Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

TOWARD ON-DEMAND WAFER FAB SIMULATION USING FORMAL STRUCTURE & BEHAVIOR MODELS

Edward Huang
Ky Sang Kwon
Leon McGinnis

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0205 USA

ABSTRACT

Contemporary factories in capital intensive industries such as semiconductor manufacturing are very complex, with many sources of risk. The highly competitive and global business environment forces companies to analyze, design, and continuously re-design factories with distributed multi-disciplinary teams. Traditional factory design approaches using spreadsheets and stand-alone simulations cannot adequately cope with the resulting time, cost, and risk requirements. In this paper, we address the opportunity to support fab design teams by providing on-demand simulation. The method for achieving this combines formal fab descriptive models with a process for generating fab analysis models from relatively standard sources of fab data.

1 INTRODUCTION

The contemporary 300mm wafer fab may contain over a thousand process tools, several thousand foups, or front-opening unified pods, hundreds of material transporters, and thousands of foup storage locations**.** It must cope with risks associated with uncertain future product design and production requirements, uncertain future technology availability and performance, and the risks inherent in very large-scale automated systems. The investment cost for even one such wafer fab is so large that poor design decision making may threaten the owning firm's existence. With technology in constant flux, and with pressure to reduce time to first good wafer out, there is enormous pressure on the fab design team, which may span a number of locations, corporate functions, and professional disciplines. Further complicating the design process are differences within the fab design team in terms of culture and language, and the technical tools, models, and metrics used. Once the fab is in operation, it will undergo almost constant re-design to accommodate new tools and technologies.

In this environment, traditional ad-hoc approaches to fab simulation, using spreadsheets, documents, and CAD layout drawings to convey design intent to simulation experts is inadequate for unambiguously communicating this complexity. What is needed is a new generation of fab simulation tools that uses formal models of the fab and its essential systems to enable unambiguous communication of design intent, and also supports on-demand full fab simulation.

One reason this is a realistic goal is that contemporary 300mm wafer fabs are approaching full automation, at least in terms of production activities (see, Hunter and Humphreys 2003). Fully automated production, unlike production involving "touch labor," is constrained by the software systems controlling the automation. Thus, at least in terms of how individual devices respond to specific events, fully automated systems are more predictable. In addition, their behavior already is "formally" described, by the control software. As a consequence, despite the complexity of the 300mm wafer fab, it is an aggregation of objects with relatively simple behavior and interaction logic.

A second reason it is realistic to aspire to a new generation of fab simulation tools is the tremendous advancements in computational infrastructure (CI) made over the past decade. High bandwidth communication enables real-time collaboration between remotely located computing resources. Inexpensive grid computing supports on-demand simulation studies. Perhaps most important, the emergence of new modeling tools, particularly SysML (OMG 2008), provides the expressiveness needed for modeling complex systems in a formal but relatively easy to use language.

In this paper, we propose an approach to developing on-demand fab simulation tools based on formal models of two distinct types: *descriptive* models and *analytic* models. A *modeling framework* will be described which supports the description of both fab resources and their behaviors using a formal modeling language. An *application framework* will be described, in which knowledge about both the application domain and analysis models is captured in appropriate libraries and (re)used. A key element of this framework is on-demand automated translation from instance descriptive models to instance simulation models.

This paper is organized as follows. Traditional approaches to analyzing wafer fabs is discussed in section 2. In section 3, we discuss issues arising from traditional approaches to fab simulation, and discuss requirements to address them. In section 4, we propose a new wafer fab modeling and analysis framework, and describe implementation examples proving the concept in section 5. We end with conclusions and suggestions for future work.

2 LITERATURE REVIEW

There are two categories of published papers related to fab modeling: those directly addressing formal models of fabs, and those primarily focused on fab design.

2.1 Formal Fab Models

Arief and Speirs (1999) propose automatic generation of discrete event simulation models from formal system models created with UML. They defined the language SML, Simulation Modeling Language, to map from the UML system model into the target simulation language, which was C++ SIM (Arjuna Team 1994), although the actual translation was performed manually.

Whittle (2000) proposed using UML as the formal modeling language. UML is a powerful modeling language but additional semantic requirements must be specified to make it useful for modeling specific systems. The author presented an overview of some of the attempts to specify the application-specific semantics of UML, but no implementation or proof of concept was provided.

Saldhana et al (2001) also tried to use UML as the formal language for modeling wafer fabs. They proposed using Petri nets as the interface between the formal system model and the corresponding simulation model. They developed Object Net Models from UML state-chart diagrams. A Colored Petri Net was generated with components defined from the Object Net models and connections between components defined using the information in UML collaboration diagrams. Finally, the Colored Petri Net model is translated into the simulation model. However, in this approach, only the structure of the system is considered, and not its behavior.

Allam and Alla (1998) proposed using hybrid Petri Nets as the formal semiconductor manufacturing system model since traditional Petri Net models present difficulties for modeling and analysis of large scale systems. However, their work only considers the production resources; AMHS is not incorporated. While they compared the resulting hybrid Petri Net model to the corresponding Petri Net, they assumed the modelers would work directly with the hybrid Petri Net models.

Zhou and Jeng (1998) use Petri Nets to model semiconductor manufacturing because Petri Nets can be used to describe such complex discrete event systems precisely and enable both qualitative and quantitative analysis, for example to guarantee deadlock free operation. The authors point out that realistic scheduling rules still are too complicated to be captured precisely in Petri Net models.

Muller (2007) took a different approach to using Petri Net models to support simulation. He first created an object model representing the data required to describe a fab in sufficient detail to support simulation modeling: included were process tools, product routes, yields, and ramps. Then he created standard Petri Net models for specific tool types, e.g., single wafer processing, batch processing, etc., and a method for generating the composite Petri Net model from the object oriented data description. His approach does not require the designer to work directly with Petri Nets, and provides very fast simulations, but it does not incorporate AMHS.

Both Petri Nets and UML have been examined as tools for creating formal system models. Petri Nets can be very useful to automate input for simulation software, but are not easy for application domain experts to understand or to use for describing design intent. In addition, Petri Net models do not consider AMHS, a critical component in 300 mm wafer fabs. UML is somewhat easer for the application domain modelers to understand and use, but using UML requires agreement on standard syntax, and UML models can’t be directly translated into simulation models.

2.2 Fab Design Models

Yang et al (1999) described the problem of designing the layout of AMHS (Automated Material Handling system)in a wafer fab so that material handling supports the production requirements in the most cost effective manner. The authors model the problem as Mixed Integer Programming Problem and use Tabu search and Simulated Annealing to solve it. Clearly, only a limited description of behavior is available in such a model.

Kumar and Kumar (2001) also discussed semiconductor system design and analysis. Instead of a mixed integer programming model, they use Queuing Networks to analyze the system performance. However, when the system is large, it is difficult to apply queuing theory. The probability assumptions for each input parameter are also very critical for the model.

3 MODELING REQUIREMENTS

3.1 Drawbacks in Tradition Approaches

Published research relevant for the fab modeling problem tends to be narrowly focused on a particular analysis. No matter how useful the particular analysis might be, there is no existing computational infrastructure to support the deployment of the analysis in practice. As a result, there remains a large gap between fab modeling research and practice.

Formal models in the fab design literature are predominantly based on UML or Petri Nets. UML is designed for software engineering, and appears to have gained little traction in the factory modeling domain. Petri Net models become large and very intricate when modeling the kinds of complicated behavior typical of wafer fabs, and their use requires deep methodology expertise. Thus, they are not a good tool for factory domain experts to use to describe a particular application.

Petri Net and UML-based approaches, in general, address either the structure of the system or its behavior but not both. In addition, prior approaches attempt to construct a complete instance model, which in the case of a wafer fab, would lead to very large and unwieldy models with a great deal of very repetitive information.

A practical approach to formal models of wafer fabs must overcome these drawbacks, i.e., must be based on formal models that the domain experts can use, must address both structure and behavior, and must lead to models that are manageable in terms of size, yet still provide a complete and unambiguous specification. There are additional desirable attributes of a fab design tool, as described in the following subsections.

3.2 COTS Tools

Engineers commonly use a variety of commercial off the shelf (COTS) tools for authoring and analyzing factory data. Some tools relevant for fab simulation are: Excel and Access for numeric data; and Visio, AutoCAD, or FactoryCAD for layout data. Any new tool for supporting fab design decision making should be integrated with these kinds of standard authoring tools.

In addition, there are a variety of COTS solvers that are often or sometimes used in analyzing factory data. These include: statistical tools; math solvers like Mathematica or MathCAD; optimization tools; and discrete event simulation tools like Arena, AutoMOD, or eM-Plant. Any new tool for supporting fab design decision making also should easily integrate with these types of analysis tools.

3.3 Distributed System Modeling

In contemporary factory design, multiple disciplines in different geographical locations are involved, and multiple designers have to collaborate. To support and enable this, a fab modeling tool should incorporate formal semantics and application domain specification to improve communication among designers, and between designers and the computational tools they use.

3.4 Support Problem Solving Process

Fab modeling is only useful to the extent it supports problem solving. Key to any problem solving process is the ability to describe the artifact or system under consideration, to analyze its performance, and then to modify or further elaborate the description based on the analysis results. Thus any new tool for fab modeling should support efficient authoring of the fab description, easy access to useful analyses, and presentation of analysis results in easy to understand formats.

4 PROPOSED APPROACH

4.1 Overall Picture

In approaching the development of fab modeling tools, we distinguish two distinct types of modeling requirements. There is the *user modeling* of the fab, i.e., the complete and unambiguous specification of the fab. User modeling is done by domain experts, i.e., those charged with responsibility for the quality of the fab design or operations. For example, an expert in AMHS would be involved in specifying—creating the user model of —the AMHS in the fab. In addition, there is *analysis modeling*, which is the creation of the analysis models whose solutions will provide important information for guiding fab decision making. Analysis modeling requires expertise in the specific methodology used in analysis, and also expertise in the specific solver to be employed. For example, expertise in AutoMod would be required in creating an AutoMod model which would be used to simulate the performance of the AMHS.

Figure 1 summarizes our application framework, in which we separate modeling into an “off-line” activity which creates libraries and a model translator, and “on-line” activities which create instances of descriptive and analytic models, and engage solvers to compute results useful in the decision making process.

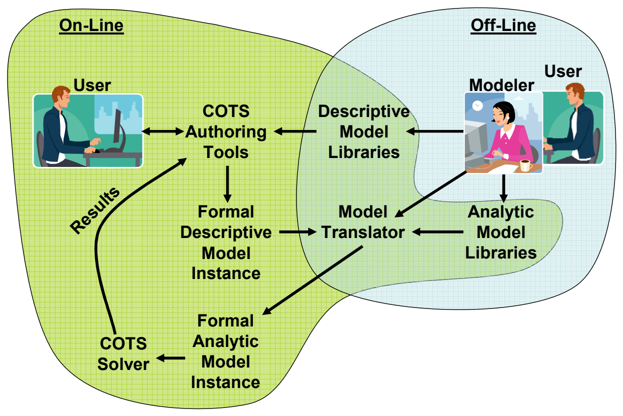

Figure 1: Application Framework

There are a number of important implications contained in figure 1.

- The role of the analytic modeler (the methodology and solver expert) is an off-line role; this person is not directly involved in the decision making process for a specific analysis.
- The methodology/solver expert creates two kinds of libraries and a translator tool; the descriptive library is used in COTS authoring tools to create the specification of the design, i.e., an instance of a descriptive model; the analysis library is used by the translator tool to create instances of an analysis model ready for a specific solver; the translator converts the instance of a descriptive model into an instance of an analysis model.
- Descriptive model libraries provide basic units of behavior and structure that have been identified by the user; the modeler has defined these units using a formal language, and converted them into a form usable by the corresponding COTS authoring tools.
- The descriptive instance model is constructed by factory experts using COTS authoring tools, augmented by the descriptive libraries.
- Analysis model libraries contain reusable units of structure and behavior in a form usable for specific analysis model (e.g., math formulations for queuing or factory physics analysis, blocks with operations for simulation analysis). These units correspond to those in the descriptive libraries and there are formal underlying relationships with descriptive libraries used by the model translator.
- The analysis model instance is created automatically by transforming the descriptive model instance through the model translator based on the relationship between the descriptive libraries and the analysis libraries.

In the framework, system modelers and analysis experts build the descriptive and analysis libraries and the mapping between them that enables the automated translation. This off-line activity, of course, will engage domain experts as well, but the domain experts are not expected to be experts in the formal system modeling language. Most importantly, this off-line activity captures the essential modeling knowledge in a re-usable form, and makes it usable by factory domain experts who are not experts in a specific analysis modeling methodology.

4.2 Modeling Framework

There are two essential elements of our modeling framework: how we conceptualize the wafer fab domain; and how we represent the wafer fab domain in order to create the descriptive and analysis libraries. For this modeling framework to be effective, it must address both the structure and the behavior of the wafer fab. Both the conceptual model and its realization are based on a formal language (in our case, on OMG SysML) and the use of the language to define specific semantics for the wafer fab domain.

We conceptualize the wafer fab as an event driven system which can be described as a collection of state machines, and these state machines interact in well-defined ways. The individual state machines are conceptualized as objects, having attributes and behaviors (or “methods” which can be invoked either internally or externally), and a set of well defined states, and clearly identified events that trigger state changes.

Our *reference model* of the wafer fab is the baseline from which both the descriptive and analysis libraries are derived. We create the reference model using a specific set of SysML diagrams with naming conventions and an organization that makes the diagrams relatively easy for domain experts to follow.

Finally, the reference model captures the basic elements of structure and behavior observed in the wafer fab domain. Furthermore, it provides the mechanisms for articulating new structures or behaviors in a way that allows them to be incorporated into the descriptive and analysis libraries..

4.3 State Machine Paradigm in Modeling Framework

While we view the wafer fab as a collection of interacting state machines, we are not necessarily interested in capturing every possible state of every device in the fab. Rather, we are interested in the modeling the fab at the level of material flow related events, i.e., lot dispatch/route/movement, process assign/start/end, or vehicle dispatching/routing/loading/unloading. Thus the fab system structural elements that interest us are the lots, the process and metrology tools, the vehicles, the stockers, the data repositories, and the controllers which directly impact the other structural elements.

Each of these system structural elements has an associated SysML state diagram. Figure 2 provides a simplified illustration for a lot and an AMHS controller. In this simplified illustration, the lot has four states: it is in process, in transit, being loaded into a tool input port, or waiting in a tool output port. Similarly, the AMHS controller is either idle or it is dispatching a vehicle to service a lot movement request.

Events are associated with state changes. For example, the job completed event occurs when a lot is finished processing on a tool. An empty vehicle event occurs when a vehicle completes a route and transfers a lot into a tool or stocker port.

When an event occurs, it may trigger a behavior, either for the structural element whose state has changed, or for some other structural element. For example, when a vehicle is dispatched by the AMHS controller, it could change its status from empty-unassigned state to the empty-assigned state. The empty-assigned event will trigger the behavior in which the vehicle travels to the assigned destination, i.e., the vehicle state change triggers a vehicle behavior. Some events will have effects beyond the structural element whose state has changed. For instance, when a vehicle unloads and becomes empty, not only will its state change, but also the AMHS controller will change from idle to dispatching the now empty-unassigned vehicle.

This type of interaction is illustrated in Figure 2, where the job-complete event changes the state of the lot, but also triggers an interaction with the AMHS controller, which goes from idle to dispatching. The interaction is represented in a SysML sequence diagram as the "Notify" message from the Lot lifeline to the AMHS lifeline. "Notify" can be thought of as a method of the AMHS structural element, and this method can be given a standard implementation, perhaps with a highly parameterized API.

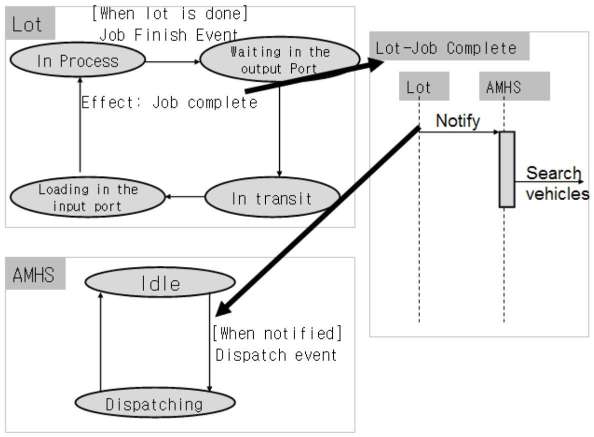

Figure 2: Behavior modeling in State machine Paradigm

To summarize, at the level of material flow, each structural element of the fab has a generic block representation in SysML, and an associated state diagram. The events which induce state changes are identified in the state diagram, and the guard conditions of the events can be associated with interactions between blocks in a sequence diagram. These interactions can be given standard implementations corresponding to particular kinds of analysis, e.g., simulation or queuing. Because both structure and behavior have been captured in a formal language, it is inherently computational, i.e., the representations can be processed for the purposes of model translation, using standard parsing and processing methods.

4.4 The Descriptive and Analysis Model Libraries

4.4.1 Descriptive Model Libraries

The fab reference model created in SysML is used to create a descriptive model library for each of the COTS authoring tools used by the domain experts, i.e., the fab decision making team. Conceptually, these libraries contain templates that the fab expert uses to describe the fab; the template components correspond to structure, behavior, or perhaps to some attribute of a structural element.

Clearly, both the reference model and the descriptive libraries are domain specific. Because these libraries are simply documents, they can be version controlled, and updated as needed if new structures or behaviors are identified and implemented. Note also that the libraries define the semantics of the wafer fab, i.e., they define a unifying terminology that can be used across all fab decision makers (and off-line modelers) when referring to specific structural and behavioral elements of the fab.

4.4.2 Analysis Model Libraries

The analytic model libraries will be used to construct specific instances of analysis models, intended for specific solvers. Constructing such models requires expert knowledge of both the analytical methodology and the specific solver to be used. Since the instance analysis model will be constructed from the instance descriptive model, the process also requires full and clear understanding of the domain reference model. In other words, the analysis modeler must understand what kinds of analysis are relevant, useful, and feasible, given the instance descriptive model.

The analysis model library provides, in effect, templates that will be selected, parameterized and used to construct the instance analysis model, and will be different for different kinds of analysis. For example, simulation blocks with attributes and operations are necessary for simulation analysis model, and one can imagine a modeling environment in which the analyst selects simulation blocks, places them in a workspace, assigns parameter values, and connects them as appropriate. In our framework, the analysis library provides the set of allowed templates.

Other types of analysis can be supported in a similar fashion. For example, one might be interested in the from/to flow times or flow volumes on the automated transport system. The analysis library would have representations of a flow network, and from-to flows, so that, conceptually, an analyst could construct the input to a network flow solver.

4.4.3 Mapping between Descriptive Libraries and Analytic Libraries

In our application framework, the construction of instance analysis models is done algorithmically, in the model translator (see figure 1). In other words, the descriptive model built by the fab decision maker using the descriptive libraries is automatically transformed to an instance of a particular kind of analysis model, e.g., a full fab simulation. The underlying translation mapping between the descriptive library and the analysis library is required for this translation to be feasible. In our framework, the mapping is accomplished in the reference model by using the formal modeling language. As an example, for simulation, elements of the descriptive libraries are linked to corresponding elements of the analysis model using generalization relationship. Through this relationship, some sharable parts of descriptive units become available to the inheriting analysis units, which may also have their own attributes and relationships. In summary, a descriptive element is linked to analytic elements corresponding to one or more types of analysis using generalization. The relationships provide a bridge between the descriptive model instance and corresponding analytic model instance.

4.5 Descriptive Model Instance

There may be a number of authoring tools involved in describing a wafer fab. For example, factory designers may use FactoryCAD^{TM} for capturing wafer fab layout, while process and machine instance data is stored in a database. For the behavior descriptive model, sequence diagrams in SysML can be used. Descriptive libraries are needed for each COTS authoring tool. Some examples of how to create these libraries are:

- Resources relevant to layout design are exported from the SysML reference model to create a template library in FactoryCAD^{TM}. (e.g., Machines, Vehicle, Stockers and Segment)
- Data schema for describing fab objects and processes is exported from the SysML reference model to database tools for authoring or collecting instance data. (e.g., Machine capacity data, Process route and Operation data)
- Using SysML, operation units in behavior libraries are used to create sequence diagrams describing behavior in the instance descriptive model.

Note that since several authoring tools may be used to describe the same system design, integrity and consistency of the descriptive libraries is essential. This is accomplished by generating each descriptive library from the underlying reference model.

4.6 Model Transformation

To summarize: off-line modeling creates a "domain reference model" which is implemented as libraries for COTS authoring tools, and libraries for specific analysis tools; fab experts author instance data, using standard COTS tools augmented by the domain specific libraries. The next step is the transformation of the resulting instance data to a specific analysis model instance. During the transformation, a translator specific to the analysis model parses necessary information in the descriptive model instance in accordance with domain specific semantics. The translator generates the corresponding analysis model instance in the format required by the analysis solver. If we have multiple transformation mechanisms and translators for various analysis models, one descriptive model instance can be translated to several analysis models, which results in much saving in modeling effort by factory designers. The following steps explain how to transform the model instance given in section 4.4 to simulation analysis model as an example.

- FactoryCAD layout drawing is exported as an SDX (Simulation Data eXchange) file.
- The simulation translator integrates all instance data from databases or spreadsheets, and layout data from the SDX file into a composite database.
- The simulation translator extracts the sequence of dispatching logic from SysML as txt file.
- Finally, the translator creates the script generating complete simulation model.

5 IMPLEMENTATION EXAMPLES

In order to demonstrate the proposed frameworks, we have developed proof of concept implementations of structure libraries, behavior libraries, authoring tools and model transformation. These are described briefly below, and in the cited papers which provide additional details.

5.1 Structure Modeling

The domain reference model defines the structural elements of a wafer fab, e.g., the single-wafer processing tool. In addition, the off-line modeler may create additional domain objects which extend the reference model. For example, an inspection machine is a type of the single processing tool. This generalization can be added back to the reference model, so that inspection machine becomes a block in the domain.

Using the generalization relationship, the domain reference model also will contain structural elements of analysis models. For instance, the inspection machine could be generalized as an M/M/1 queue from the queue analysis perspective. The mapping via the generalization relationship will be in the SysML model and be used in the model translation process.

An example of creating structure libraries for a factory layout authoring tool, FactoryCAD™, is describe in (Kwon and McGinnis 2007). The paper explains the integration between structure libraries created in SysML and template libraries in FactoryCAD™.

5.2 Behavior Modeling

There are two aspects of behavior modeling: the off-line behavior modeling which defines the basic behavioral elements in the behavior library; and the online behavior modeling for a specific fab design using these basic behavior elements. The basic behavioral elements are developed in the process of creating the domain reference model. For example, when the generic structural element "stocker" is defined, it will have its own basic behavior elements such as "check the number of the lots in the stocker" or " find the highest priority lot in the stocker".

For online behavior modeling, the basic behavior element is used to describe the behavior from user perspective. Since the behavior of a fab could be different in different companies, the modelers could compose the basic behavior elements for customized and complicated behavior actions.

The behavior in the semiconductor industry is a type of discrete and event-driven system. The online behavior modeler could use state-machine diagrams to define the events related to the states changed and sequence diagrams for the complicated behavior when the event happened.

For example, when a vehicle completes a delivery, it will be dispatched to the next waiting lot if there is a waiting log. The state machine diagram in Figure 3 shows the states for a vehicle. When the vehicle state changes from "Load-Assigned", the corresponding transition event is triggered and the sequence diagram in Figure 4 is executed.

In the sequence diagram in Figure 4, the vehicle notifies the AMHS controller, which then checks the waiting list and dispatches the vehicle if there is a lot waiting.

When the users want to describe a new dispatch rule, the state machine and sequence diagrams will be the user interface to describe the behavior logic. The user will select the corresponding basic behavior component and compose the new sequence diagram, and it will be transferred into the simulation automatically.

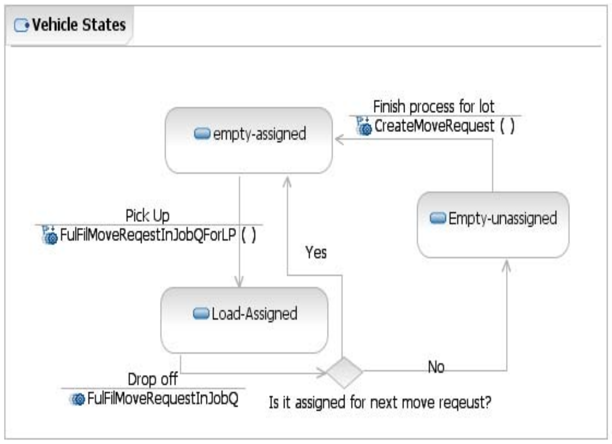

Figure 3: State machine diagram of vehicles

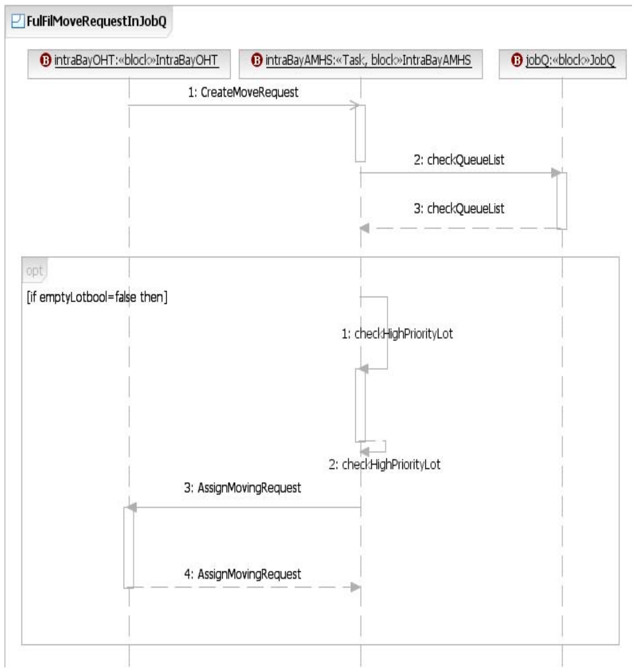

Figure 4: Sequence diagram for dispatching a vehicle to the next waiting lot

5.3 Authoring tools

Integration of libraries with authoring tools is described in McGinnis et al. (2006). There, a particular interface was created to integrate a variety of authoring tools, including FactoryCAD, text files, spreadsheets, or database. The interface enforced a step-wise collection and integration of the instance data which enabled consistency checking between different data sources. As an example, if the process plans referenced a tool that was not present in the tool database, the inconsistency could be detected and noted for correction.

5.4 Model Transformation

Huang et al. (2007) demonstrate a method to translate the formal instance descriptive model into different instance analysis models. When the formal model is constructed, the user can choose either simulation or queuing network analysis. The model translator will generate the corresponding instance analysis model automatically based on the relationship of the domain libraries and analysis libraries as captured in the domain reference model.

6 CONCLUSION AND FUTURE WORK

We have described two frameworks which, together, enable the creation of a new generation of on-demand fab simulation tools. . The state machine paradigm plays a key role in embodying important knowledge of both the application domain and the analysis domain in a formal model. This formal model, in turn, provides the elemental behavioral and structural units needed to compose the descriptive model. In the end, the descriptive model is translated into specific analysis models for specific solvers. We have provided several implementation examples to demonstrate the concepts.

There are many opportunities for further work in this area, not the least of which is to scale up the concepts presented to enable on-demand full scale wafer fab simulation. Other domains could be explored as well, such as automobile manufacturing, automated warehousing and transportation. When it comes to analysis, optimization is an analysis model that hasn't fully been explored, but is worthwhile to consider. In addition, we think the transformation process itself should be formalized. In spite of the formal languages, we haven't made a full use of the formalism for the transformation mechanism itself. We're considering whether the open source model transformation technology such as VIATRA is applicable to our research. Finally, there are many examples of analysis which requires some input from the designer; for instance, when the analyst wants to simulate only a portion of the overall design, or wants summary statistics for only a portion of the design. Effective mechanisms for interfacing this kind of input to the instance analysis model generation process are needed.

REFERENCES

Allam, M., and H. Alla. 1998. Modeling and Simulation of an Electronic Component Manufacturing System Using Hybrid Petri Nets. *IEEE Transactions on Semiconductor Manufacturing* 11(3).

Arief, L. B., and N. A. Speirs. 1999. Automatic Generation of Distributed System Simulations from UML. In *Proceedings of the 13th European Simulation Multiconference* (ESM'99), Warsaw, Poland, 85–91.

Arjuna Team 1994. C++SIM User's Guide. Department of Computing Science, University of Newcastle upon Tyne <http://cxxsim.ncl.ac.uk/>.

Huang, E., R. Ramamurthy, and L. F. McGinnis. 2007. System and simulation modeling using SYSML. In *Proceedings of the 2007 Winter Simulation Conference,* 796-803.

Hunter, R., and C. Humphreys. 2003. Trends in 300 mm factory automation. *Semiconductor International*, 26(6):60-64.

Kumar, S., and P. R. Kumar. 2001. Queuing network models in the design and analysis of semiconductor wafer Fabs. *IEEE Transactions on Robotics and Automation* 17(5).

Kwon, K., and L. F. McGinnis. 2007. SysML-based simulation framework for semiconductor manufacturing. In *Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference,* 1075–1080.

McGinnis, L. F., E. Huang, and K. Wu. 2006. Systems engineering and design of high-tech factories. In *Proceedings of the 2006 Winter Simulation Conference,* 1880-1886.

Mueller, R. 2007. *Specification and Automatic Generation of Simulation Models with Applications in Semiconductor Manufacturing*. Ph.D. thesis, Department of Industrial and Systems Engineering in Georgia Tech, Atlanta, Georgia.

Saldhana, J. A., S. M. Shatz, and Z. Hu, 2001. Formalization of object behavior and interactions from UML models. *International Journal of Software Engineering and Knowledge Engineering* 11(6):643.

Whittle, J. 2000. Formal approaches to systems analysis using UML: an overview. *Journal of Database Management* 11(4).

Yang, T., M. Rajasekharan, and B. A. Peters. 1999. Semiconductor fabrication facility design using a hybrid search methodology. *Computers & Industrial Engineering* 36:565-583.

Zhou, M., and M. D. Jeng. 1998. Modeling, analysis, simulation, scheduling, and control of semiconductor manufacturing systems: a Petri Net approach. *IEEE Transactions on Semiconductor Manufacturing* 11(3).

AUTHOR BIOGRAPHIES

LEON MCGINNIS is Gwaltney Professor of Manufacturing Systems at Georgia Tech, where he also serves as Director of the Product and Systems Lifecycle Management Center, Associate Director of the Manufacturing Research Center, and Director of the Keck Virtual Factory Lab. His research is focused on the representation of complex industrial systems, such as warehouses and factories, to enable analytic and simulation modeling to support performance assessment, behavioral prediction, and system design. His email address is <leon.mcginnis@gatech.edu>.

EDWARD HUANG is a Ph.D. candidate in the School of Industrial and Systems Engineering at the Georgia Institute of Technology. His research interests include simulation modeling and , as well as application of simulation in semiconductor manufacturing. His email address is <edward-huang@gatech.edu>.

KYSANG KWON is received the B.S. degree in mechanical engineering from Seoul National University, Seoul, Korea, in 2000, and and the M.S. degree from School of Industrial & Systems engineering in Georgia Tech in 2008. He was a mechanical engineer in SunYang Tech, Korea from 2000 to 2003, and a PLM (Product Lifecycle Management) consultant in UGS, Korea from 2003 to 2006. Currently, he is a Ph.D. student at ISyE, Georgia Tech. His research interests are object-oriented simulation, SysML-based simulation modeling and discrete event simulation in manufacturing industry in particular.
His e-mail address is <kkwon3@mail.gatech.edu>.