MULTI-FORMALISM MODELING APPROACH FOR SEMICONDUCTOR SUPPLY/DEMAND NETWORKS

Gary W. Godding

Component Automation Systems
Intel Corporation
5000 W. Chandler Blvd – MS CH3-68
Chandler, Arizona, 85226, U.S.A.

Hessam S. Sarjoughian

Arizona Center for Integrative Modeling & Simulation
Computer Science & Engineering Dept.
Arizona State University
Tempe, Arizona,85281-8809, U.S.A.

Karl G. Kempf

Decision Technologies
Intel Corporation
5000 W. Chandler Blvd – MS CH3-10
Chandler, Arizona, 85226, U.S.A.

ABSTRACT

Building computational models of real world systems usually requires the interaction of decision modules and simulation modules. Given different models and algorithms, the major hurdles in building a principled and robust system are model composibility and algorithm interoperability. We describe an approach to the composibility problem including initial results. This exposition is given in the context of Linear Programming as the decision technique and Discrete Event Simulation as the simulation technique, both applied to the design and operation of semiconductor supply/demand networks.

1 INTRODUCTION

A recurring theme in developing computational models of the real world is the necessity of including both physical and logical processes. The system being studied presumably has some physical behavior that develops over time, but requires direction from some logical process to select among possible actions to achieve a system performance goal (see Figure 1).

Although the algorithms, models, and data usually overlap to some degree, decision making and decision execution are generally very different computationally. There are at least two extreme options that can be employed to address these differences.

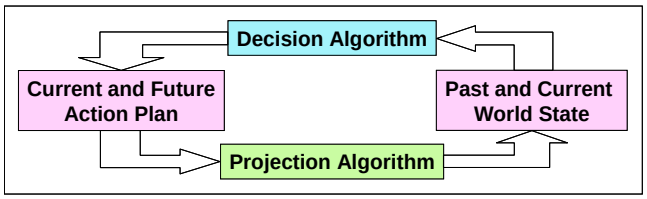

Figure 1: The Abstract Computation Problem

On one hand, a system design can force the required functionality into a single monolithic model. However the result is often very difficult to develop, validate, verify, use, and maintain because of the confounding of the decision making and decision execution processes.

On the other hand, the models and algorithms for decision making and decision execution can be designed and implemented separately and then used together in an integrated fashion. One major difficulty with this approach lies in the details of integrating the operation of the decision making module and the decision execution module. While there exists a rich and extensive literature on computational models for both decision making and decision execution, most examples where these models are used in combination rely on ad hoc approaches to integration of the specific implementations of interest. From a more general and formal perspective, the computational issues with robust integration are model composibility (our particular focus here) and module interoperability (see Figure 2). Composability is concerned with how to create different models that are

Accepted for Publication

semantically consistent. Interoperability is focused on the software used to support communication and synchronization at runtime.

In addition to composibility and interoperability, another domain-independent issue inherent in this approach is the concept of "netting". Execution of the plan will rarely (if ever) produce the state of the stochastic world expected by the planning module. This "netting" will have to reconcile the expected and actual states of the world for the decision module, and while this problem is universal, its solution is highly dependent on the details of the domain. Composibility is similar in that it is universal in this class of problems, but different in that it has both domain independent and dependant facets as we will show here.

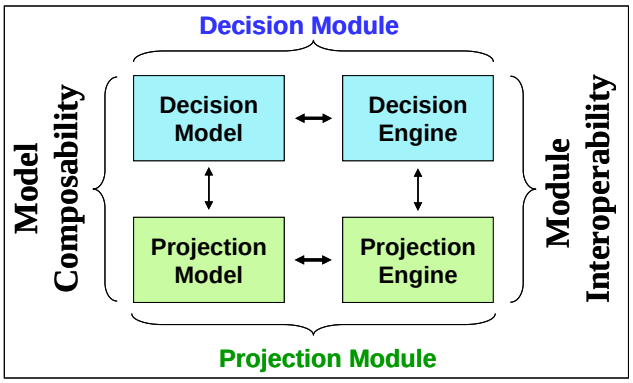

Figure 2: Composibility and Interoperability

To explore these issues without loss of generality, we have selected as an application domain that of managing a semiconductor supply/demand network (SSDN) (Figure 3). The decision module is charged with building a plan for current and future activities including starting materials into factories, shipping materials between locations, and moving materials into and out of warehouses based on a profitability goal. The execution module is intended to manage the (virtual) current and future application of the plan to the 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 as input the state of the world resulting from the past application of the previous plan, and produces the plan for the future. The execution module needs the plan of action to project forward in time to produce the future states of the world. Clearly these are quite different algorithms requiring different models.

To ground our exposition of composability, interoperability, and netting, we have selected, without loss of generality, specific decision and execution approaches for experimentation. From the range of possible mathematical and heuristic decision algorithms, Linear Programming (LP) has been selected. Discrete Event Simulation (DES) has been selected from the range of possible continuous and discrete real world execution projection approaches. To reiterate, the focus of our work is not LPs and DESs for SSDN, but rather the general and specific aspects of the composability of their models in this application domain. We have described our work in the domains of LP and DES in other publications (Kempf et al. 2001, Kempf 2004) and the references they contain.

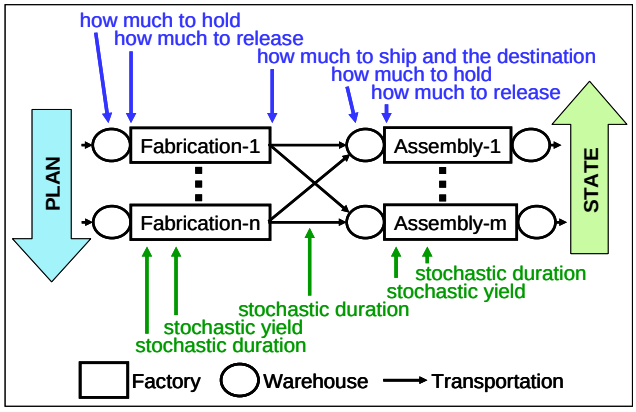

Figure 3: An Example from Manufacturing in a SSDN

The practical use of the test-bed resulting from robust model composibility (and attendant module interoperability and netting) is the improved design and operation of the SSDN example. This type of economic system can generate, in the case of Intel Corporation, 30B$ in annual revenue. Given the complexity of operating such a network, the test bed partially described here is invaluable. The ability to refine decision processes as well as the number, location, topology, and properties of the physical entities without the overhead of continuously reconfiguring interfaces is a major improvement over existing methods.

2 RELATED WORK

2.1 Model Composability

With model composability, a large problem can be partitioned and appropriate modeling approaches used to model the overall system. An approach to model composability, therefore, must ensure semantic integrity of the parts and their combinations. Composite models with appropriate semantic underpinning can then be ensured to correctly execute and interoperate. The lack of an approach and its associated methodology to support simulation model composability is a well recognized and documented concern (e.g. Davis and Anderson 2003; Fishwick 1995). For SSDN, it is possible to integrate LP and DES models using custom-built bridges through a combination of programming languages and communication protocols (e.g.,

Godding and Kempf 2001). Similarly, Agent and DES models can be composed by ensuring correct ordering of events and valid types of message exchanges (Sarjoughian and Plummer 2002). These kinds of approaches result in *pseudo composability* – achieving model integration through interoperability between a simulation engine and an optimization engine or an interpreter.

2.2 Multi-formalism modeling

Constructing models using two or more modeling approaches requires a multi-formalism approach where each part of a composite model is described in a formalism that is most suitable. Within a system theoretic framework, much progress has been made on formal theories and methodologies of how to construct discrete and continuous simulation models that have well-defined properties such as conversion of continuous data to discrete data and correctness of synchronization between continuous and discrete simulation protocols (Zeigler, Praehofer, and Kim 2000). Other approaches such as Ptolemy II (Plotomy 2004) offer capabilities to compose continuous models of physical systems with discrete models of controllers that are responsible for managing behavior of the former. Unfortunately, neither of these approaches or other similar ones are suitable for the SSDN. For example, an agent based approach such as (Swaminathan, Smith, and Sadeh 1998) does not address how to compose complex decision models with large stochastic data flows. For these formalisms to support development of complex decision models and thus SSDN, their underlying frameworks need to be extended with new concepts and methods.

Due to the unavailability of a suitable framework for modeling SSDN, decision and process modules are generally combined using ad-hoc low-level programming techniques. Optimization modules have been integrated with simulation modules for specific cases (e.g. Hung and Leachman 1996) resulting in ad-hoc composability. Customized combination of models, however, are not based on sound principles that can support well-defined relationships among the components of a composed model.

3 BACKGROUND

3.1 Linear Programming Formalism

The purpose of an LP is to find the best answer when many exist. Linear programming can be classified as a selection algorithm where an answer is selected 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, a set of decision variables,* and *a set of constants* (Wu and Coppins 1981). The LP relationships including constraints and objective functions must all be linear.

LPs have been applied successfully to a wide range of planning applications (Hopp and Spearman 1996, Chopra and Meindl 2001) when each problem can be described as a set of linear constraints and a cost function. Linear programming, however, is not suitable for describing dynamic (time-driven) behavior of systems. Instead, constraints and objective function are useful for formulating the inputs to an optimization problem – i.e., a problem that searches for the best assignment of values assigned to decision variables to achieve the maximum (minimum) objective function value.

3.2 DEVS Formalism

The DEVS (Discrete Event System Specification) is a modeling formalism for describing (discrete and continuous) dynamical systems as discrete-event models. Complex models can be hierarchically constructed from *atomic* and *coupled* models that communicate with other models.

An atomic model specifies *input variables* and *ports*, *output variables* and *ports*, *state variables*, *internal* and *external state transitions*, *confluent transition function*, *output* and *time advance functions* (Zeigler, Praehofer, and Kim, 2000). An atomic model is stand-alone component capable of autonomous and reactive behavior with well-defined concepts of causality, timing, handling multiple inputs and generating multiple outputs.

A coupled model description specifies its constituents (*atomic* and *coupled models*) and their interactions via *ports* and *couplings*. 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 2003). Coupled models, similar to atomic models, have sound causality, concurrency, and timing properties that are supported by various simulation protocols in either distributed or stand-alone computational settings.

While this formalism may be used for optimization, its conceptual framework is not well suited to support linear programming or other optimization modeling approaches where general mathematical equations specify constraints among decision variables (Godding, Sarjoughian, and Kempf, 2003). Instead, DEVS and more generally systems theory is concerned with describing the structure and behavior of a system and simulating it for some period of time.

3.3 SSDN modeling using LP and DEVS

Composing multi-formalism models described in linear programming and discrete-event formalisms requires a set of well-defined concepts that account distinctly for both *composition* and *interoperability*. Many benefits can be foreseen with such an approach. For example, it can enable development of a methodology for describing models composed of decision models and process models without requiring modelers to delve into the details of a given simulation engine and a given optimization solver. Evaluation of decision policies against physical process flows, or evaluation of the impact on decision policies when the physical system is changed would be possible without requiring custom software development. In addition, if a methodology is available, it would pave the way to enable the development of tools that allow modelers of different expertise (such as mathematicians and simulation modelers) to create different, yet semantically consistent model components of the composite model that could then seamlessly execute and interoperate.

4 APPROACH

Two significant issues must be addressed to support composition and execution of SSDN multi-formalism models. One is how to support the complex data transformations required between the models. The second is how to enforce semantically correct composition of the different models.

Our approach to manage the data transformations is a broker between the models. The broker is referred to as a *Knowledge Interchange Broker* (KIB). This broker must provide capabilities beyond those offered by the simulation/execution (e.g., ILOG CPLEX and DEVS simulation protocol) and middleware (e.g., CORBA) layers.

Our approach to enable the enforcement of the composition of semantically correct models is to introduce the concept of a common model. This model would provide a common vocabulary to impose constraints on the decision and physical models.

4.1 KIB

The proposed multi-formalism composability approach for SSDN focuses on combining two classes of models – optimization and simulation models expressed in LP and DEVS formalisms, respectively. This approach is based on a three-layer worldview where the *modeling*, *simulation/execution*, and *middleware* layers are separated from one another (Sarjoughian and Cellier 2001). Within the modeling layer, decision policies can be modeled as linear programs and discrete event simulation can be used for modeling process flows. A conceptual view of the modeling layer is depicted in Figure 4.

The KIB must provide its own modeling constructs and execution capabilities to support composition of decision models and process models. The KIB can be viewed from three complementary perspectives. First, it needs to provide primitive capabilities for translating (or mapping) of data from LP to DEVS and vice versa. This requires not only syntactical (structural) translation between LP and DEVS but also imposing semantically sound behavior of the composed model and its constituents.

Second, given the decision and process behavior of the SSDN, it is necessary to describe how the SSDN application domain affects or constrains the KIB. For example, the decision and process models illustrated in Figure 4 need to use the same names for the messages and have a common model of what data is shared, exchanged, and the transformations needed for a composed model.

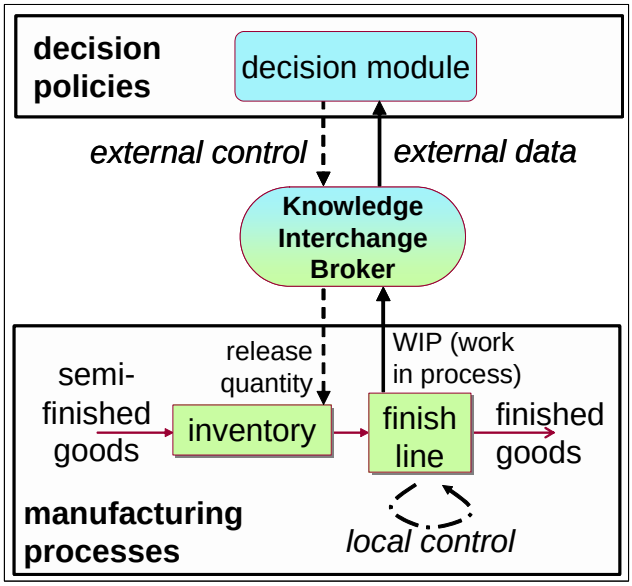

Figure 4: Three Layer World View with KIB

Third, the specification of the KIB must account for timing between the LP and DEVS models. This includes enforcement of (time-based) causal relationships supporting their concurrent execution, and their synchronized communications.

Figure 5 illustrates some simple primitive capabilities of the KIB. Part (a) is a scenario showing the generic aggregation and disaggregation of the port ID for messages between LP and DEVS. LP models do not have ports while DEVS models do. Part (b) is a domain specific scenario exemplifying the passing of messages and data transforma-

tion between the SSDN decision policy and the associated manufacturing process flow it is controlling.

Netting is a special type of transformation needed to correct discrepancies between the plan and what actually happened. One simple strategy is to divide the plan into smaller time periods and add or subtract the differences from the current time to the next period. For example, the netting algorithm could divide a weekly plan by seven and at the end of each day, add or subtract the daily discrepancies to the plan for the next day.

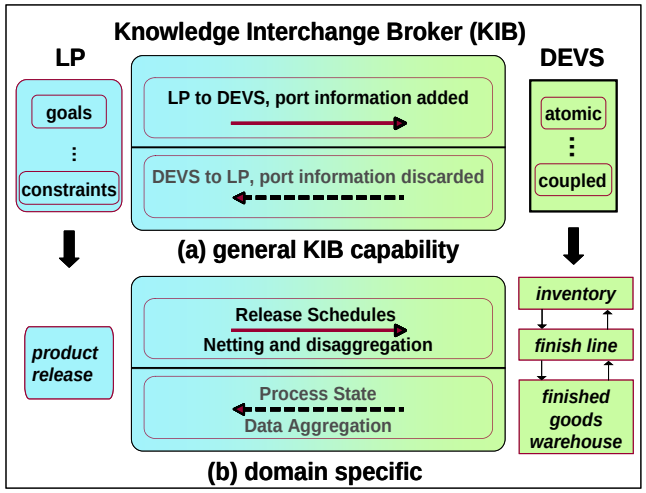

Figure 5: Capabilities of KIB

4.1.1 LP and DEVS I/O Composability via KIB

The mathematical representations for LP and DEVS modeling formalisms are shown in Table 1. Each formalism provides a set of general modeling constructs for describing a class of behaviors – e.g., dynamical discrete process models with DEVS. To form a composable modeling approach, it is necessary to formulate appropriate logic for the KIB to support the transformation of data/control between LP and DEVS, the addition of missing data/control, and the deletion of superfluous data/control that is irrelevant. Furthermore, it is central for the KIB to support causality, concurrency, and synchronization as the keys to correct behavior specification and combined execution of LP and DEVS models.

Since these two formalisms are distinct, the KIB needs to translate the inputs and outputs from one formalism to the outputs and inputs of the other, respectively. This is appropriate since each formalism can be viewed as a system with well-defined input and output interface. The possible inputs to the LP formalism from DEVS would be the cost vectors, the constraint matrix, and the constraint values. (i.e. c, A, and b shown in Table 1). Some or all of these could be DEVS inputs depending on the problem being modeled. The outputs of the LP formalism to DEVS would be *x*, the decision variables. For the DEVS formalism, the inputs from the LP would be a subset of X – i.e., the set of values available on the input ports of atomic and coupled models. Similarly, the outputs of the DEVS to the LP would be a subset of Y – i.e., the set of output values from the atomic and coupled models.

LP and DEVS models may have inputs and outputs that are independent of one another. For example, a process model can generate outputs that are of no interest to the decision model, yet of importance to observe the process flows. The process model may receive inputs that are not sent by the decision model. These types of interactions between LP and DEVS models provide a basis for formulating an appropriate logic for the KIB such that it can support transforming data structures and ensuring causality, concurrency, and synchronization between the models.

LP Formalism	DEVS Formalism
min{cx: Ax=b, x≥0) where: $x \in \Re^n$ $c \in \Re^n$ $b \in \Re^m$ $A \in \Re^{m \times n}$ • *c* is a vector of cost variables • *x* is a vector of decision variables (unknowns) • *b* is a vector of constants • *A* is the constraint matrix	<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

Table 1: Standard Forms for LP and DEVS

4.2 Problem Domain Common Model

To formulate decision models and process models that are semantically correct, they need to share some common data of the problem they are representing. A methodology is necessary to characterize ways in which various pieces of the common model can be mapped onto the optimization and process models. Additionally, the common model is important in the development of the KIB – defining translations (interactions) between the optimization and simulation models. For example, KIB needs to have a well defined knowledge representation in order to convert a release command from the decision layer into an instruction for the correct entity in the process model. As mentioned

earlier, these translations may involve complex calculations such as netting.

Figure 6 shows a conceptual representation of how the common model relates to the other models. The common model constrains the valid names, configurations, and translations that can be modeled in the LP, KIB, and DEVS. The common model needs to specify the structure of the problem, such as the SSDN topology, and also the valid relations between the models. For example, the common model may specify that some specific data values need to be aggregated from a daily resolution to a weekly resolution. The KIB transformation model would use this to specify the detailed aggregation algorithm to use on the data variables given by the common model.

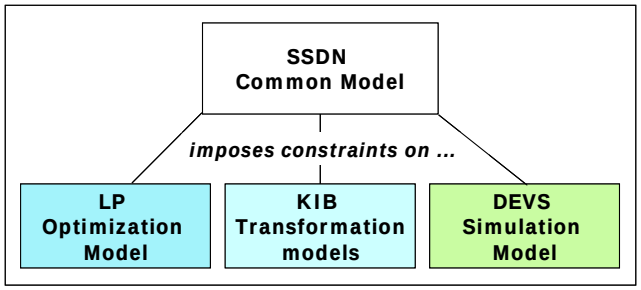

Figure 6: Common Model Relationships

A small example of the type of common structural data needed to represent the topology and product flow of a SSDN is shown in Figure 7. This common model is a mapping of the BOM (Bill of Materials) to a process flow topology. It specifies facility names, product names, and the relation of product routes to facility topologies. Material can be shipped between facilities via transportation, can be held within a manufacturing line and considered as work in progress (WIP), or can be sitting in a warehouse.

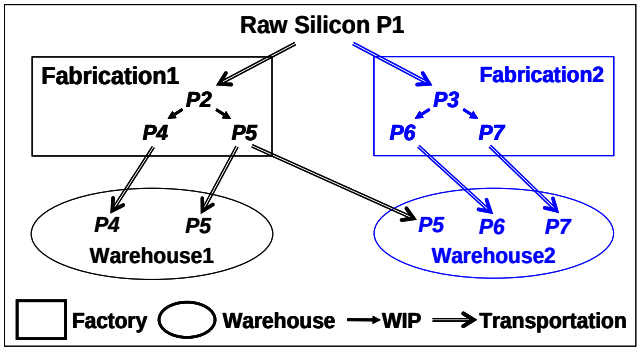

Figure 7: SSDN Common Model

This model corresponds to a segment of the topology shown in Figure 3. The possible routes through the network have been modeled for products P1-P7. For example, raw silicon (P1) can be sent into plant 1 or 2 and fabricated into one of the P4-P7 products. If the material is made into P5, it can be shipped to either warehouse 1 or 2. The knowledge of which products can be made in what facilities and where it can be routed must be the same in both models.

4.3 Behavioral Composability between LP and DEVS models

The approach described in Sections 4.1 and 4.2 must support the passage of time since simulation models are time driven. The necessity of modeling time has important consequences as alluded to earlier. In particular, causality, concurrency and synchronization need to be accounted for with respect to time. For example, causality not only needs to be consistent in the decision and process model specifications, but also the concurrent execution of the models and synchronized data exchange between them need to be consistent with the causality specification of the KIB. Some of the details of the causality, concurrency, and synchronization are presented next.

4.3.1 Causality

The KIB will need to enforce the correct causality for message passing between the decision and process models. For example, in the scenarios we are modeling, it is not possible in the real world for the planning organization to have knowledge of the future state of the world, so it would not be correct for the decision algorithm to have any future projected state from the process models. Causality is already accounted for within the DEVS framework. However, LP models are not time dependent and their behaviors do not evolve over time. Therefore, support for causality of the composed models may be placed in the KIB. That is, the KIB will support causality between LP and DEVS models while preserving the causality of the DEVS models.

4.3.2 Concurrency

The KIB will need to manage concurrency since the execution and simulation modules must be under the control of two different processes. One of the most straightforward methods to support this type of concurrency is via the use of synchronization.

Aside from synchronization of processes, there is also the potential of “logical concurrency”. That is both linear and process models consume logical time. Since both models can be expressed in terms of time, they can be exe-

cuting concurrently in logical or physical time. For example, assuming it takes two days (logical time) to generate a plan, the process model may execute concurrently with the decision model for two days (logical time) before receiving a plan.

4.3.3 Synchronization

In addition to causality and concurrency, the KIB also needs to support synchronization. A basic form of synchronization is to ensure that both input/output exchanges between the process and decision models are logically correct – i.e., the order of events produced and consumed by process and decision models are maintained. More advanced forms of synchronization can account for timing and therefore support causality using both ordering and timing of events given concurrent processes.

5 PROTOTYPE KIB DESIGN

The design and implementation of the KIB was based on OPL Studio (ILOG 2004) and DEVSJAVA (ACIMS 2003). The KIB was developed using a combination of software objects specified in DEVSJAVA and JAVA™. The prototype implementation has been created and tested with the simple models shown in Figure 8. The DEVS model consists of an inventory, a finish line, and a finished goods inventory. The LP calculates release schedules using Inventory, WIP, and Demand models. The inventory and WIP data is supplied from the DEVS model. The demand model is used only by the LP.

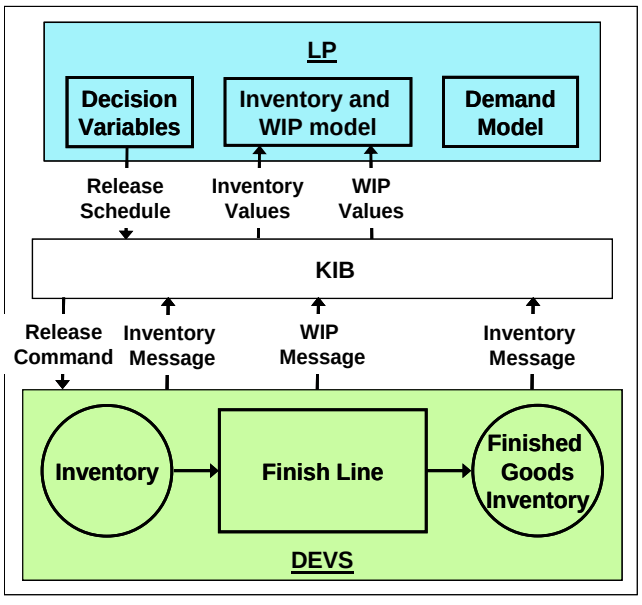

Figure 8: SSDN First Prototype

The SSDN model execution sequence starts with the DEVS model initializing the KIB. When the KIB receives the initialization call, it initiates the simulation and sends the appropriate data to the LP followed by an LP solve request. The results of the initial solve are sent to the simulation prior to the current time period being completed. This sequence continues until a desired number of iterations have been executed.

The implementation of the KIB requires components for a data model, synchronization, and mappings of outputs to inputs between LP and DEVS and vice versa. The data model is necessary for maintaining a representation of the SSDN common model and the relationships of the LP and DEVSJAVA models to the SSDN. The synchronization logic ensures correct ordering of the DEVS messages sent and received to KIB and also used to request an LP solve at the proper time instances during the SSDN execution. The mapping logic performs the necessary data transformations between the LP and DEVS models.

The data model consists of three different sets of data. The first set of data is the SSDN common model representation. The second set of data is the DEVSJAVA specific data such as port names to use for each of the atomic models. The third set of data is the OPL specific interface data. The prototype implementation used a Hashtable to hold this data. Each entry of the Hashtable represents a part of the SSDN model such as Finish Line. For each of the SSDN entries, additional entries were linked for DEVS and LP specific data. The input and output specifications for DEVS and LP were included so the KIB logic could do the required data translation. No complex translations of the data such as disaggregation were required for this implementation. In the future, when these types transformations are added, a fourth set of data tied to the KIB will need to be added. The DEVS portion of the Hashtable is populated automatically. The DEVS model reports its structure in a set of initialization messages. The LP portion is populated manually, although it is important to support automatic configuration of the data model.

The synchronization between the models is accomplished using a clock in the KIB. A clock event occurs at the end of each time period and specifies what the value of the next time is. The clock is implemented as an atomic DEVSJAVA model and is synchronized with the SSDN process models. The KIB uses the clock to timestamp messages to and from the LP and to synchronize the execution of the LP and DEVS models.

The mapping logic translates data representations from DEVSJAVA event messages to OPL data structures. This includes the addition and deletion of port information for DEVS messages, and the translation of DEVS messages sets into OPL arrays and variables.

6 FUTURE WORK

The research described in this paper provides a basis for composing LP and DES models. Follow up work falls into four categories – develop a specification for the SSDN Common Model, support additional capabilities for the Knowledge Interchange Broker, apply the composability approach in real-world settings, and devise a methodology to assist modelers to develop composable models. Research in each of these categories is expected to extend the basic capabilities of the proposed model composability approach. For example, since the common model is a meta-model imposing constraints on the constituent models, its specification needs to support representation of the product routing relations and data transformations between the models. Similarly, the kinds of transformations within the KIB need to be extended. In particular, it is important for the KIB to support a set of generic as well as domain-specific modeling constructs for input and output transformations – e.g., supporting netting. Furthermore, it is important to show how the Common Model and the KIB can be used systematically for developing decision and process models. It is also useful to provide a roadmap and guidelines to help modelers develop composable SSDN models.

7 CONCLUSIONS

A multi-formalism modeling approach enabling composition of models described as decision and projection algorithms is presented. The proposed approach is based on the separation of decision and physical process models with the aid of a Common Model and Knowledge Interchange Broker. The roles of the KIB for LP and DEVS applied to a SSDN problem were described and demonstrated using a prototype to offer a sound basis for model composability in conjunction with simulation/execution interoperability.

One of the key offerings of the multi-formalism modeling approach is its support for describing models in widely used LP and DES modeling paradigms. A complementary advantage is the separation between model composability and module interoperability. The ability to use multiple modeling formalisms and execute each separately can afford important benefits such as relying on soundness of the models and the correctness of their distributed execution. Moreover, modelers can focus their attention on model specifications since execution (i.e., simulation and solver) interoperability would already be supported as part of the multi-formalism approach.

ACKNOWLEDGMENTS

This research is supported by the Intel Research Council, Chandler, Arizona. The authors would like to express their appreciation to Bin Xie of the Computer Science & Engineering Department at Arizona State University for assistance in the development of the SSDN model.

8 REFERENCES

ACIMS. 2003. DEVSJAVA Software. Available online via <http://www.acims.arizona.edu/SOFTWARE> [accessed August 19, 2004].

Chopra, S. and P. Meindl. 2001. *Supply Chain Management: Strategy, Planning, and Operation*, Upper Saddle River, NJ: Prentice-Hall, part 2 (Planning Demand and Supply in a Supply Chain).

Davis, P. K. and R. H. Anderson. 2003. *Improving the Composability of Department of Defense Models and Simulations*, RAND.

Fishwick, P. A. 1995. *Simulation Model Design and Execution: Building Digital Worlds*, Prentice Hall.

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

Godding, G. W., H. S. Sarjoughian, and K.G. Kempf. 2003. Semiconductor Supply Network Simulation, In *Proceedings of 2003 the Winter Simulation Conference*, New Orleans, LA, pp. 1593-1601.

Hopp, W. J. and M. L. Spearman. 1996. Aggregate and Workforce Planning. In *Factory Physics: Foundations of Manufacturing Management*, 535-581. New York: McGraw Hill.

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.

Kempf, K. G., K. Knutson, J. Fowler, D. Armbruster, P. Babu, and B. Duarte. 2001. Fast Accurate Simulation of Physical Flows in Demand Networks, In *Proceedings of Semiconductor Manufacturing Operational Modeling and Simulation Symposium*, Tempe, AZ, pp. 111-116.

Kempf, K. G. 2004. Control-Oriented Approaches to Supply Chain Management in Semiconductor Manufacturing, In *Proceedings IEEE American Control Conference*, Boston, MA, to appear.

ILOG. 2004. ILOG OPL Studio. Available online via http://www.ilog.com/products/oplstudio/ [accessed July 14, 2004].

Ptolemy. 2004. Ptolemy project; Heterogeneous Modeling and Design. Available online via <http://ptolemy.eecs.berkeley.edu> [accessed July 14, 2004].

Sarjoughian, H. S. and F. E. Cellier. 2001. Toward a Unified Framework for Simulation-Based Acquisition, in *Discrete Event Modeling and Simulation Technolo-*

gies: A Tapestry of Systems and AI-Based Theories and Methodologies, ed. H.S. Sarjoughian and F.E. Cellier, pp. 1-14, Springer-Verlag.

Sarjoughian, H. S. and J. Plummer. 2002. Design and Implementation of a Bridge between RAP and DEVS, Internal Report, Computer Science and Engineering Department, Arizona State University, Tempe, AZ, pp. 1-38.

Swaminathan, J. M., S. F. Smith, and N. M. Sadeh. 1998. Modeling Supply Chain Dynamics: A Multiagent Approach, *Decision Sciences*, 29(3) : 607-632.

Wu, N. and R. Coppins. 1981. *Linear Programming and Extensions*, McGraw-Hill.

Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. *Theory of Modeling and Simulation*, Second Edition, Academic Press.

AUTHOR BIOGRAPHIES

GARY W. GODDING is a Software Engineer at Intel Corporation and a PhD candidate in the Computer Science and Engineering department at Arizona State University. His research includes modeling and simulation of supply networks, software architecture, and artificial intelligence. He can be contacted by e-mail at <gary.godding@intel.com>.

HESSAM S. SARJOUGHIAN is Assistant Professor of Computer Science and Engineering at Arizona State University, Tempe. His research includes hybrid simulation modeling and intelligent agents, collaborative modeling, distributed co-design, and software architecture. His industrial experience has been with Honeywell and IBM. For more information visit <http://www.eas.asu.edu/~hsarjou/index.htm >.

KARL G. KEMPF is Director of Decision Technologies at Intel Corporation and an Adjunct Professor at Arizona State University. His research interests span the optimization of manufacturing and logistics planning and execution in semiconductor supply chains including various forms of supply chain simulation. He can be contacted by e-mail at <karl.g.kempf@intel.com>.