinishOuts for time period 4 would be equal to the FinishStarts at time period 1.

The unknowns in the constraints are the FinishWIP values for $1 < t \leq k$, FinishOuts for $1 < t \leq k$, and the FinishStarts for all t. The LP will populate the unknowns with values for the optimal solution when a solve is initiated. The LP projects a starts schedule for finish manufacturing and a ship schedule to the customers. This projection is what is sent to the simulation as release commands through the KIB transform function g(x1) in Figure 25.

5.1.2 Non-Stochastic Base Model Results

The first set of experimental data was collected against a non-stochastic base model. The demand input, data messages, and the topology for the LP, KIB, and simulation models are shown in Figure 27. The topology for the simulation model includes a die inventory, a finish manufacturing line, a finished inventory, a shipping

link, and one customer. The die inventory has an infinite supply providing the input material for the simulation. The throughput time is 11 days for finish manufacturing, one day for Finish Warehouse, and two days for shipping. There is no yield loss and the TPT values are constant. The demand input is a sinusoidal with a mean of 750 and varies +/- 150 in each cycle. The output results are shown in Figure 28 and Figure 29.

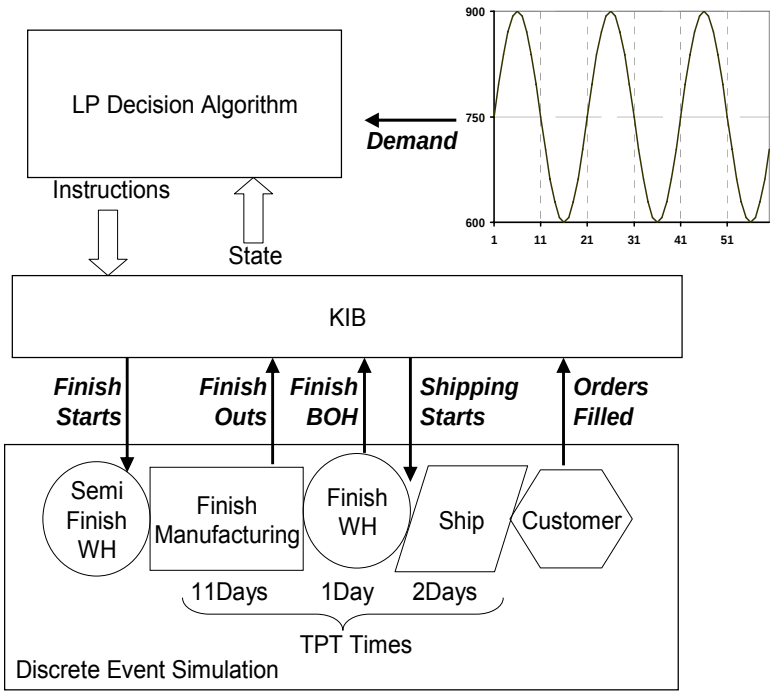


Figure 27. Base Model

In Figure 28, the Finish starts data are compared to FinishOuts. The measured TPT time is 11 days as expected. This starts pattern is the results of the LP algorithm discussed earlier. The conflicting goal of minimizing inventory and minimizing factory thrash has caused the oscillating pattern.

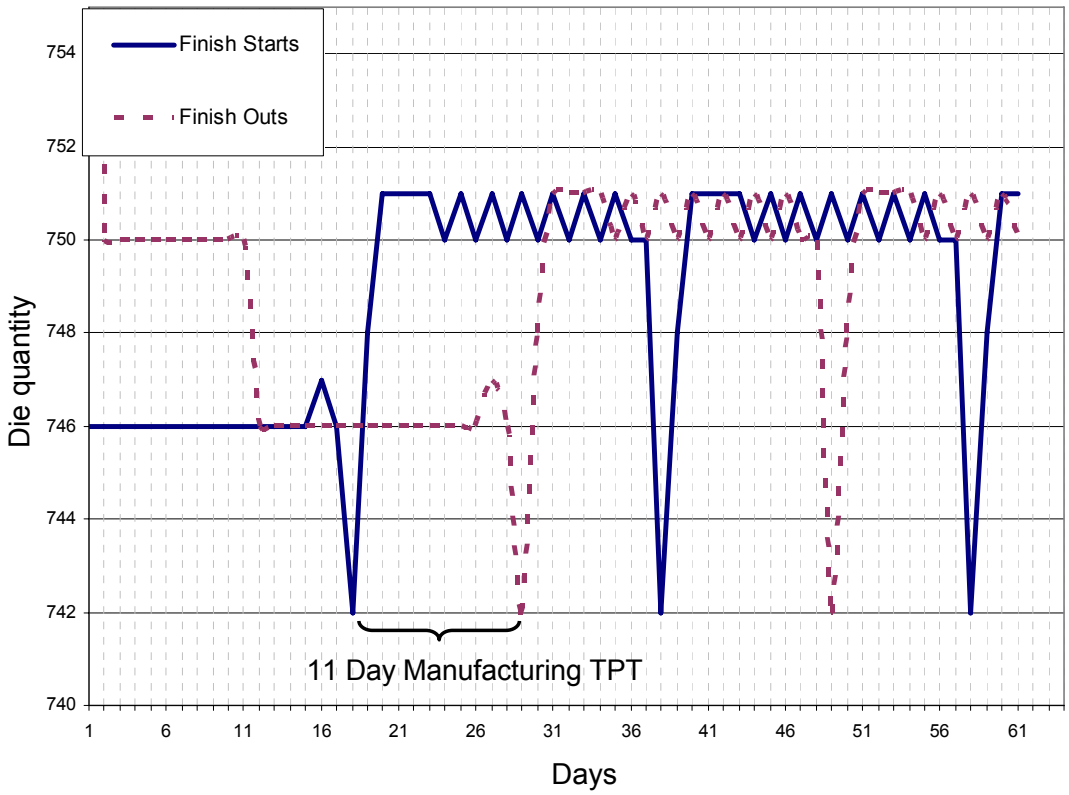


Figure 28. Base Model Finish Starts versus Finish Outs

The results in Figure 29 show that the LP is successfully commanding the simulation to build material ahead of time in the Finish Warehouse while meeting all demand on time. It can also be seen that shipping is working correctly. Orders are reported filled three days after material is started in shipping. Shipping has a two day delay, so material started at beginning of day one, will arrive at customer at beginning of day 3.

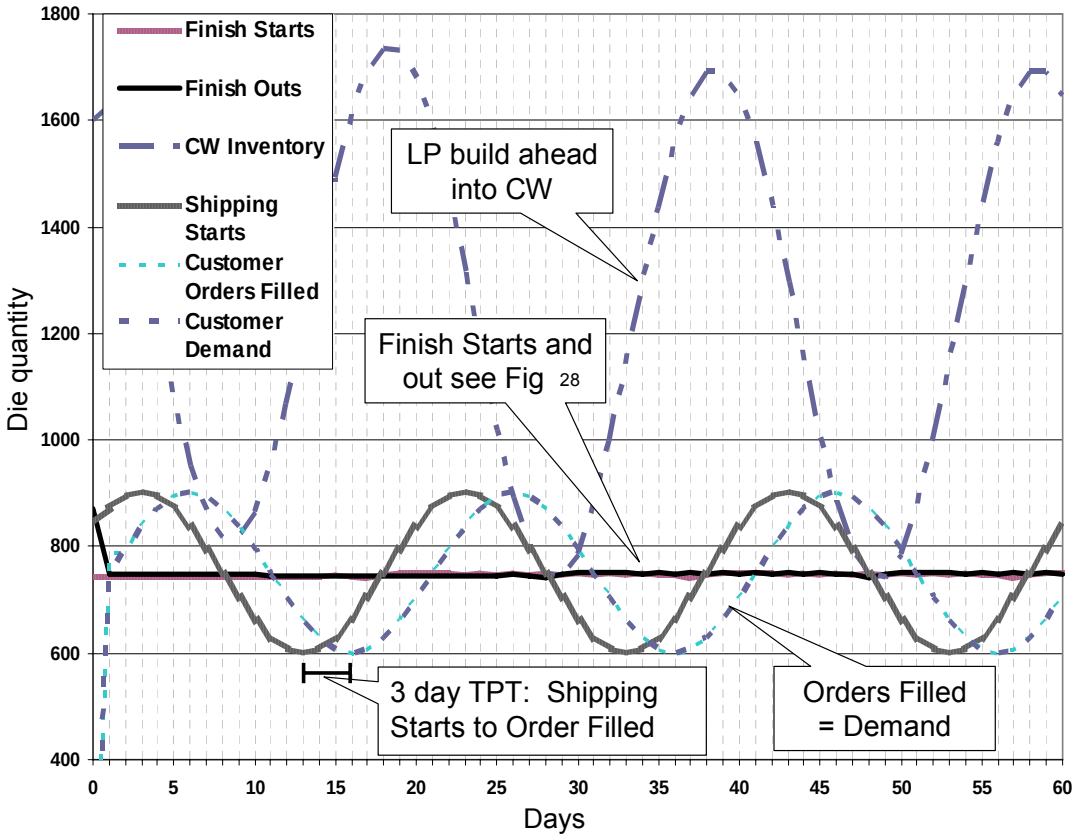


Figure 29. Base Model Non-Stochastic Results

5.1.3 Two Customers Daily Plan

The setup for two customer demand is shown in Figure 30. One additional customer has been added from which product can be shipped out of the Finished Warehouse. The demand input to the LP for both customers is the same. A new demand message transform configuration needed to be added to the KIB integration model for

supporting the additional customer. The LP and simulation models were updated accordingly. No software changes were required to support the new interface.

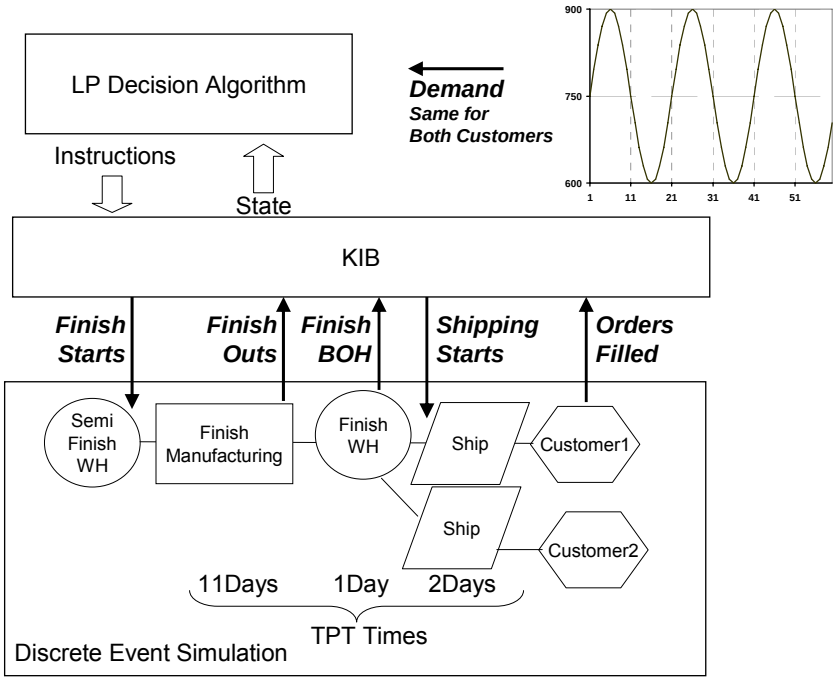


Figure 30. Two Customer Experiment Setup

The output of these experiments is shown in Figure 31. The results are similar to those described for the base case. The major difference is twice the material is being started and building up in the finished inventory. This is expected since demand must be met for both customers. The KIB enabled the configuration of release commands to include multiple destinations to the simulation and for the simulation to provide data for multiple shipping and customer entities.

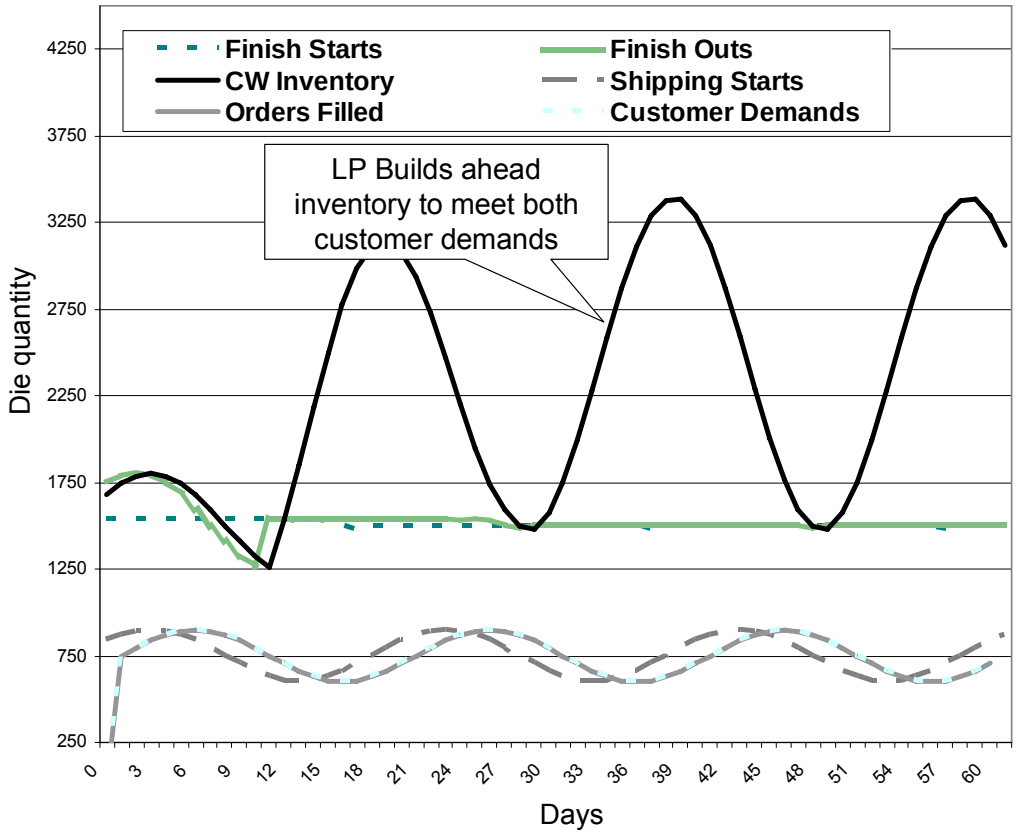


Figure 31. Two Customers, No Stochastic, Daily Plan Results

5.1.4 Two Customers Weekly Plan

The data results for running a weekly plan is shown in Figure 32. It can be seen in these plots that customer 1 is regularly missing orders. This was not expected since the LP looks 32 days ahead with a perfect forecast. Further analysis determined the cause of why orders are being missed. Recall the mass balance constraints the LP used for the FinishManufacturing (See section 5.1.1.3). The LP divides the total quantity of WIP in manufacturing by the TPT time for the projection of what is leaving in the first 11 days

(TPT time). If more material resides in the back half of the factory (i.e. leaving earlier), the LP will incorrectly calculate less material is leaving. In this simulation run, the LP did not account that there was enough arriving at the FinishWarehouse to meet demand four days out when the FinishManufacturing line was not linearly loaded. To correct this problem, the LP would either need to use a more detailed model of the FinishManufacturing line or the LP needs to keep an inventory buffer to compensate for error. It would be straight forward from interface perspective to experiment with different levels of detail in the data models sent to the LP. The KIB allows the number of WIP time buckets sent to LP to be easily configured. The impact on LP performance with the extra data constraints would need to be considered with such analysis.

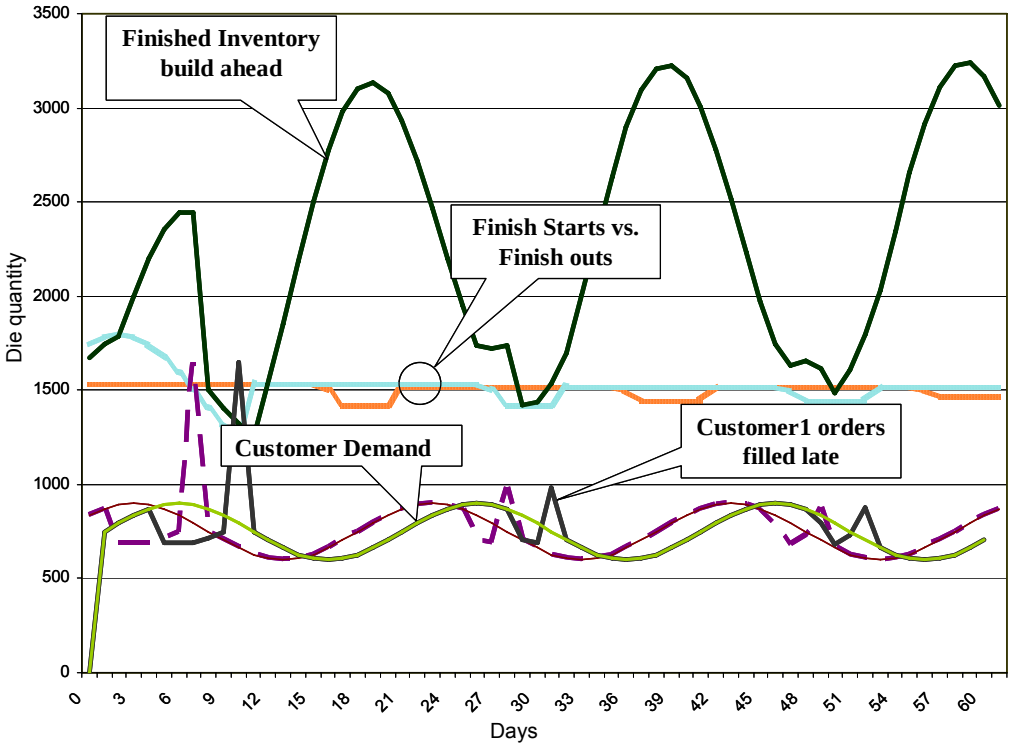


Figure 32. Two Customers with Weekly Plan Results

A conclusion that can be drawn from this experiment is that there is a dependency existing between the LP input data model and the frequency of the plan. If the plan is run at a lower frequency, it needs a more accurate model of what will happen during that planning cycle to produce good output. If an accurate data model cannot be obtained, then the error will need be buffered, which could be accomplished by holding inventory.

5.1.5 Stochastic base model results

The base stochastic model is illustrated in Figure 33. For this set of experiments the ideal sinusoidal demand signal has been replaced by noisy demand. In addition, triangular distributions have been used to generate the random values for yield and TPT. The distribution parameters for the yield are: 80% for lower limit, 90% for the mode, and 95% for the upper limit. The parameters for TPT are: 9 days for lower limit, 10 days as the mode, and 12 days for the upper limit.

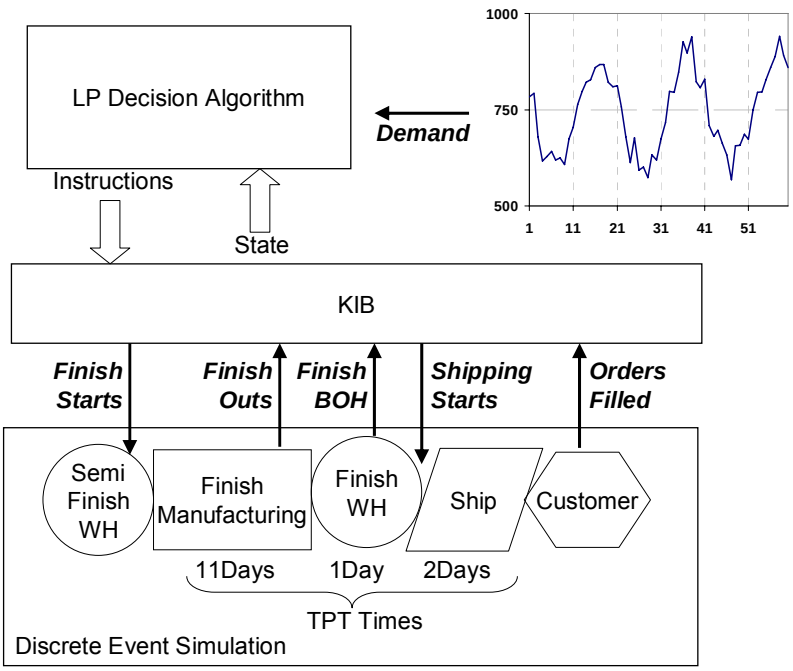


Figure 33. Base Stochastic Model

An analysis of the simulated yield results is shown in Figure 34. The simulation was run five times, and for each run, the total units in and out of FinishManufacturing were collected and graphed. The expected yield of the simulation would be the mean of the triangular distribution which is 1/3*(80+90+95) = 88.33%. The maximum deviation is on the second run in which the actual simulated yield is 0.89% lower than the expected yield of 88.33%. Closer observation of run 2 also shows that the LP decision algorithm was making up for the extra yield loss by starting more material.

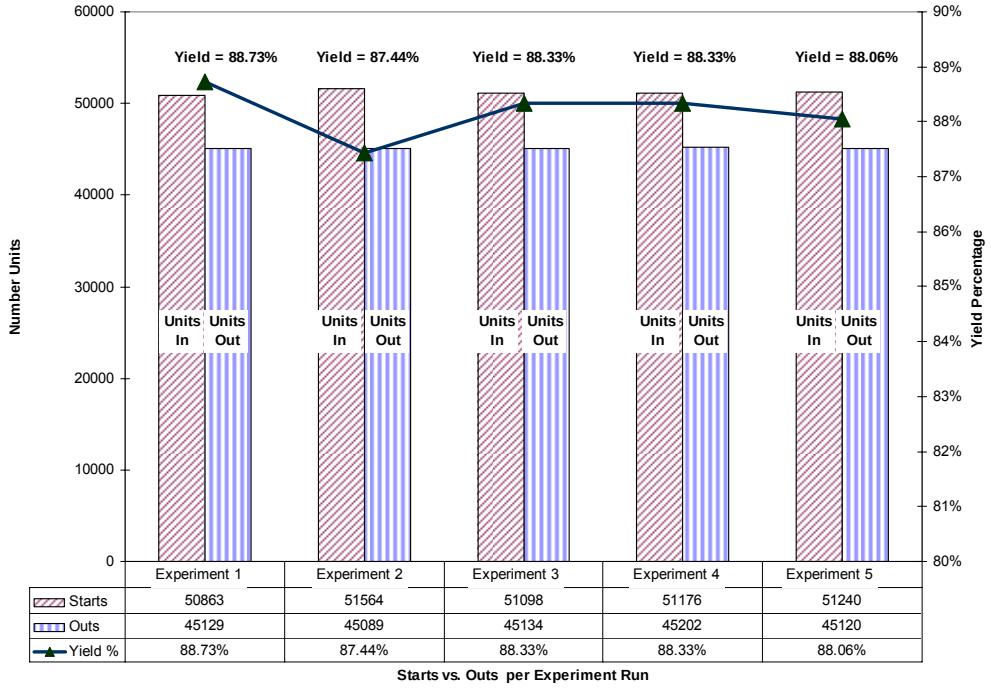


Figure 34. Stochastic Yield Analysis over 5 Experiment Runs

An analysis of the simulated TPT results is shown in Figure 35. This is based on the same five experiments used for the yield analysis. The TPT was configured as a triangular distribution with parameters: 9, 10, and 12. TPT times in the simulation are rounded up to the next discrete integer values. For the simulations, there would be three different TPT values: 10, 11, and 12. Their expected frequency would be 33.33% for TPT of 10, 50% for TPT of 11, and 16.67% for TPT of 12 (See Figure 36).

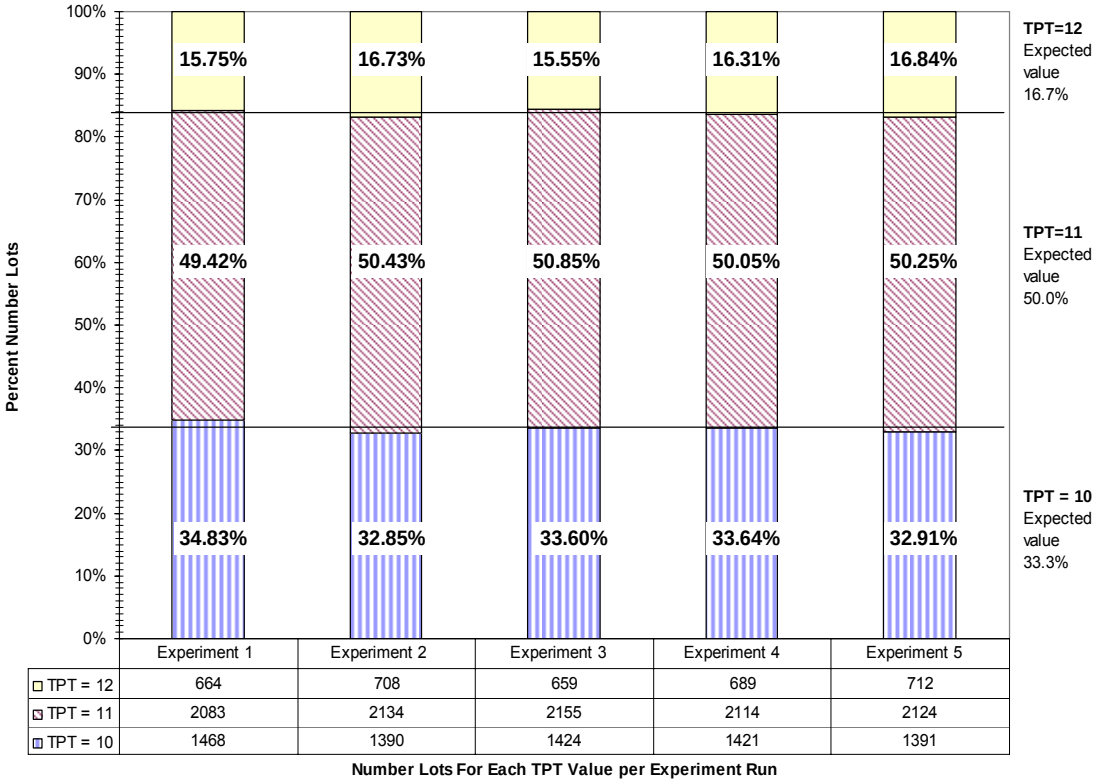


Figure 35. Stochastic TPT Analysis over 5 Experiment Runs

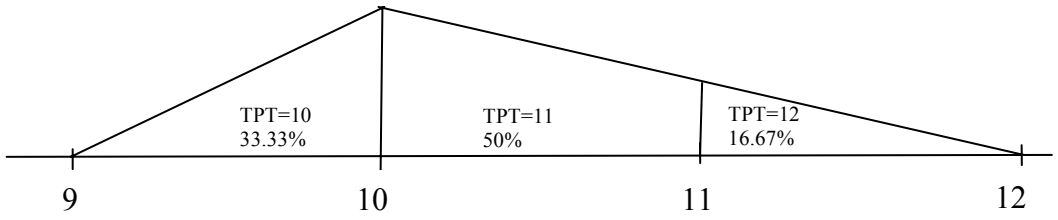


Figure 36. Expected TPT Distribution

The segments of the bars are the percentage of lots that had TPT of 10, 11, and 12 respectively. It can be seen that the results shown in Figure 35 were consistently close to the expected values of 33.33%, 50%, and 16.67%.

Additional analysis has been performed on a single point for one of the experiments. A plot of FinishStarts versus FinishOuts can be seen in Figure 37. There is a peak for finish starts at t = 21. The corresponding peak in outs is at t = 32.

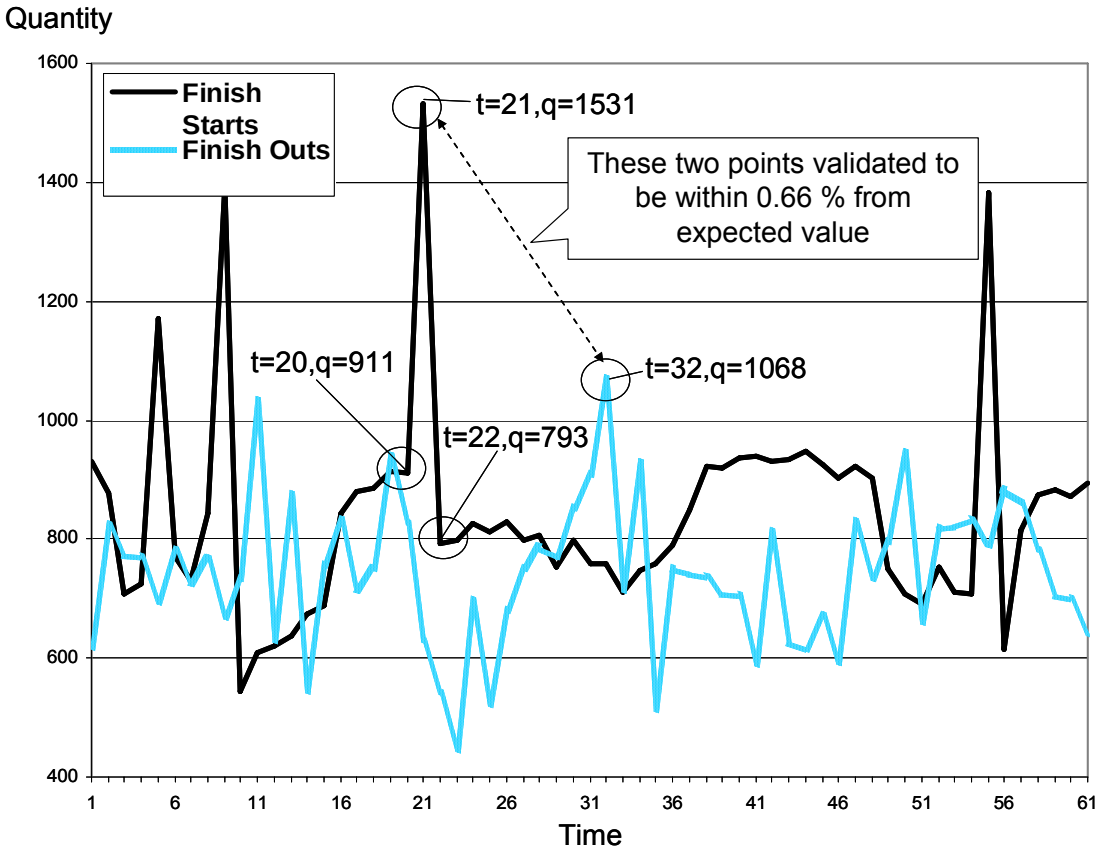


Figure 37. Stochastic Base Case Finish Starts versus Outs

An analysis of what the expected value of outs would be at t = 32 is shown in Table 5. The difference of the actual value from the calculated value is off by seven units or 0.66%.

Start Time	Quantity Started	Total Expected Quantity Out (88.33% yield)	Expected Percent out at t=32	Expected Quantity Out at t=32
T=20	911	804	33.33%	268
T=21	1531	1352	50.00%	676
T=22	793	700	16.67%	117
			Total	1061

Table 5. Expected Finish Outs at t=32 for Stochastic Base Case

In further analysis every point was compared to the expected value. The average error across for each discrete point was ±6-8% in 10 simulation runs. The daily errors can be attributed to the discreet flow of lots and TPT times. The overall results were acceptable as shown in Figure 34 and Figure 35. The daily errors canceled each other out as positive and negative values.

In Figure 38, it can be seen that the LP is pre-staging material FinishInventory as expected. It was also able to successfully meet all orders on time. From these results, we can conclude the simulation, LP, and KIB performed as expected.

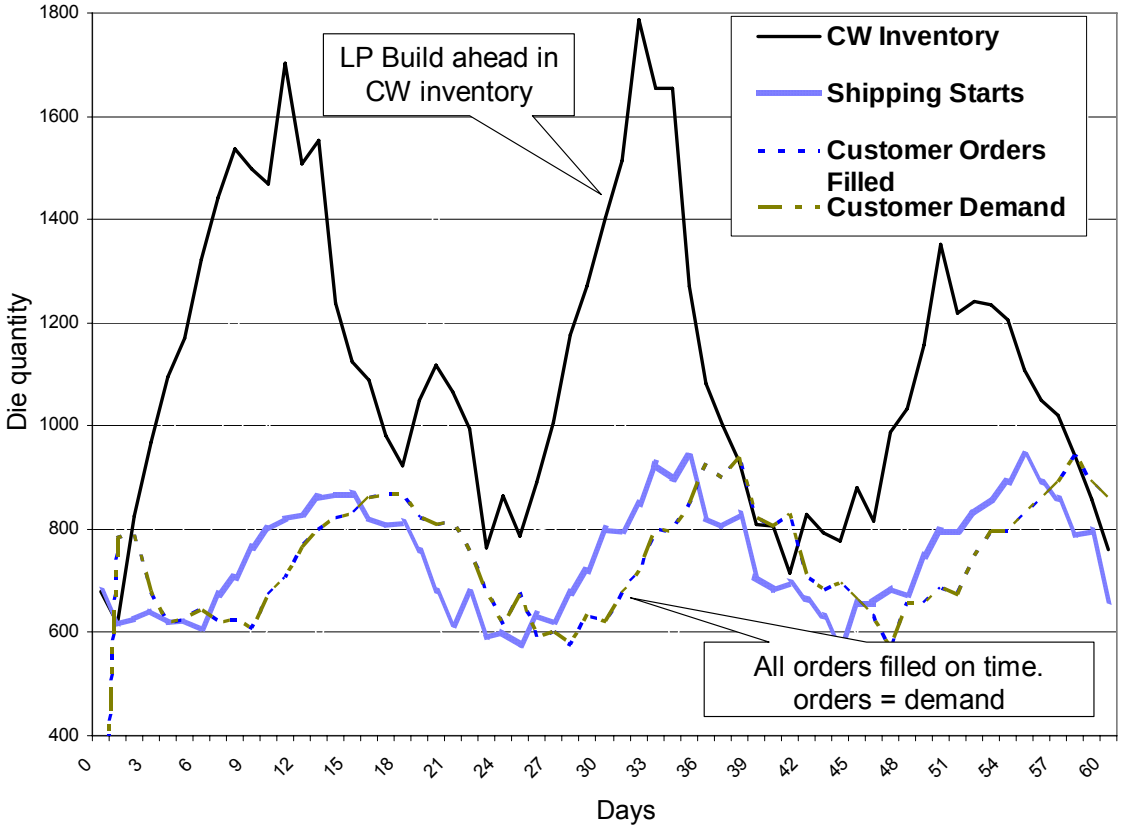


Figure 38. Base Stochastic Order Fulfillment Performance

5.1.6 Optimistic Model Results

For the optimistic experiment, we changed the DEVS➔LP data transformation configuration in the KIB to pass the best values from the simulation for TPT and Yield. This required a simple configuration change in the KIB model. The transforms were changed from MEAN to MIN for TPT and from MEAN to MAX for yield.

This experiment used the same initial data set and input demand as the base stochastic model. The results of the experiment are shown in Figure 39. It can be seen

that orders were being missed consistently after time period 26. Up to this period, the initial inventory was being depleted from FinishWarehouse as can be seen in Figure 40. These are the results expected; since the LP is receiving data that the process has lower yield loss and shorter TPT's than the actual average values. A shortage of material being built in this scenario is expected.

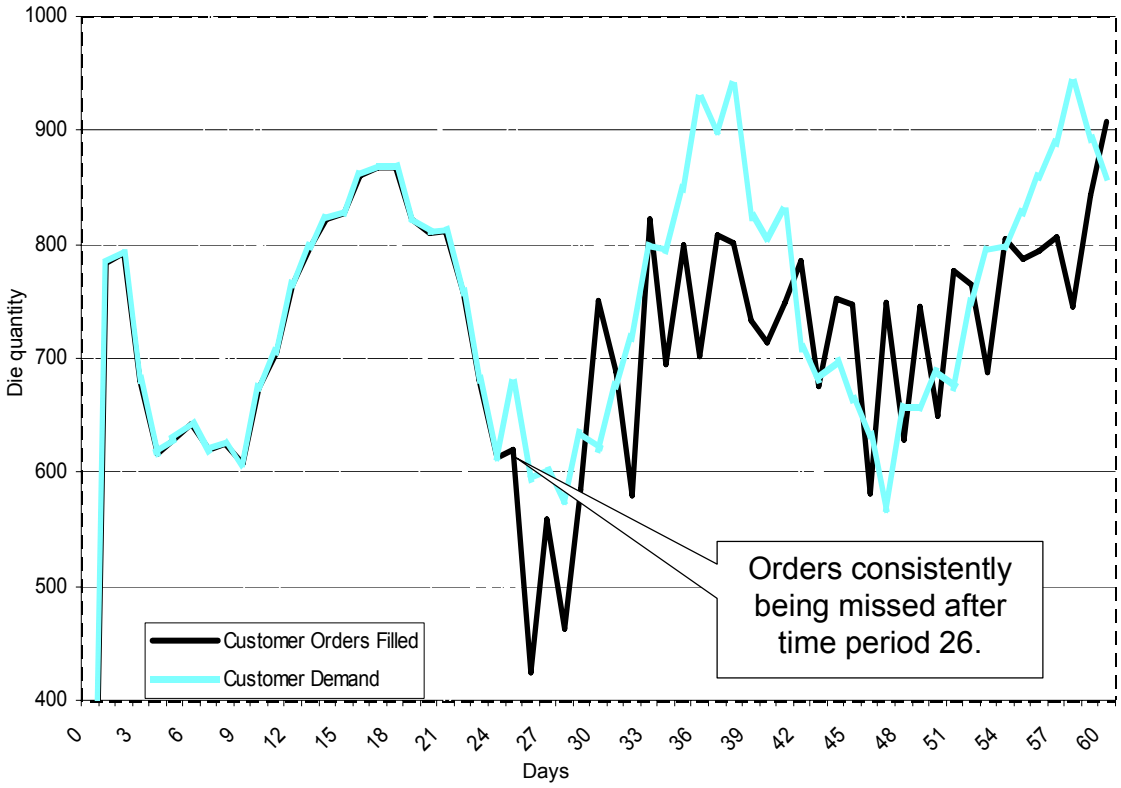


Figure 39. Optimistic Orders Filled versus Demand

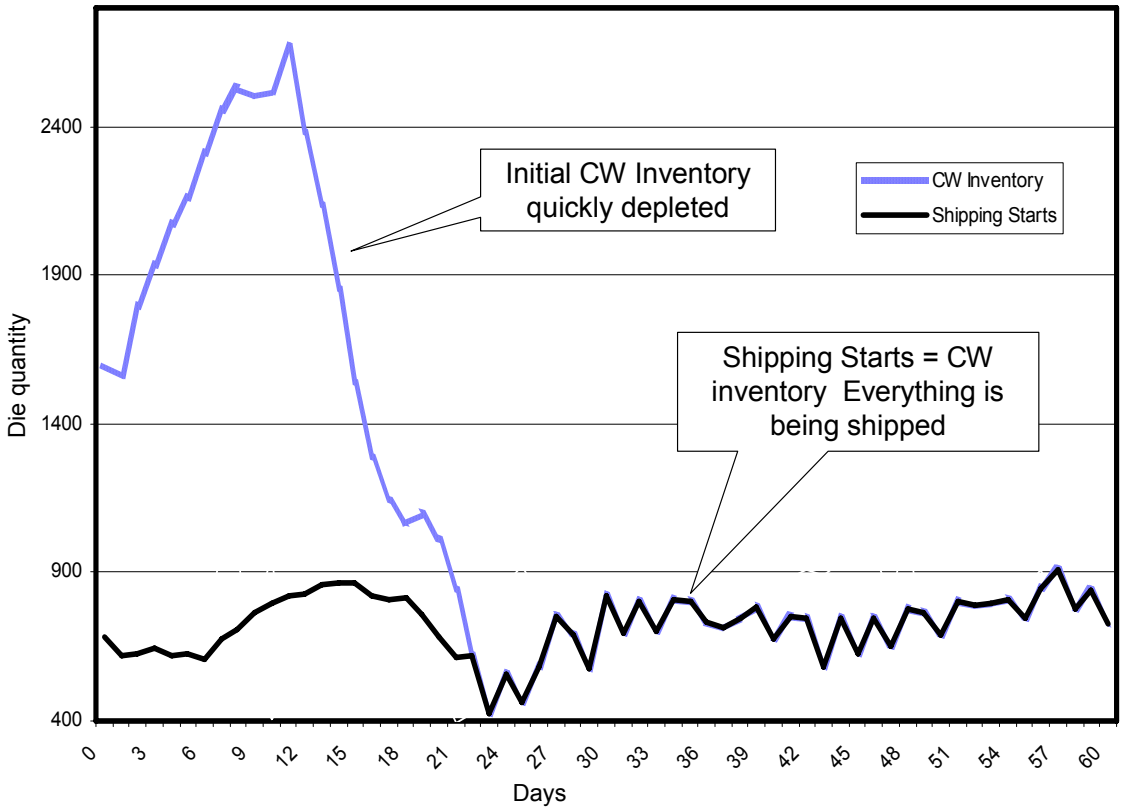


Figure 40. Optimistic CW versus Shipping Starts

5.1.7 Pessimistic Model Results

The pessimistic model reports the worse case values of TPT and yield from the simulation to the decision model. For this experiment, the data transformation for TPT was set to the MAX value obtained from the simulation, and the yield was set to the MIN. The results are shown in Figure 41. For this experiment, all orders were filled on time. However, the inventory built up to almost twice as much as the base case. The LP decision model consistently recommended more material than what was required. The

values the LP used for TPT and yield loss were larger than the actual average. However, it can be seen that the inventory leveled off instead of constantly rising. The LP ended up buffering enough inventories to compensate for the longer TPT's and bigger yield losses.

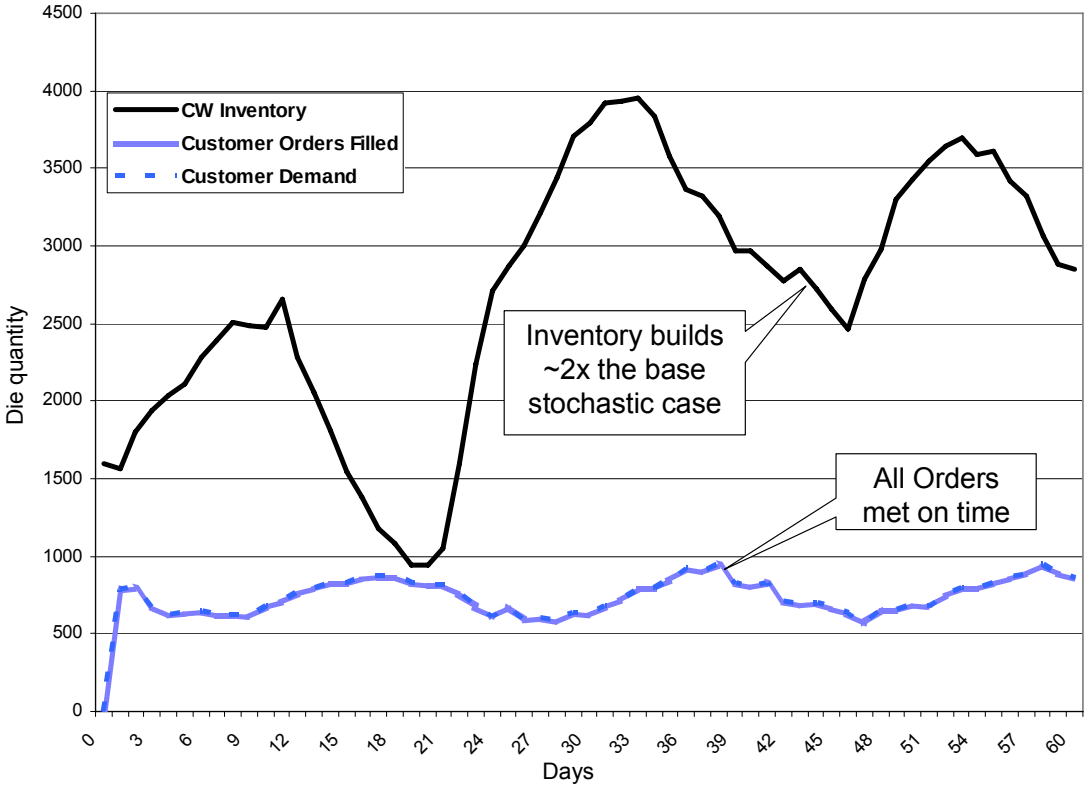


Figure 41. Pessimistic Model

5.2 Industrial Models

For the next category of case studies, the KIB environment was expanded to work on some actual supply chain network problems at Intel Corporation. The implementation used in the previous experiments was extended to integrate to the commercial Honeywell

Profit Suite™ set of controller applications. This application enables the use of model predictive control (MPC) as the optimizer to use for finding solutions.

5.2.1 Logistics Model

The first model can be classified as a logistics model. The MPC controller is used to calculate shipping schedules for products leaving the fabrication plants to assembly sites. The assembly sites vary in the products they can build and how they are supplied. Some of the assembly sites are company owned and the others are sub-contracted. There is an objective to keep the company-owned sites fully utilized and to minimize use of sub-contractors. However, sub-contractors provide a critical service of supplying extra capacity to cover demand upswings. The sub-contractors only have a finite amount of capacity. Different companies bid on the capacity to meet their assembly requirements. The capacity needs to be secured via contracts. If not enough capacity is secured, the company risks losing orders. If there is too much capacity, then money is also lost.

5.2.1.1 Integrating the Honeywell Controller application

The KIB environment requires updates in two areas. First, at the software level, a connection needed to be made with the Honeywell application. Second, at the model composability level, the KIB synchronization model capabilities had to be extended to work with the Honeywell optimization models and solvers. The Honeywell solvers were based on model predictive control (Camacho and Bordons 1995).

5.2.1.1.1 Model Composability with Honeywell

The KIB synchronization model needed to be extended to support the execution model of the Honeywell Controller. MPC models were utilized. The input and output to these models use single valued variables. The existing modeling for data mapping and transforms could be used for the data composition models. However, the MPC model had a different type of execution control loop for synchronization (see Figure 42). The MPC models run at discrete time steps with its own synchronization variable that needs to be toggled to tell the controller when to start the next step. A new variable was added for MPC synchronization. The KIB synchronization model could include this variable for specifying when data could be read and written and when a new controller cycle should be started.

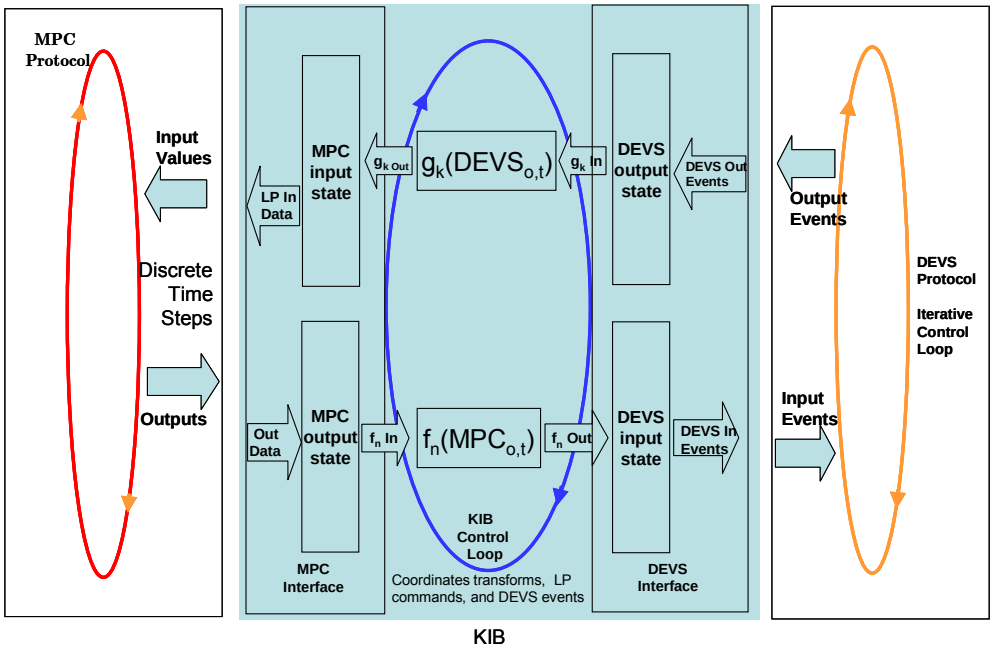


Figure 42. MPC Control Loops

5.2.1.1.2 Interoperability with Honeywell

The Honeywell controller supported the OPC standard (Iwanitz and Lange 2006) for connectivity. OPC provides a set of specification to support interoperability between industrial automation applications. It is built on top of Microsoft's DCOM protocol and the set of standard libraries is provided by the OPC foundation.

These libraries were utilized to support interoperability between the Honeywell application and the KIB. This was a straightforward implementation that only required a onetime development effort. Once the interoperability connection was made, the KIB capabilities supporting model composability between the controller and simulation models were available.

5.2.1.2 Sort to ADI Logistics Shipping Model Topology

The real world supply chain network topology is shown in Figure 43. For simplicity, shipping components are depicted as arrows. This topology has three factories, 27 shipping lanes, nine warehouses, and nine assembly sites. Each of the fabrication plants can produce up to 15 different products.

The MPC was implemented using the Honeywell Profit Suite™ set of applications. The DES had been developed using the DEVSJAVA simulation environment.

The MPC controller design required a different model instance for each product built from the fabrication plants. This resulted in 15 different product controllers that needed to run concurrently. The product controllers were coordinated using a dynamic,

real-time optimizer. This optimizer provided both dynamic coordination and steady-state optimization to the underlying 15 control applications.

A separate simulation model was connected to each of the product controllers, resulting in a distributed simulation. That is, each of the 15 controllers had a designated simulation model. Each simulation matched the topology shown in Figure 43, but had different stochastic distributions configured to match each of the products characteristics for the supply and demand forecast vectors.

The KIB had to coordinate the execution of the 15 different simulations, 15 different controllers, and one dynamic, real-time optimizer. It also needed to map and transform the data between each of the simulation and controller models.

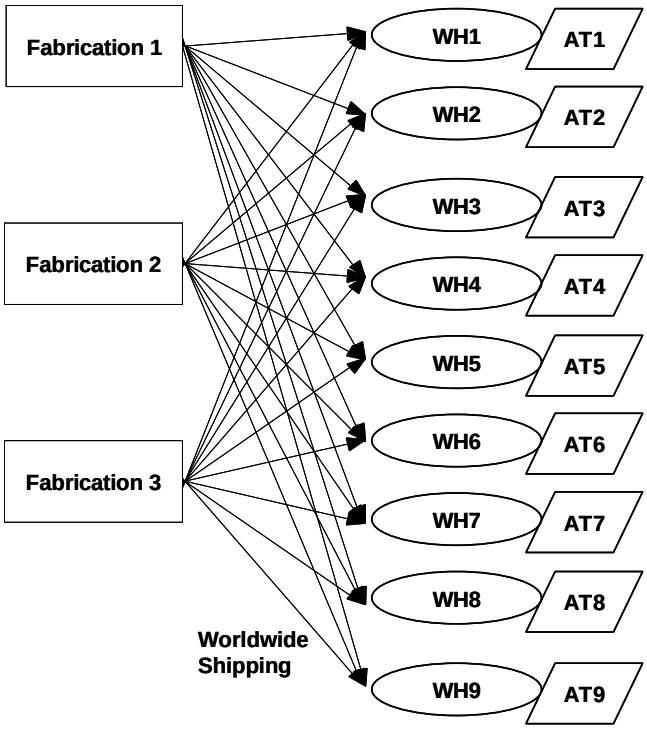


Figure 43. Supply Chain Network Topology

5.2.1.3 Product Routing

The product routing in this model was very simple. Products did not change names from the output of a fabrication plant to the input of an assembly test site. Products leaving a fabrication plant could go to any assembly site. Therefore, product to destination was a 1:1 mapping in this model. In Figure 43, a product leaving any of the three fabrication plants would maintain its characteristics and have the same name in any of the assembly test (AT) sites.

5.2.1.4 Simulation Scalability Concerns

The topology shown in Figure 43 required a simulation model with 64 supply chain network entities. The simulation model had three lot generators supplying the fabrication plants with simulated material, three fabrication plants, three end of fabrication inventory points, 27 shipping lanes, nine assembly warehouses, and nine assembly test lines. This simulation was run at a daily interval over one quarter (approximately 90 days). In addition, 15 instances had to be run in parallel to accommodate the controller design. This was the largest model run to date and we encountered scalability issues in the data collection for historical analysis. The simulation was setup to run 12 weeks of simulated time but was using all the available memory after simulating a few days. More efficient methods of logging the results had to be implemented. A design was put in that would log everything to disk every 10 intervals. This gave an acceptable performance. The simulation runtime was around 15 minutes for a full 90-day run.

5.2.1.5 KIB modeling

The KIB model enabled the mapping and transformation of data between each of the 64 simulation entities to over 200 input and output variables to the controller. The controller required single-valued variables, whereas the simulation enabled data models of object collections. The DEVS interface specification could be modeled much more compactly than that Honeywell MPC interface. For the DEVS interface, a few lines of specification for a collection-based structure could define the data interface to a single simulation module, whereas many variable specifications were required for the MPC.

The only KIB scalability concern was the size of the XML integration models. The overhead of running the KIB models was negligible in comparison to the simulation and controller. The controller required the most computing resources taking up to 10-15 seconds per solver iteration.

The KIB model ended up being 1300 lines of XML with our current design. The current XML editors on the market enable a fairly straight forward editing environment. However, visual modeling tools that can support the DEVS/MPC KIB modeling formalism would be much more powerful in enabling capabilities for syntax checking and model integrity.

5.2.1.6 Controller Development Approach

The goal of the simulation environment was to enable the development and validation of the controller prior to putting it in production. Although this kind of simulation-based design is common practice, the use of the KIB enabled experiments and

engineering of the complex interactions between discrete manufacturing processes and controller. A two-step iterative process was devised (Figure 44).

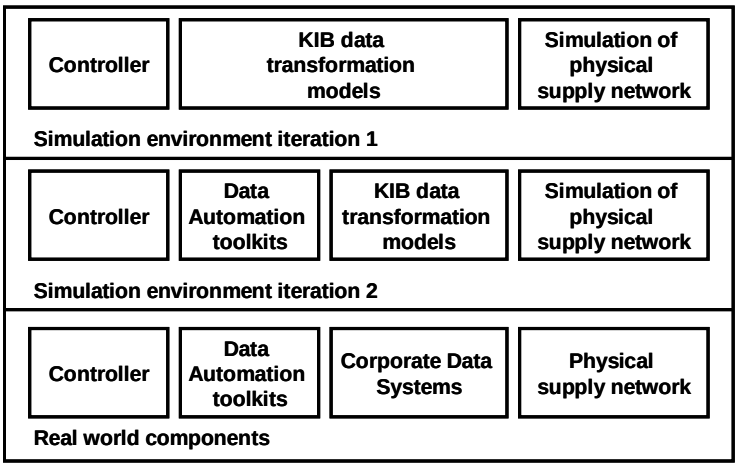


Figure 44. Simulation Iterations versus Real World

First, the KIB model generated data that was directly read and written into the controller variables. Second, the KIB model was revised to generate data in the same manner as the enterprise data systems. The production data automation toolkits would be developed against simulated data feeds.

The real world components section of Figure 44 shows the physical systems we needed to work with. There is the physical supply chain network, the corporate data systems that capture current states and information about the physical systems, some data automation software to transform the data into the format required by the controller, and the actual controller system.

The first iteration of experiments supported the development of the controller. Realistic stochastic simulations were run and validated against historical data. The controller was then developed and integrated with these simulations using the KIB. This

stage of simulation supported the development of the controller and its required data interfaces. Base issues were worked out, such as scalability and controller design.

In iteration 2, the KIB was changed to output data to the controller in a format that matches the corporate data systems. The production data automation toolkits were developed during this iteration. The same simulation of the physical models was used; however, the KIB data output to the MPC was different. This enabled testing and development of the production data automation toolkits for the controller.

After the two iterations of development, the controller and associated data automation toolkits were put into the production configuration.

5.2.1.7 Findings

5.2.1.7.1 KIB Benefits

The KIB enabled the development and experimentation of the MPC controller against the existing simulation environment. In particular, it facilitated the research and development for using MPC technology on an actual supply chain network discrete manufacturing problem. In the first experimental runs, it was found that the initial design for sending the supply and demand forecasts to the model would not work. The KIB enabled experimentation with several approaches by allowing different transformations of the data supplied from the simulation. First, the experiments involved sending data of differing granularity depending on how close in time it was (e.g. the first seven values of a supply forecast may be what is coming in the next seven days, the next three values could be what was coming in the following weeks). It was found that discrete values did

not work well. Therefore, the next set of experiments transformed the data into rolling averages (e.g. the first three values could be the hourly value and the next seven could be average daily for each week). That was the final design used by the controller.

The KIB also provided a test bed in which the data interfaces could be changed to simulate what was in actual production. This enabled the design and development of a production controller that could drop into the existing data systems.

5.2.1.7.2 Supply chain network Experimental Findings

On the first simulation/controller runs, it was found there would be scalability issues with the controller design and the simulation. Although each of the models ran OK in standalone mode, the integration highlighted invalid assumptions each had made about the other system. The controller had to be changed into a hierarchical design where separate instances controlled each product. Performance tuning had to be executed on the simulation to manage the large numbers of active simulation entities. Changes also had to be made in the simulation to correctly model the manner in which discrete lots are shipped from the factory. It was found this was an important behavior to simulate for the controller. The discrete nature of lot-sizing errors had impacts on how the controller needed to be tuned.

In the second iteration, we changed KIB models to exactly reproduce how data is sent and received from the production data systems. This resulted in development of the data automation toolkits prior to plugging the controller into production. The KIB enabled experimentation and refinement of the aggregation needed for forecast vectors. It

also highlighted invalid assumptions as to how data is provided from internal company systems versus subcontractors. The KIB provided a quick and efficient way to do experimentation with many different types of aggregation strategies.

Data Plots in Figure 45 show actual data results for one product in one of the controlled warehouses. The plot depicts the inventory target upper and lower limits along with the actual inventory. The objective is to keep the inventory between the limits. The limits are based on future demand forecasts. The first portion of the graph shows how inventory tracked in regards to the limits when the shipping was under manual control. Multiple planners were coordinating the shipping schedules from the fabrication plants to the assembly warehouses. The second portion of the graph starting at day 98 illustrates the performance under automated MPC control. The controller is performing the job of multiple planners while maintaining inventory at acceptable levels. This was the same controller that was developed and validated using the KIB simulation environment.

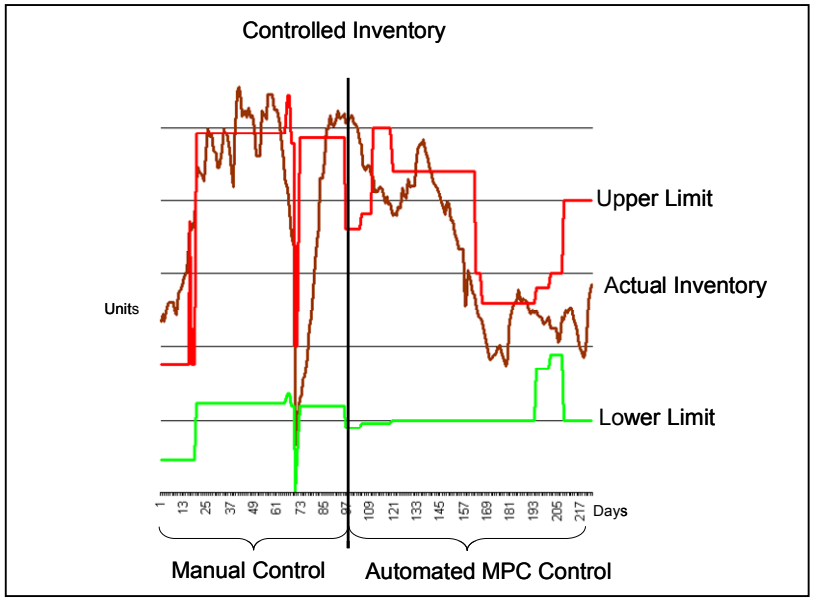


Figure 45. Manual vs. MPC Performance

5.2.1.7.3 Results

The MPC models and production data automation toolkits worked as designed on the first production run. This was a significant accomplishment since it was the first time MPC was used in an actual production instance of a discrete semiconductor manufacturing problem.

The controller design worked with a daily update to the shipping signals. The supply and forecast vectors needed daily granularity for the first few days and then could use weekly buckets for the next few weeks. When the controller was plugged into production, it was able to automatically keep inventory within limits at all warehouses as well as the current manual processes could.

The team that developed the controller and simulation comprised of three engineers, two senior control engineers, and one software/simulation engineer. Projects of this scale typically take more resources. Without the KIB modeling approach, the ability to experiment with different control frequencies and data sources would have been limited or impossible within the time constraints. More time for the simulation or a less robust controller put in at start of production would have been required.

5.2.2 Multi-Solver Experiments: MPC and LP

The next experiments evaluated how to design an MPC for controlling the builds out of a semi-finished inventory into a finished goods warehouse. Material in the semi-finished inventory can be set to different final configurations. The configuration may be specific to a particular customer, or it can be configured with a general set of

characteristics to match a certain type of product demand. The objective of the controller is to only build enough finished goods to meet actual customer orders. The controller must consider the available supply in the semi-finished goods, the time to process through finish, and the short term demands from the customers to generate build schedules. Due to the variability of customer demands and incoming supply, the controller is set up to maintain a buffer of finished goods. The controller works to keep the buffer within an upper and lower control limit.

The topology of the manufacturing line is simple; however, the product routings are complex. Each of the different types of semi-finish goods can make a subset of the finished goods. For example, assume there are three types of semi-finish goods. Assume Type1 can make products A, B, and C, type 2 can make products A and B, and Type 3 can make products B and C. Assume there is a high demand for products A and C, but only semi-finished goods of type 1 and 3 are being produced. If too much of semi-finished good type 1 was used building product C not enough would be left to meet product A demand. In this model, there were 111 semi-finished good product types with 716 possible combinations. Because of the complex routings, an LP was added to reduce the number of control variables needed by the MPC. The LP optimizes the selection of semi-finish goods to meet the finished good output schedules generated by the MPC.

5.2.2.1 Model Description

5.2.2.1.1 Topology

The integrated model topology is shown in Figure 46. This model requires the integration of LP, MPC, and simulations models. The KIB was extended to support the integration of all three using the existing logic.

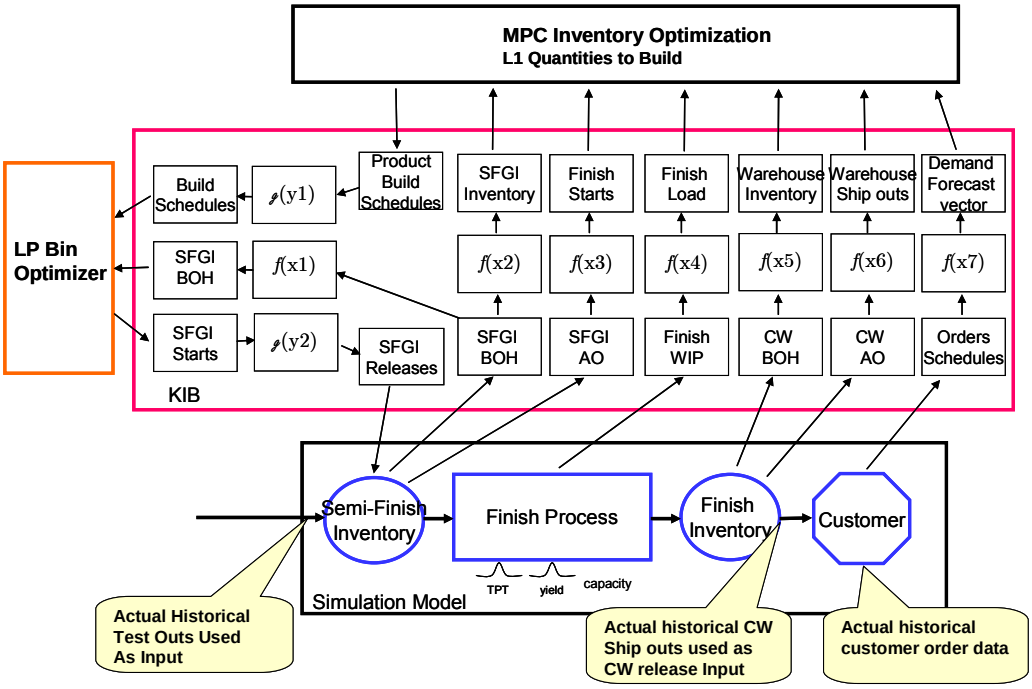


Figure 46. Multi-Solver Topology

The simulation models a portion of the final processes in semiconductor manufacturing. This includes a semi-finished inventory, the finish processes, the finish warehouse (CW), and a customer. The simulation has three historical data files for input. The first file provides actual historical data on incoming supply to the semi-finished

inventory. The second file provides actual ship data to the customer, and the third provides actual order data.

The controller objective was to create release schedules to keep enough supply in the FinishWarehouse to support shipments to the customer. The simulation modeled the stochastic yield and throughput time of the FinishProcess.

The MPC controller gets data on customer order forecast, what was shipped to a customer in the last interval, the quantities of semi-finished WIP, and the inventory of finished materials. From this, it calculates which finished goods should be built at each time MPC control interval. The semi-finished inventory could have 111 different products. Out of these products, 15 different types of finished goods could be built. These semi-finished materials can only build subsets of the finished goods. Overall, there were 716 possible combinations. The MPC controller modeling strategy used for these kinds of supply chain network problems would have required 716 different dependant variables. The controller would not easily scale to that size. To simplify the control design, a simple LP optimization was created that could use the MPC schedule and make an optimal selection of semi-finish material to support.

5.2.2.1.2 Product Routing

Figure 47 illustrates the complexity of the product routing possibilities for this problem set. For the simulation, the semi-finished inventory would output BOH and AO values for the 111 different products. It could accept release commands that specify which of the 111 products to use for making one of the nine target finish products. The

finish process could output WIP values for the 15 finish products as well as the finish warehouse and customer.

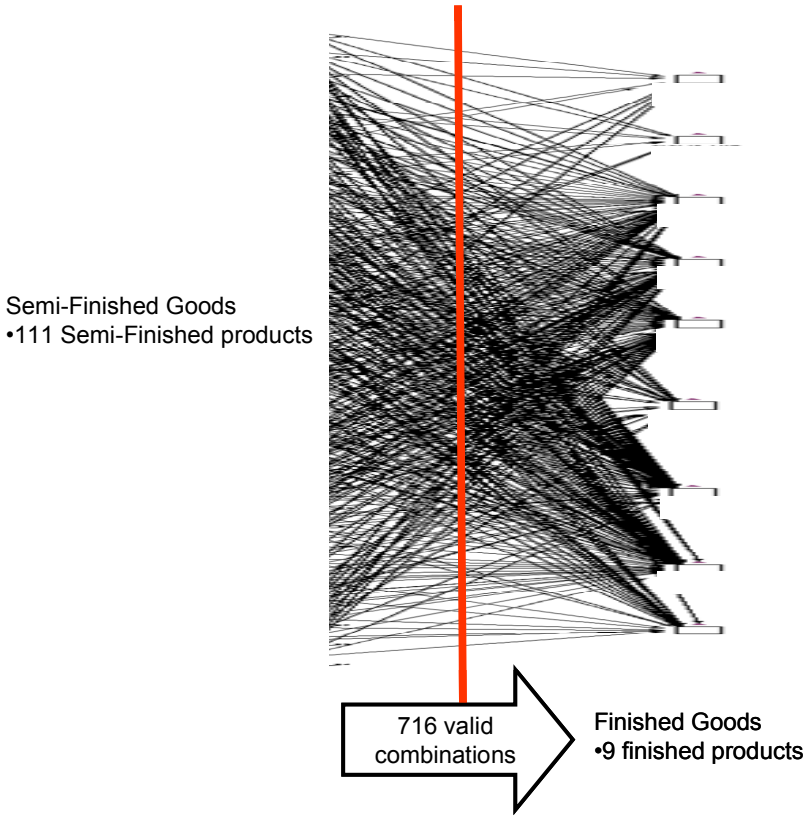


Figure 47. Product Routing Complexity

The controller, solver, and KIB transformation models would need to work with the same number of products. The large number product routing combinations is the most difficult level of complexity in this set of experiments.

5.2.2.2 KIB Multi-Solver synchronization model extensions

The KIB needed to be extended in this set of experiments to support two solvers. The data interfaces and transform functions were already in place from the previous experiments. The ability to synchronize the MPC, LP and simulation execution did not exist. The KIB control modeling language and underlying execution algorithm needed to be extended.

The data flows that needed to be supported for this set of experiments are shown in Figure 48. The simulation, LP, and MPC could read and write their associated states in the KIB. The transformations that needed to be supported are a subset of all possible transforms. The data transformations that needed to be modeled were determined by how the models needed to be synchronized. The logical flow of the composed model is: 1) the simulation runs creating a new state for input to the solvers, 2) the MPC calculates how much of each finished product needs to be started from semi-finished goods inventory (SFGI), and 3) the LP solver selects which semi-finished products should be used to support the MPC schedules. The LP also needs the quantities of each semi-finished good that is currently available, so it needs this data from the simulation. The LP output then goes to the simulation. This implies that data transformations need to happen from simulation to MPC (DEVS➔MPC), MPC to LP (MPC➔LP), simulation to LP (DEVS➔LP), and then LP to simulation (LP➔DEVS).

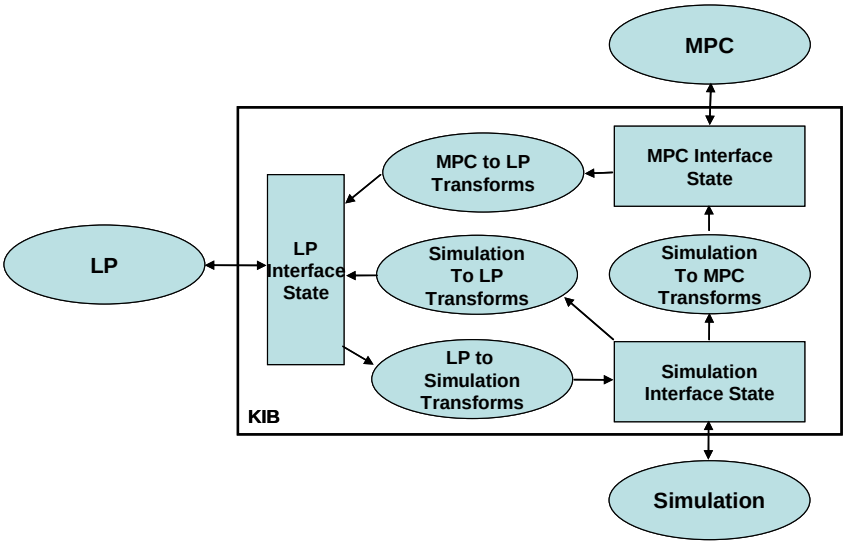


Figure 48. Data Flows in the Composed Multi-Model

Figure 49 shows the model synchronization that needed to be supported for this set of experiments. It is an iterative loop coordinating when to do the data transforms and when to execute the solver and simulation models. First, the simulation runs; second, the simulation to MPC transformations are executed. Third, the MPC solver is run; fourth, the MPC output is transformed to LP input. Fifth, the simulation output is transformed to LP input. Sixth, the LP solve is run; and finally, the output of the LP is transformed into simulation data input.

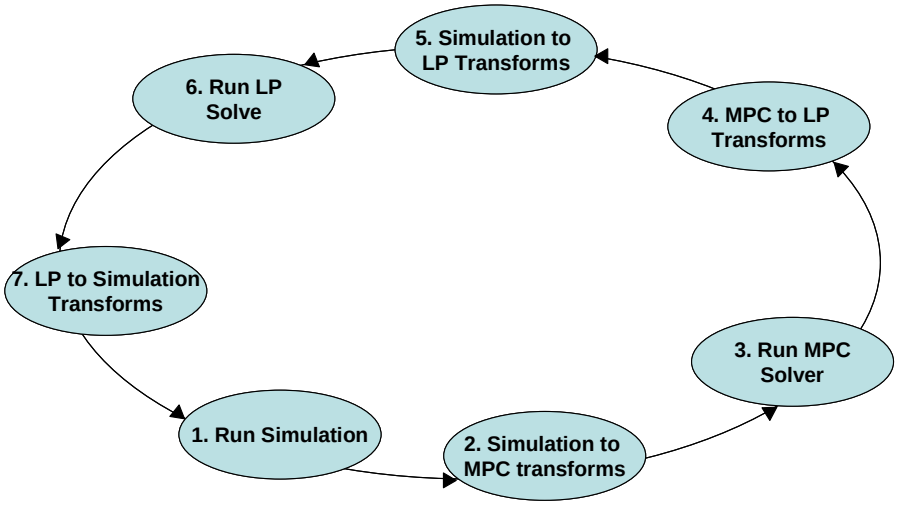


Figure 49. Control Flow of the Composed Multi-Model

The simulation to LP transformation in Figure 49 could have been configured anytime after step 1 and the composed model would have executed correctly. The transformation was designated as step 5 to make the synchronization model easy to follow. This is pointed out to emphasize the flexibility of the KIB control modeling. The KIB modeling language was extended to enable the specification of when the models and data transformations were executed. The specification is a sequential order of actions to take place.

5.2.2.3 Experiments

The experimental model and environment was illustrated in Figure 46. The simulation scenarios would be driven by using actual historical data for incoming supply and for outgoing CW warehouse ship-outs. These scenarios would represent a historical

snapshot of actual supply/demand conditions. There was an initial iteration of simulation runs that used historical semi-finish inventory releases to validate the models. The simulations models were validated to match the behavior of the finish manufacturing lines. After this validation, all remaining experiments used the outputs of the solvers for generating the release commands to the semi-finished inventory.

The historical data used for input was for 90 days at an hourly granularity. Each file had 2,160 data points. The simulation would run at an hourly interval. The solvers could be run at any hourly multiple. The experiments were run using hourly or daily control. Many of the data inputs needed to be input at daily or weekly granularity.

The metrics used to determine the solvers performance would be a measurement of inventory levels and customer order response. The inventory levels in the semi-finished and finished warehouses would be recorded for each simulation iteration. The difference between the due date and the time each customer order was filled would be tracked. The success of the solvers would be a measurement of their performance meeting orders on time compared to historical results. The solvers need to match customer service levels while at the same time use supply as well or better than what was achieved historically. The criterion for measuring how supply was utilized was based on how well the controller kept finish inventory levels at a minimum while maintaining customer service levels.

In the initial experiment runs, the controller was given a perfect demand forecast from the customer entity in the simulation. This enabled the development of the controller against an ideal case. After the controller was performing as expected,

stochasticity was added to the customer demand. The error from actual demand increased farther out in time.

As the controller development progressed, it was found there were several important data inputs that needed to be tuned. First, how the demand forecast is sent to the controller and second, how to specify inventory limits and how often to update them. For the demand forecast, there were tradeoffs involving sending actual values, sending rolling averages over segments of the forecast horizon, and the granularity of the segments. For the inventory limits, it was determined the controller worked better when limits were specified as a function of demand. For example: Setting the finished goods lower limit as a sum of orders in the next two weeks and the upper limit as the sum of orders in the next three weeks made the timing of how often to update the limits critical. If limits were updated too frequently, the controller would become unstable; if time between updating limits was too long, customer service levels suffered. Many experiment runs were performed to find the best operating regions of demand forecast and inventory limits settings.

5.2.2.4 Findings

The objective for this set of experiments was to determine the viability of using MPC for semi-finished goods scheduling. The multi-formalism modeling environment enabled the development of the controller against data sets of the scale seen in actual manufacturing. It enables the discovery of data requirements, different methods for configuring the controller, and the scale of the real world problems.

5.2.2.4.1 Simulation Scalability Concerns

The simulation needed to be redesigned in several areas to address performance issues. The large number of products and number of simulation iterations required increased runtimes from less than 15 minutes to over 3 hours. The large number of product combinations exponentially increased the number of entities required to be sent on the data and control ports on each simulation interval. For the data ports, the simulation would send a separate lot entity out each port at the end of each interval. For the semi-finished inventory, this meant hundreds (up to 716) could be sent at each hour. Each one of these messages generated an external event on the connected port. For data, this was always a single port. For the control messages, a separate status message was sent out for each product. For semi-finished inventory, this mapped to 111 BOH messages and up to 716 AO messages per simulated hour. Each of these messages generated an external event on connected ports. For the control messages, this meant an external event generated for all entities in the simulation. Since the lot and status messages occur at the same logical time instant, the message count was reduced by consolidating the data going to the same destination into a single container message. Using this strategy, the worse case message count was reduced by a factor of 716, the number of product mappings. The run times decreased from hours to minutes for a full 2,160 hour run.

The other scalability problem encountered was the size of the simulation output files. Many of the files exceeded one million lines. Previously, all offline data analysis

had been accomplished using Microsoft Excel, which has a file limit of 64,000 lines. Offline tools had to be created to summarize the data in an easily analyzable format.

5.2.2.4.2 Data input for controller

The previously described experiments were invaluable for determining the requirements to build a MPC solution for final product configuration. MPC had not been applied to an industrial application in semiconductor supply chain domain before. The experiments highlighted the upper bound of allowable control variables in the current MPC implementation. The KIB test bed allowed experimentation using a hybrid solution with an independent LP to address the limit on control variables.

Two important areas were researched by the MPC engineers, how to set inventory control limits and how to pass a demand forecast on discrete process flows with characteristics seen in semiconductor manufacturing.

5.2.2.4.3 KIB Benefits

The KIB provided a test bed that enabled many different controller designs to be tested against a single validated simulation. The KIB facilitated experiments using different controller frequencies, data aggregation algorithms, and data feeds. There was a onetime setup and validation on the simulation. The KIB insulated changes to the simulation by providing the modeling capability to match the semantics and control with the LP and MPC models.

In addition, the effort to try multi-level solves was simplified by the KIB. Small extensions were made to existing capabilities. This one-time effort supported many iterations of control design development.

5.2.2.4.4 Data results

A small example of data results from these experiments is shown in Figure 50. This figure illustrates inventory levels on a product historically compared to levels using MPC control. The historical data is what was actually seen using the current production processes. The simulated inventory levels are what were seen using the MPC and LP solvers to manage semi-finished releases. There are also upper, lower, and mid-limit values shown. The objective is to keep the inventory as close to the mid-level limit as possible. As seen in Figure 50, the historical data far exceeded the limits around 50% of the time. These experiments show that MPC may be a good fit for these types of problems. The assumption made is that the simulated demand forecast errors were good enough to represent the errors actually seen. A second round of simulations is planned for the future using the real demand forecasts from day-to-day when the production control environment was generating starts. This will give a more realistic comparison as to how the MPC control would perform during actual production.

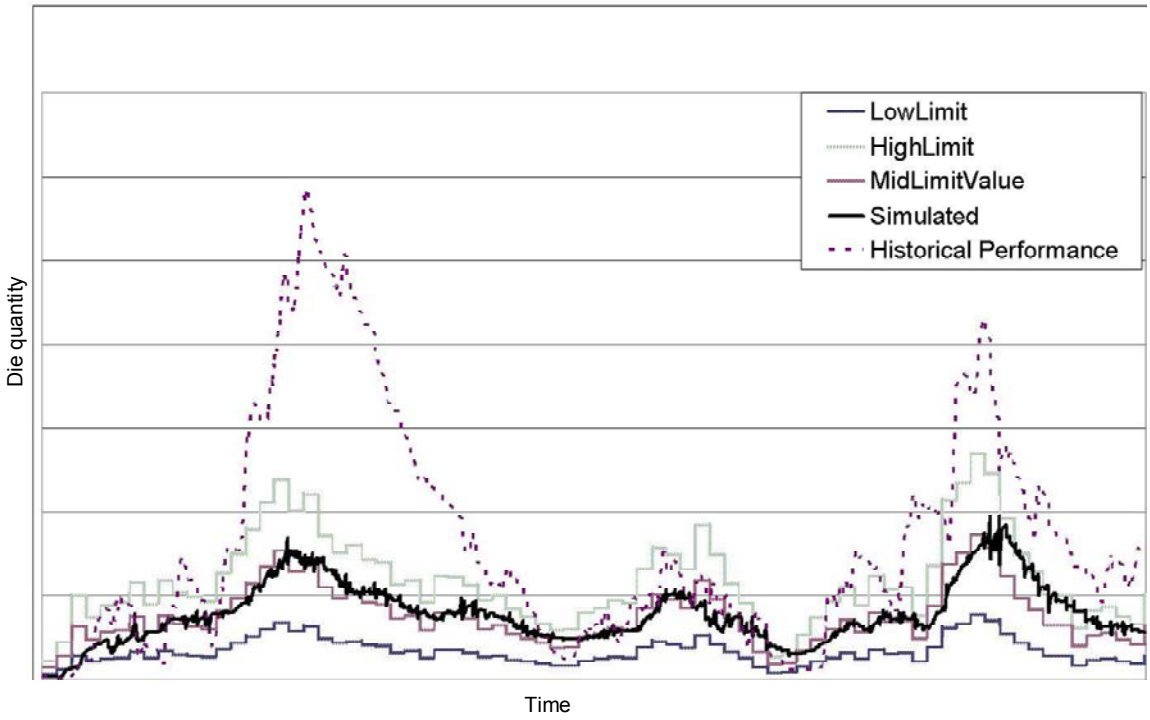


Figure 50. Simulated Control versus Actual Historical

5.2.3 Current Work

A new set of experiments are being run that combines the previous two models. The new models require the complex logistics seen in the first industrial case study and a product mapping hierarchy almost as complex as the second industrial case study. The same KIB and simulation environment are being used. The data and models are confidential so they cannot be described. The results from using the KIB and simulation environment are promising.

The KIB enabled enhancements made to the commercial controller to be tested prior to being put into production. The resulting enhanced MPC is the largest

commercial controller of this type ever put into production. The end users are confident enough with the data results provided by the KIB test bed to use it for pre-production controller validation.

6 SYSTEM AND KIB DESIGN

This chapter will describe the design and implementation of the KIB environment created to demonstrate the multi-formalism methodology and enable the Semiconductor Supply Chain Network experiments. First, the software applications and how they were integrated with the KIB will be described. Next, the architecture of the LP/DEVS KIB and the design approach taken for enabling interoperability while supporting model composability will be explained. Then details of the KIB software design will be shown in UML. Finally, there will be discussion on specific Semiconductor Supply Chain Network modeling concerns for integrating DEVS, LP, and KIB models.

6.1 Software Application Architecture

The software applications used to run the experiments in Chapter 5 were ILOG OPL Studio for modeling and executing LP's (ILOG 2008), Honeywell Profit Suite (Honeywell 2008) for modeling and executing MPC's, DEVSJAVA for modeling and executing DEVS simulations, and the KIB implementation described in this chapter. The applications and how they are connected is illustrated in Figure 51. The applications are connected to the KIB using various protocols chosen because they are directly supported by existing implementations of the applications.

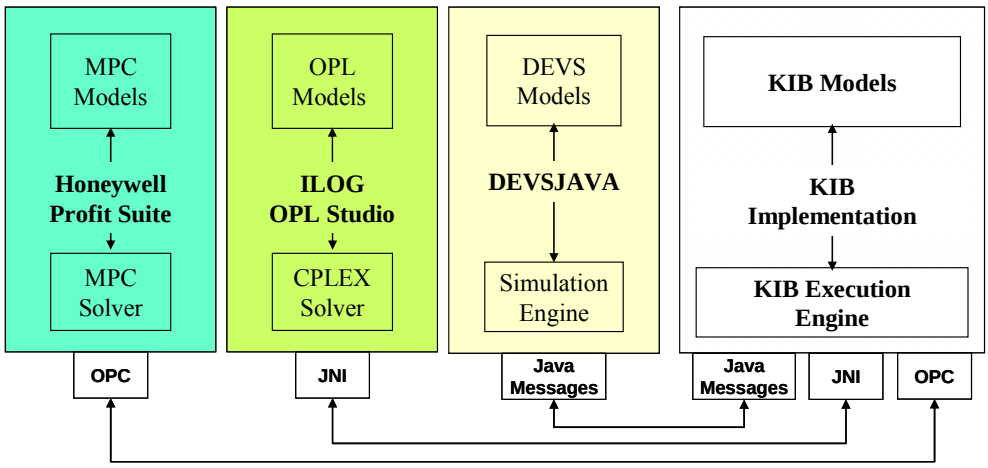


Figure 51. Software Application Architecture

The DEVSJAVA application connects to the KIB through JAVA messages. The DEVSJAVA and KIB implementations can call each other directly through JAVA messages and ports. This is because the KIB and DEVSJAVA are both implemented in JAVA and for this KIB implementation they share the same process space. The KIB was designed for an easy separation using JAVA RMI, but was not necessary for any of the experiments or case studies. This would become a necessity if using a different simulation environment was desired.

The KIB communicates to the OPL Studio application through JNI. OPL studio includes a JNI API with its commercially available product. This interface allows an external application to directly populate initial states into an OPL model, execute a solve, and read the results.

The communication to the Honeywell controller is executed through a protocol known as OPC (Iwanitz and Lange 2006). OPC is an interoperability protocol

maintained by a standards committee for the process control industry. OPC provides a standard way to send and receive data and alarms from equipment sensors and process controllers. The OPC protocol is built on top of DCOM. Standards compliant DCOM libraries are available (OPC-Foundation 2008).

The KIB has been implemented in a straightforward manner to enable connectivity to many different protocols so experimentation can leverage existing environments supporting various types of modeling. There are advantages to leveraging existing implementations. First, the existing implementations have been optimized to work very well within their domain from both performance and usability aspects. Second, there is an existing group of experts that know how to use the environments. And third, the existing models that had previously been developed can be reused.

6.2 Interoperability Approach

The purpose of this KIB implementation is to create an environment that enables the development and validation of different Semiconductor Supply Chain Network control models against accurate simulations. We would like the environment to be easily extended to a variety of modeling tools utilized in the industry. Since the different planning and simulation tools that exist today use a variety of API's to enable interoperability, the KIB needs to have a flexible implementation to enable adaptability.

It is common practice for the approach shown in Figure 52 to be used for interoperability for enterprise planning and execution applications. Organizations using groups of experts develop a standard message structure to use for communicating between applications. An example found today for supply chain network and enterprise

systems would be OAGIS (OAGIS 2008). This approach has many benefits for the integration of large software applications in which the application suppliers may come from many different vendors. It eases the integration efforts by having applications communicate to one standard message protocol. This approach requires all applications that need to interoperate communicate use this standard. Model composability mismatches need to be addressed within each application. In Figure 52, this composability would need to be addressed within the *message transform logic* layer.

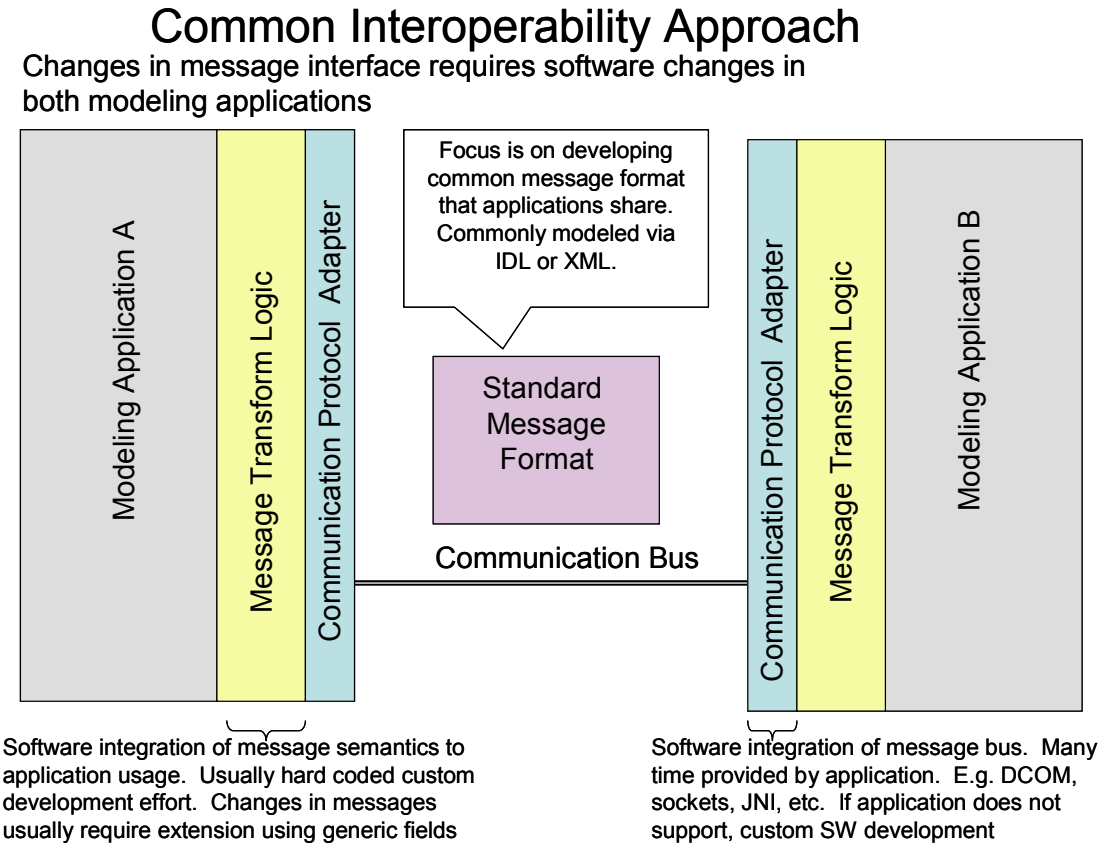


Figure 52. Industry Standards Approach to Interoperability

In the environment being implemented for our KIB, we do not want to change any of the applications being connected. Rather than trying to enforce a single interoperability standard across all applications, a generic adapter class is created that can connect to a particular protocol and then expose the data to the KIB interfaces. The KIB transform specification can then address any mismatches in the data.

The approach taken for implementation of the KIB is shown in Figure 53. Since we are looking at the integration of models and require the interactions and data exchanged between them to be flexible, we have developed an architecture that allows data to be sent from the applications in their native form and the transforms between the two are modeled in the KIB.

This approach requires an adapter to be built that can communicate with each modeling application. In this way, existing techniques are used to solve the interoperability. The KIB is used to enable model composability through its modeling layer above the interoperability.

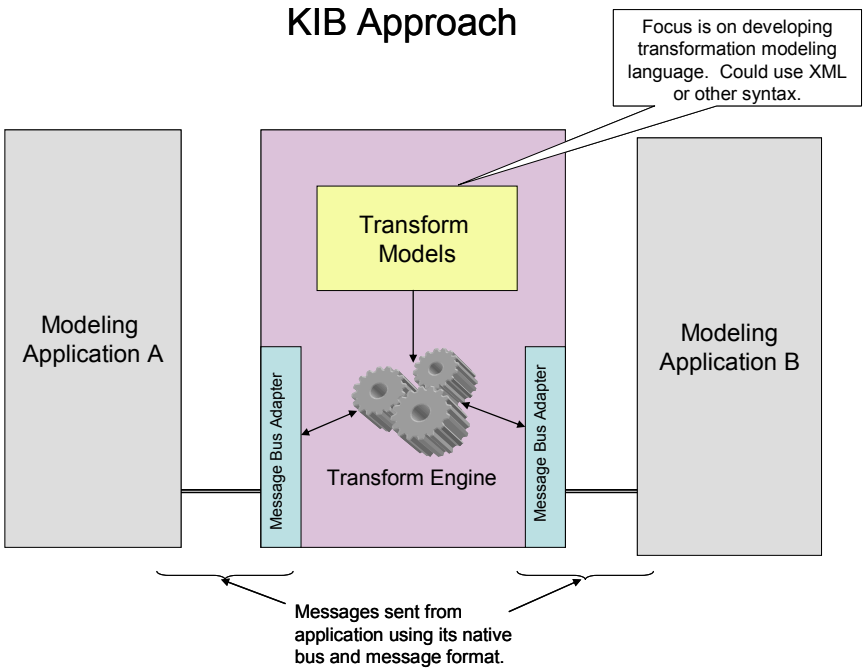


Figure 53. KIB Approach to Interoperability

If two different applications use the same interoperability technique and they are supporting the same type of modeling formalism, the adapter could be reused.

6.3 KIB Software Architecture

The main components for the implementation of the DEVS/LP KIB architecture are shown in Figure 54. This figure illustrates the KIB configuration for connecting to OPL Studio and DEVSJAVA. The KIB is made up of custom adapters (OPL Adapter, DEVS Adapter), the Generic Interface, Execution Module, a Data TransformEngine, a Model File Reader, and a Data Store.

The Execution Module coordinates the initialization and the control of all the other KIB components. It is the first component to be loaded when the KIB application

is started. It coordinates the execution by reading the Control Model from the Data Store and receiving Sync Events from the KIB generic interface. The Control Model is loaded into the Data Store when the KIB XML file is read.

The Data TransformEngine module performs the data transformations between the models. The transformations to be used for each variable are modeled in the KIB XML file. Transformations are performed whenever instructed by the Execution Module. The Data TransformEngine will read the current state from one formalism state model and transform the data to the correct representation for the other. The input and output variables to use is also configured in the KIB XML file.

The Generic Interface can read and write the state data from the Data Store and then supply this data to the adapter module. The Generic Interface can supply the KIB ExecutionEngine with Sync Events. There is a Generic Interface instantiated for each custom adapter. Which KIB generic interface supplies the Sync Event is determined by the Control Model.

. The custom adapters are written for each type of formalism and interoperability protocol the KIB needs to interface with. A different adapter has been implemented for both OPL Studio and DEVSJAVA.

The Data Store holds state data from the LP and DEVS models in generic data structures. Each of the formalisms has its own storage area. The Data TransformEngine can move this data between the storage areas using transforms modeled in the KIB model file. The Data Store schema is set at initialization from the KIB model file. Data values are written to the Data Store when the different models start executing.

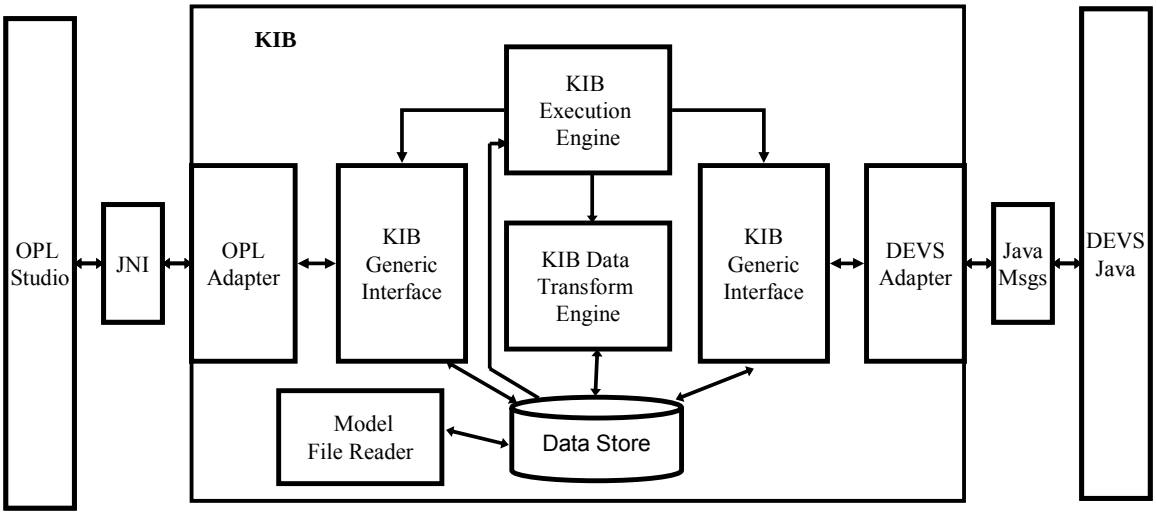


Figure 54. DEVS/LP KIB Architecture

The model file reader reads the XML KIB model configuration file at startup and keeps a representation in the Data Store. The KIB Data TransformationEngine uses these models to transform data during execution; the LP and DEVS interfaces utilize the model to map the generic representation of data to the specialized format required for the formalism.

6.4 KIB Implementation

The top level class diagrams for the KIB package is shown in Figure 55. The KIB is separated into the following classes: ExecutionEngine, TransformEngine, ModelFileReader, and the InterfaceConfigs, which includes composition of other classes. The details of each class will be described and then some sequence diagrams will be shown to illustrate their interactions.

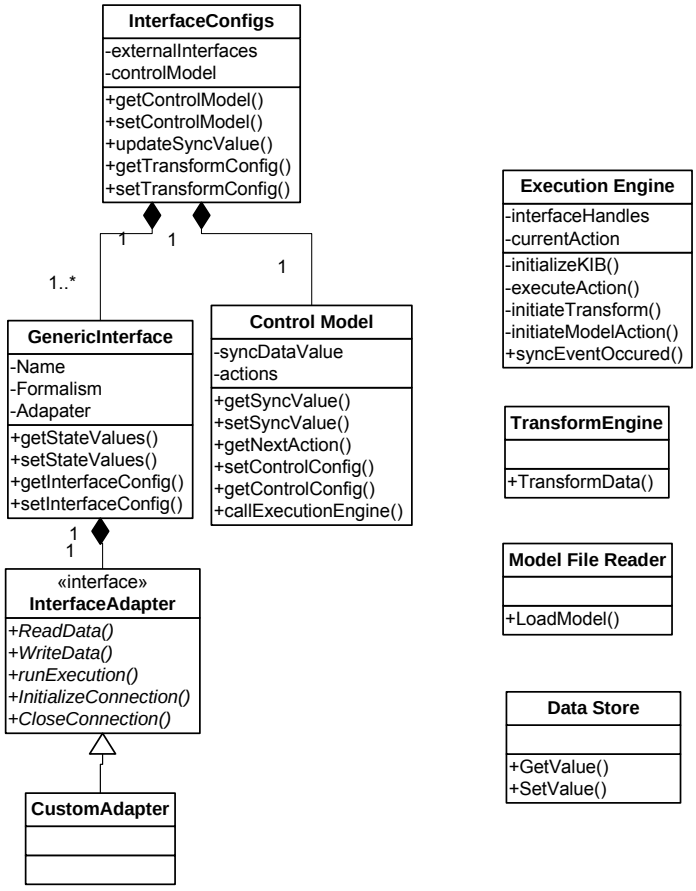


Figure 55. Main KIB Classes

6.4.1 InterfaceConfigs

The InterfaceConfigs class maintains the configurations and state values for each of the external KIB interfaces. This class is a composition that includes a collection of externalKIBinterfaces and a Control Model. Each of the GenericInterfaces includes a customAdapter to implement the InterfaceAdapter interface.

The InterfaceConfigs class has two private attributes. They are references to the GenericInterfaces and the Control Model. These attributes can be set by the setControlConfig and setInterfaceConfig methods.

Attributes

- GenericInterfaces.
  - *Type:* Collection GenericInterfaces.
  - *Description:* Reference to the active GenericInterfaces instantiated when the KIB XML model was loaded by the ModelFileReader.
- controlModel
  - *Type:* ControlModel
  - *Description:* Reference to the Control Model. Value populated when the KIB XML model was loaded by the ModelFileReader.

Methods

- *getControlModel:* Returns reference to the Control Model
- *setControlModel:* Sets the Control Model to object passed in
- *updateSyncValue:* Updates the sync value in the Control Model
- *getTransformConfig:* Returns the transformation configurations between the interfaces.
- *setTransformConfig:* Sets the transformation configurations.

6.4.2 GenericInterfaces

The GenericInterfaces class defines the interface to one of the connected applications supporting the development and execution of a formalism model (e.g. DEVS or LP). The interface provides the latest state values sent or received to the application in the generic KIB format. The interface adapter communicates with the GenericInterface through method calls.

Attributes

- name
  - *Type:* String
  - *Description:* Used to name the interface. The InterfaceConfigs class uses the name to identify the sending and receiving GenericInterfaces for the transformations.
- Formalism
  - *Type:* String
  - *Description:* Name of the model the interface supports
- Adapter
  - *Type:* InterfaceAdapter
  - *Description:* Reference to the adapter assigned to this GenericInterface

Methods

- *getStateValues:* Returns the current state values

- *setStateValues:* Sets the state values to what was passed in
- *getInterfaceConfig:* Returns the configuration of the interface. This includes input and output variables, the variable type definitions.
- *setInterfaceConfig:* Sets the interfactConfig to the object passed in. This method is called by the KIB Model File Reader. The interface configuration is read from the XML file.

6.4.3 ControlModel

The Control Model class is used by ExecutionEngine to track the current state and supply the next one. The current state includes the value of the synchronization variable, the model execution frequency configuration, and the next control action that should be performed.

Attributes

- syncDataValue
  - *Type:* Object, details of which is defined by the KIB model
  - *Description:* Maintains the value of the synchronization variable
- actions
  - *Type:* Circular List of Strings
  - *Description:* Defines the order and sequence of actions for the ExecutionEngine. The sequence is defined in the KIB XML model.

Methods

- *getSyncValue:* Returns the current value of the synchronization variable
- *setSyncValue:* Sets the sync variable to the value passed in.
- *getNextAction:* Returns the next action that should be performed. This would be either data transformation or the schedule model execution.
- *getControlConfig:* Returns the control configuration.
- *setControlConfig:* Sets the current control configuration to the value passed in. The control configuration is read from the KIB XML file.
- *callExecutionEngine:* Private method. Calls the ExecutionEngine when the sync variable changes value.

6.4.4 InterfaceAdapter

The InterfaceAdapter interface must be implemented by the custom adapter. For OPL Studio a custom adapter was created that could communicate over JNI and call OPL's methods defined in their API. For Honeywell, a custom adapter was created that could communicate via OPC. For DEVSJAVA, a custom adapter was implemented that could communicate directly via Java methods.

Attributes

- name
    - *Type:* String
    - *Description:* Name of the adapter

Methods

- *readData:* Used to read data from the external model. Customized to work with the application protocol.
- *writeData:* Used to write data to the external model.
- *runIteration:* Run an iteration of the model. For DEVS, this would be one simulation iteration as defined by the model. For LP, this would be the initialization of a solve. For Honeywell, this would be a new iteration of an MPC solve.
- *initializeConnection:* Initializes connection with the external application. Performs all the low level methods required by the protocol.
- *closeConnection:* Closes out the connection with the external application.

6.4.5 ExecutionEngine

The ExecutionEngine coordinates the control across all other objects at runtime. The ExecutionEngine is also implemented as the main class. It instantiates all the other KIB objects at runtime.

Attributes

- interafaceHandles:
  - *Type:* InterfaceAdapter
  - *Description:* Reference to the interface adapters. Communicates directly to the adapters for commanding them to read, write, or perform an execution.

- currentAction
    - *Type:* String
    - *Description:* The current action being executed. Can be transforms or instructions to send to the external applications.

Methods

- *Initialize:* Private method called at startup. Instantiates all objects and request the initial state from the first model to be scheduled for execution.
- *executeAction:* Private method called when starting a new action.
- *initiateTransform:* Private method to call the TransformEngine.
- *initiateModelAction:* Call method on interface adapter objects.
- *syncEventOccurred:* Method called when a sync data variable changes. Starts a new cycle of actions.

6.4.6 TransformEngine

The TransformEngine performs the data transformations. It reads the input state from the source model and writes the transformed data to the destination model. It is called by the ExecutionEngine.

Attributes: none

Methods

- *transformData:* Performs the data transformation. A parameter specifies the source and destination models. This method will lookup the

transformation configurations, determine if it is time to perform transform, read source data, transform the data, and write to destination.

6.4.7 KIBModelReader

Attributes: none

Methods

- *loadModel:* Loads the XML KIB model into the InterfaceConfigs objects. This method is only called at startup time. It reads the XML KIB model and then calls the setInterfaceModel, setControlModel, and setTransform model methods on the GenericInterfaces, the ControlModel, and the TranformEngine classes.

6.4.8 Data Store

This class function is to store the state data and KIB models into data structures that are accessible to the other KIB objects. Data elements are set and retrieved by String name.

Attributes: none

Methods

- *GetValue:* Gets a value from the Data Store.
- *Set Value:* Sets a value in the Data Store.

6.4.9 Initialization Sequence Diagram

A sequence diagram for the KIB initialization is shown in Figure 56. This initialization is coordinated by the ExecutionEngine object. At startup, the ExecutionEngine first instantiates the TransformEngine, the ModelFileReader, and the InterfaceConfigs objects. Next, it calls the ModelFileReader to load the XML KIB File. The ModelFileReader then parses the XML KIB file and writes the InterfaceConfigs object. The InterfaceConfigs object then instantiates each GenericInterface, which in turn instantiates the CustomAdapter. After that is complete, the InterfaceConfigs stores the interface models read by the ModelFileReader into each of the GenericInterface objects. When this is complete, the ExecutionEngine is notified that the interfaces have been initialized. The ExecutionEngine will then ask for the initial state from the first model configured in the Control Model configuration, which is done through communication to the interface adapter. After the initial state is populated, the interface adapter notifies the ExecutionEngine that it is complete. The ExecutionEngine then gets the first control action and the main loop of control is started as described in Chapter 3.

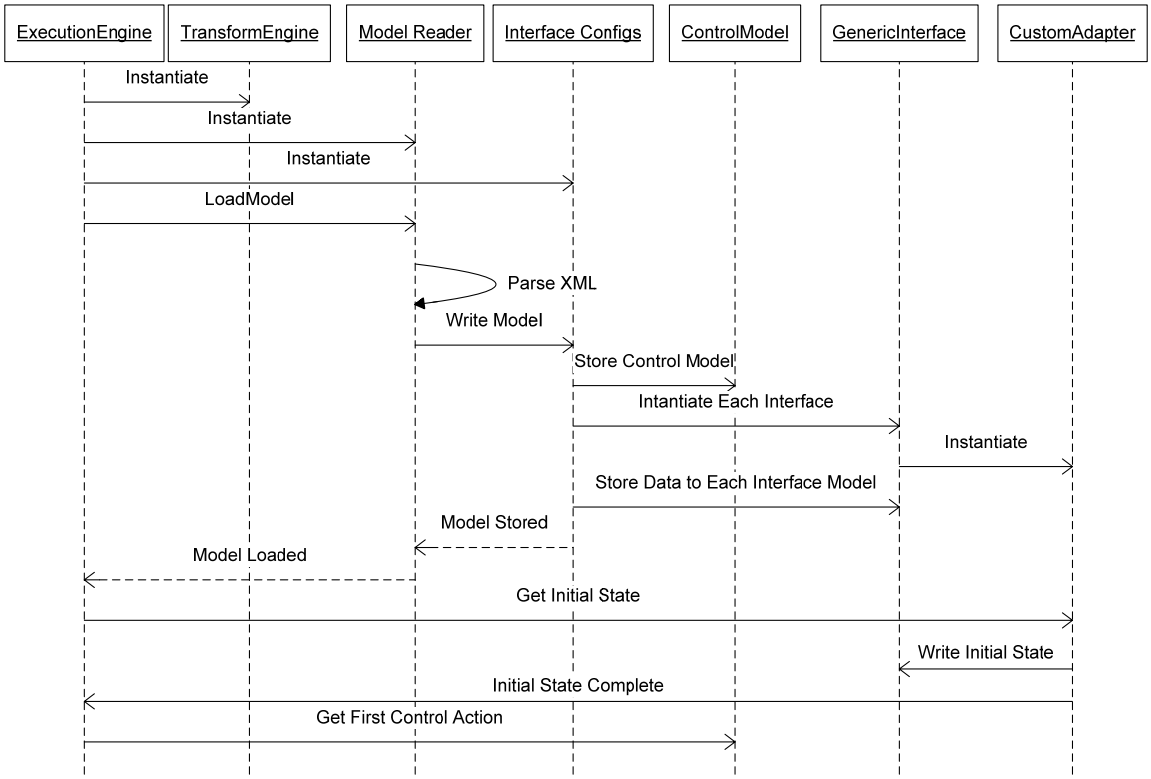


Figure 56. Use Case Sequence Diagram for Initialization

6.4.10 Sequence Diagram for Execution initiated by Sync Event

The sequence diagram in Figure 57 illustrates what happens when a Sync Event occurs. See Chapter 3 for details on how this occurs. The Sync Event can transpire when the state data is updated from an external model. The Sync Event is based on some data changing from the external model. Which data is used for this is configured in the KIB XML model. In this sequence diagram, the state data is first updated to the GenericInterface object, which notifies the InterfaceConfigs that the state update is complete. The InterfaceConfigs notifies the ControlModel that data was updated. Once the ControlModel detects that the sync data was updated, it notifies the ExecutionEngine

that a Sync Event occurred. The ExecutionEngine asks the ControlModel for the next action to perform. In this scenario, the ControlModel returns a next transform action. The ExecutionEngine initiates a transform by calling the TransformEngine, which then gets the configurations and input data from the GenericInterface object. The transform configuration specifies which external interface to get data from by the transform request. E.g. if it was a DEVS➔LP transform, the next action supplied to the ExecutionEngine would have been 'DEVSLP', which it would pass to the TransformEngine.

The TransformEngine then performs the transformations and updates the states on the GenericInterface object. After the transform is complete, the ExecutionEngine then requests the next action. This continues until all actions have been completed in the control sequence. After they are completed, the ExecutionEngine will wait for the next Sync Event.

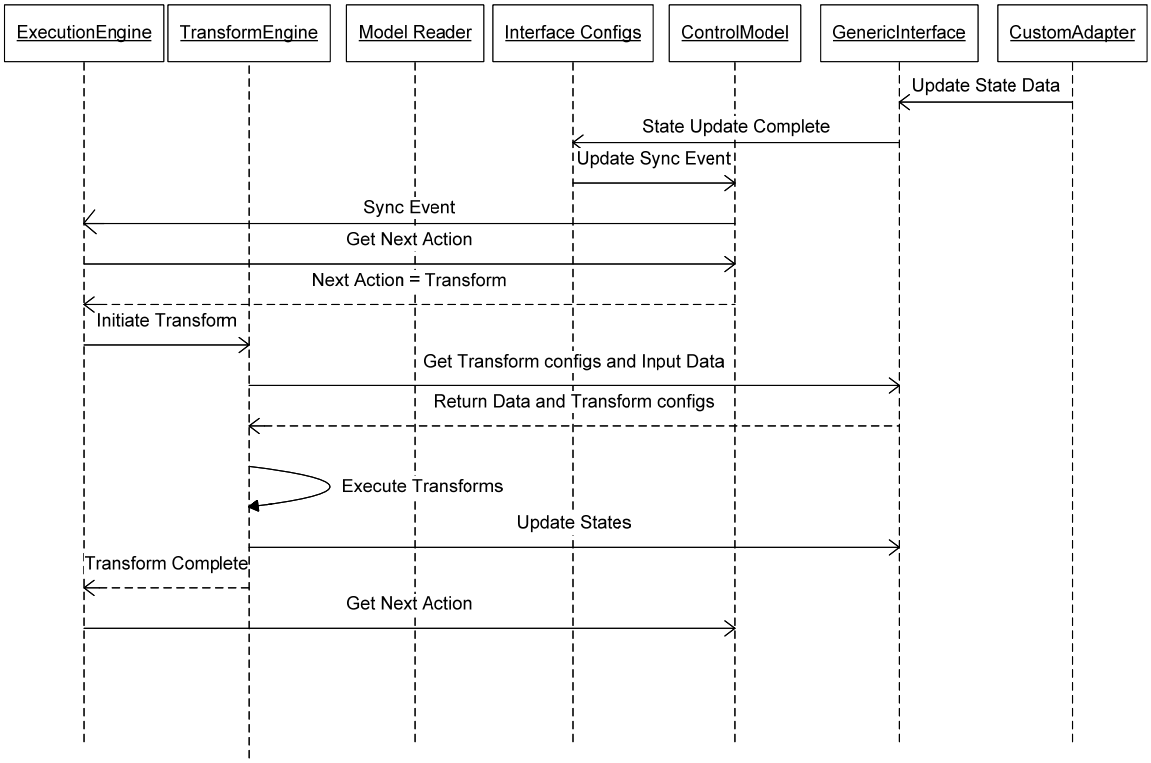


Figure 57. Use Case Sequence Diagram for Action Initiated by a Sync Event

7 CONCLUSIONS

A methodology was developed for composing multi-formalism DEVS/LP models. DEVS and LP are two very different modeling formalisms with distinct execution protocols. The DEVS formalism is well-suited for modeling and simulating system behavior. The LP formalism can quickly find optimal answers when models are abstracted to constraints and objective functions. There is a lot of research on how to solve a large variety of problems using the two different formalisms for their specialized domains. There is also existing research on the value of combining LP models with simulation models (Hung and Leachman 1996); however, this has been accomplished previously in ad-hoc ways that only support point solutions with little integration flexibility at the modeling level. The methodology enables the LP and DEVS formalisms to be composed into a semantically consistent multi-model via a KIB.

A KIB was designed and implemented to support the DEVS/LP multi-formalism modeling methodology. It was then demonstrated on a class of semiconductor manufacturing supply chain network problems. LP's were developed to optimize planning schedules and DEVS simulations were created to reproduce the semiconductor manufacturing behavior. The KIB was used to enable the composition of the two. The KIB enabled the models to be developed independently using domain experts in each area. Techniques for developing the LP's followed practices developed by specialists in planning operations. Similarly, practices developed by experts in semiconductor manufacturing simulation were used for creating the DEVS models. The mismatches in semantics between the I/O across the models, the differences in the formalism modeling

structures, and the coordination of the two at execution time were solved by the KIB model. The KIB allowed for experiments in the integration of these models by highlighting the mismatch between the models and the difficulties in synchronizing data and control between them. Items such as planning frequency and how to aggregate and disaggregate data between the LP and DEVS models were straightforward to model in the KIB. That is, it explicitly provided visibility to the integration issues and a capability to methodologically develop robust and scalable design solutions.

The software design of the KIB enabled existing implementations of LP and DEVS application development suites to be integrated. Specifically, it integrated the OPL studio and DEVSJAVA applications. Models written in these two development environments could easily be composed into multi-models by use of the KIB. This enabled the use of well-known high performance environments for modeling the LP and DEVS components. This also enabled expertise to be leveraged that already existed for creating models in these environments. The KIB enabled the correct execution and coordination of the LP and DEVS models.

The KIB was first demonstrated on a set of representative semiconductor supply chain network problems. LP's were developed to generate starts schedules into a manufacturing line; DEVS simulations of the manufacturing lines were developed to execute the instructions and simulate the resulting states. The outcomes of these experiments showed the multi-formalism environment worked correctly and also enabled the observation of manufacturing behavior under different integration scenarios.

The KIB was then extended to work with a commercial Honeywell controller application that used MPC for optimization. This type of application had not been previously used for semiconductor discrete manufacturing problems. The sequential control scheme and transformations for the DEVS/LP implementation were reused when integrating with the Honeywell applications. Real world models were developed at the actual scale seen at Intel Corporation. The simulation models matched the physical topologies and market variability, whereas the planning models matched the product mixes and demand forecasts. The KIB enabled a rapid prototyping and development of these controllers prior to production. It also enabled the discovery of data integration requirements, experimentation with different control frequencies, and experimentation with data aggregation and disaggregation algorithms. This resulted in a controller that worked within tolerances on first implementation into a multi-billion dollar production line. The KIB played an important role in accomplishing this in less than a year.

The scenarios the KIB was demonstrated on were for a particular class of semiconductor supply network problems. These experiments focused on evaluating planning and controller algorithms against simulations generating expected values seen in semiconductor manufacturing. The KIB enabled optimization and simulation models to be combined. This KIB could be applied to other domains such as simulation optimization to enable the modeling of data interactions and the coordinated execution between the simulation and optimization models. However, how the simulations, optimizations, and their interactions were modeled would be at the discretion of the expert in the field.

7.1 Future Work

The KIB methodology could be used as an enabler for integrating existing models via a composition model. This could be beneficial for large existing simulation and planning models that have many man years of investment in development and validation. Composability across these models could be supported using similar techniques to what was developed here. The KIB methodology would enable the data semantics, data timing, and coordination of control to be specified in an integration model.

Another future research area would be the application of the KIB across different domains. Areas other than semiconductor supply chain network problems could benefit from the decomposition of the domain model into multi-formalism models. Simulation Optimization was mentioned as a potential area in Chapter 2. Currently, there is ongoing related research in socio-ecological systems looking at the integration of cellular automata with agent-based models. This research requires work on how to conceptually decompose the models in the domain, identify which formalisms are best suited for the different parts, and compose the selected formalisms.

The semiconductor supply chain network problems studied in this exposition work well with serialized coordination. However, if models are computationally intensive, it could be beneficial to use a coordination scheme that supports the concurrent execution on distributed machines. Other types of coordination could involve asynchronous and non-sequential events.

There are also several areas where the KIB design could be enhanced. Visual modeling tools for the KIB XML files could provide a more intuitive modeling interface.

Automation could be created to enforce constraints to help assure consistent modeling. For example, automation could look at the available interfaces on the LP or DEVS models and then automatically detect the data structure and available I/O variables. Modelers would be constrained to specify only valid interface elements. This type of automation would require detailed knowledge of the formalisms and the interfaces of implemented models. Visualization of KIB execution could also provide insight into how the different models are interacting during execution. This could be useful for optimizing interfaces for performance or to gain insight into how the coordination across the models is actually impacting the overall behavior.

The KIB could be expanded to look at different dimensions in the Semiconductor Supply chain network domain. The multi-formalism models implemented in this work support the experimentation of tactical planning algorithms against manufacturing simulations. Tactical planning involves the creation of day to day manufacturing schedules to meet a forecasted demand. There is also a strategic planning component which looks further into the future. The KIB could be extended to coordinate multi-level planning algorithms and manufacturing simulations. Tactical and Strategic planning models work across different time horizons and generally use dissimilar data types and internal execution schemes. This would require the KIB to coordinate two different planning algorithms and maybe different formalisms with a simulation. Another research area is extending the KIB to work with multiple simulations. A possible scenario would use different simulations for manufacturing flows and market demand. The KIB would

then need to be able to support multi-models in the same formalism as well as multiple formalisms.

REFERENCES

ACIMS (2002). DEVSJAVA software. Tucson, AZ. 2002.

Buschmann, F., R. Meunier, et al. (1996). Pattern-Oriented Software Architecture Volume 1:A System of Patterns, John Wiley and Sons.

Camacho, E. F. and C. Bordons (1995). Model Predictive Control in the Process Industry. New York3-540-19924-1, Springer.

Chopra, S. and P. Meindl (2001). Chapter 5: Aggregate Planning in a Supply Chain. Supply Chain Management: Strategy, Planning, and Operation. Upper Saddle River, NJ, Prentice-Hall**:** 101-120.

CORBA (2005). CORBA Basics. Needham, MA, OMG. 2005.

Dahmann, J., M. Salisbury, et al. (1999). HLA and Beyond: Interoperability Challenges. Fall 1999 Simulation Interoperability Workshop, Orlando, FL: pp. Paper No. 99F-SIW-073.

Davis, P. K. and R. H. Anderson (2004). Improving the Composability of Department of Defense Models and Simulations. Santa Monica, CA, RAND Corporation.

de Lara, J. and H. Vangheluwe (2002). ATOM3: A tool for multi-formalism and meta-modelling. European Joint Conference on Theory and Practice of Software (ETAPS), Fundamental Approaches to Software Engineering (FASE), Grenoble, France: pp. 174-188.

Dijkstra, E. W. (1976). A Discipline of Programming. Upper Saddle River, Prentice-Hall.

Firby, R. J. and W. Fitzgerald (1999). The RAP System Language Manual**,** Version 2.0. Evanston, IL, Neodesic Corporation.

Fishwick, P. A. (1995). Simulation Model Design and Execution: Building Digital Worlds. New Jersey, Prentice Hall.

Fujimoto, R. M. (2000). Parallel and Distributed Simulation Systems. New York, NY, John Wiley and Sons, Inc.

Gartland, K., G. Godding, et al. (June 30, 2000). Scheduler/Dispatcher User Requirements. International Sematech. Technology Transfer #00063966A-TR.

Godding, G. and K. Kempf (2001). A modular, scalable approach to modeling and analysis of semiconductor manufacturing supply chains. SIMPOI/POMS IV, Sao Paulo: pp. 1000-1007.

Godding, G., H. S. Sarjoughian, et al. (2003). Semiconductor Supply Network Simulation. 2003 the Winter Simulation Conference, New Orleans, LA: pp. 1593-1601.

Godding, G., H. S. Sarjoughian, et al. (2004). Multi-Formalism Modeling Approach for Semiconductor Supply/Demand Networks. 2004 Winter Simulation Conference, Washington, D.C.: pp. 232-239.

Godding, G., H. S. Sarjoughian, et al. (2007). Application of combined discrete-event simulation and optimization models in semiconductor enterprise manufacturing systems. Proceedings of Winter Simulation Conference, Washington DC, USA: pp. 1729-1736.

Hentenryck, P. V. (1999). The OPL Optimization Programming Language. Cambridge, MA, The MIT Press.

Honeywell (2008). Profit Suite, http://hpsweb.honeywell.com/Cultures/en-US/Products/ControlApplications/AdvancedControlOptimization/ProfitSuite/default.htm.

Hopp, W. J. and M. L. Spearman (1996). Chapter 16: Aggregate and Workforce Planning. Factory Physics: Foundations of Manufacturing Management. New York, McGraw Hill**:** 502-553.

Huang, D. (2008). Composable Modeling and Distributed Simulation Framework for Discrete Supply-Chain Systems with Predictive Control. Computer Science Engineering Department, Ph.D Thesis, Arizona State University, Tempe, AZ.

Huang, D. and H. S. Sarjoughian (2006). Experiment Analysis of Hybrid Discrete Event Simulation with Model Predictive Control for Semiconductor Supply Chain Systems. Winter Simulation Conference, Monterey, CA: pp. 1863-1870.

Huang, D., H. S. Sarjoughian, et al. (2007). "Simulation of semiconductor manufacturing supply-chain systems with DEVS, MPC, and KIB." The Special Issues of IEEE Transactions on Semiconductor Manufacturing(accepted under revision).

Hung, Y.-F. and R. C. Leachman (1996). "A Production Planning Methodology for Semiconductor Manufacturing Based on Iterative Simulation and Linear Programming Calculations." IEEE Transactions on Semiconductor Manufacturing 9(2): 257-269.

IEEE (2000). IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and rules. IEEE Std 1516-2000**:** i-22.

IEEE (2001). IEEE Standard for Modeling and Simulation (M & S) High Level Architecture (HLA) - Federate Interface Specification. IEEE Std 1516.1-2000**:** i-467.

ILOG (2005). ILOG OPL Studio. ILOG Corp., CA. 2005.

ILOG (2008). ILOG OPL Studio. ILOG Corp., CA, http://www.ilog.com/products/oplstudio/. 2008.

Iwanitz, F. and J. Lange (2006). OPC - Fundamentals, Implementation and Application, Huthig Fachverlag.

Kasputis, S. and H. C. Ng (2000). Composable Simulations. 2000 Winter Simulation Conference, Orlando, FL: pp. 1577-1584.

Kempf, K. (2004). Control-Oriented Approaches To Supply Chain Management In Semiconductor Manufacturing. IEEE American Control Conference, Boston, MA: pp. 4563-4576.

Kempf, K., K. Knutson, et al. (2001). Fast Accurate Simulation of Physical Flows in Demand Networks. 2001 International Conference on Semiconductor Manufacturing Operational Modeling and Simulation, Seattle, WA: pp. 111-116.

Knutson, K., J. Fowler, et al. (2001). Modeling and Analysis of Material Flows in Complex Supply Networks. SIMPOI/POMS IV, Sao Paulo, Brazil: pp. 1123-1131.

Law, A. M. and W. D. Kelton (1999). Simulation Modeling and Analysis. New York, NY, McGraw-Hill.

Mayer, G. and H. S. Sarjoughian (2007). Complexities of simulating a hybrid agent-landscape model using multi-formalism composability. Agent-Directed Simulation, Spring Simulation Multiconference, Norfolk VA, USA: pp. 161-168.

Mayer, G., H. S. Sarjoughian, et al. (2006). Simulation modeling for human community and agricultural landuse. Agent-Directed Simulation, Spring Simulation Multiconference, Huntsville, AL: pp. 65-72.

Moré, J. and S. Wright (1993). Optimization Software Guide. Philadelphia, PA, Society for Industrial and Applied Mathematics (SIAM).

Mosterman, P. J. and H. Vangheluwe (2004). "Computer Automated Multi-Paradigm Modeling: An Introduction." Simulation 80(9): 433-450.

Müller, J. P. (1996). The Design of Intelligent Agents: A Layered Approach, Springer.

OAGIS (2008). The Open Applications Group Integration Specification, http://www.ibm.com/developerworks/xml/library/x-oagis/.

OPC-Foundation (2008). OPC Standards, http://www.opcfoundation.org. 2008.

Platt, D. S. (2003). Introducing Microsoft .NET. Redmond, WA, Microsoft Press.

Prähofer, H. (1991). Systems Theoretic Foundations for Combined Discrete Continuous System Simulation. Institute of Systems Science, Department of Systems Theory and Information Engineering. Linz, Austria, Johannes Kepler University.

Rardin, R. L. (2000). Optimization in Operations Research, Prentice Hall, Inc. Upper Saddle River, NJ 07458.

Sarjoughian, H. S. (2006). Model Composability. Proceedings of the Winter Simulation Conference, Monterey CA, USA: pp. 149-158.

Sarjoughian, H. S. and D. Huang (2005). A Multi-Formalism Modeling Composability Framework: Agent and Discrete-Event Models. Proceedings of the 2005 Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications, Montreal, Canada: pp. 249-256.

Sarjoughian, H. S., D. Huang, et al. (2005). Hybrid discrete event simulation with model predictive control for semiconductor supply-chain manufacturing. Proceedings of Winter Simulation Conference, Orlando FL, USA: pp. 256-266.

Sarjoughian, H. S. and J. Plummer (2002). Design and Implementation of a Bridge between RAP and DEVS. Tempe, AZ, Computer Science and Engineering, Arizona State University**:** 1-38.

Sarjoughian, H. S. and B. P. Zeigler (2000). "DEVS and HLA: Complementary Paradigms for Modeling and Simulation?" Transactions of the Society for Modeling and Simulation International 17(4): 187-197.

Singh, R. K., H. S. Sarjoughian, et al. (2004). Design of Scalable Simulation Models for Semiconductor Manufacturing Processes. Summer Computer Simulation Conference, San Jose, CA: pp. 235-240.

Vangheluwe, H. and J. de Lara (2002). Meta-Models are Models Too. 2002 Winter Simulation Conference, San Diego, CA: pp. 597-605.

Vangheluwe, H. and J. d. Lara (2004). Computer Automated Multi-Paradigm Modelling for Analysis and Design of Traffic Networks. Proceedings of 2004 Winter Simulation Conference, Washington, D.C.: pp. 249-258.

Wang, W. (2006). Model Predictive Control Strategies for Supply Chain Management in Semiconductor Manufacturing. Department of Chemical and Materials Engineering, Ph.D Thesis, Arizona State University, Tempe, AZ.

Wang, W., D. Rivera, et al. (2007). "Model predictive control strategies for supply chain management in semiconductor manufacturing." International Journal of Production Economics 107(1): 56-77.

Wu, N. and R. Coppins (1981). Linear Programming and Extensions. New York, NY, McGraw-Hill.

Wymore, A. W. (1993). Model-based Systems Engineering: An Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design. Boca Raton, CRC Press.

Zeigler, B. P. and T. I. Oren (1986). Multifacetted, Multiparadigm Modelling Perspectives: Tools for the 90's. 1986 Winter Simulation Conference, Washington, D.C.: pp. 708-712.

Zeigler, B. P., H. Praehofer, et al. (2000). Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. New York, NY, Academic Press.

Zeigler, B. P. and H. S. Sarjoughian (1997). Object-Oriented DEVS. 11th SPIE, Orlando, Florida: pp. 100-111.