Mönch, Lars; Rose, Oliver; Sturm, Roland:

A simulation framework for the performance assessment of shop-floor control systems

URN: urn:nbn:de:gbv:ilm1-2014210331

Published OpenAccess: November 2014

Original published in:

Simulation : transactions of The Society for Modeling and Simulation International. - Thousand Oaks, Calif : Sage (ISSN 1741-3133). - 79 (2003) 3, S. 163-170.

DOI: 10.1177/0037549703256039

URL: http://dx.doi.org/10.1177/0037549703256039

[Visited: 2014-10-14]

A Simulation Framework for the Performance Assessment of Shop-Floor Control Systems

Lars Mönch, Oliver Rose and Roland Sturm

SIMULATION 2003 79: 163
DOI: 10.1177/0037549703256039

A Simulation Framework for the Performance Assessment of Shop-Floor Control Systems

Lars Mönch
Institut für Wirtschaftsinformatik
Technische Universität Ilmenau
D-98684 Ilmenau, Germany
Lars.Moench@tu-ilmenau.de

Oliver Rose
Lehrstuhl für Informatik III
Universität Würzburg
D-97074 Würzburg, Germany
rose@informatik.uni-wuerzburg.de

Roland Sturm
Department Cleanroom Manufacturing
Fraunhofer Institut Produktionstechnik und Automatisierung (Fraunhofer IPA)
D-70569 Stuttgart, Germany
sturm@ipa.fhg.de

The authors discuss the concept and design criteria for a framework that facilitates the performance assessment of shop-floo control systems. Their basic concept includes a simulation model that emulates the shop floo of a wafer fab, sends information to the control system, and receives information back from the control system. The shop-floo control system is realized as a separate module that interfaces to the simulator via a data layer that contains the current shop-floo status and the control information generated by the controller. The authors provide detailed information on how the simulation model and shop-floo control system communicate and how each system triggers events in the other system. They show how this framework supports the performance assessment of the shop-floo control system under consideration. They also present a prototype of the framework currently implemented in the course of the SRC/International Sematech FORCe project "Scheduling of Semiconductor Wafer Fabrication Facilities."

Keywords: Semiconductor manufacturing, shop-floo emulation via simulation, performance evaluation, shop-floo control software

1. Introduction

Recently, the electronics industry has become the largest industry in the world. The most important area in this industry is the manufacturing of integrated circuits (IC).

The production of integrated circuits on silicon wafers is characterized by

- reentrant process fl ws due to the layered structuring of the wafers,
- mix of different process types (i.e., batch processes vs. single-wafer processes),
- diverse product mix,
- inhomogeneous parallel machines with complicated dedication practices,
- sequence-dependent setup times,
- rework.

In the past, sources of reducing costs were decreasing the size of the chips, increasing the wafer sizes, and improving the yield while simultaneously improving operational processes inside the fabs.

Currently, there is a strong indication that the improvement of operational processes creates the best opportunity to realize the necessary cost reductions [1]. Therefore, the development of efficien state-of-the-art planning and control strategies is highly desirable in the semiconductor manufacturing domain.

SIMULATION, Vol. 79, Issue 3, March 2003 163-170

DOI: 10.1177/0037549703256039

Generally, the performance or the impact of new planning and control algorithms on manufacturing systems is unknown in advance. Thus, a simulation test bed that emulates the shop floo of an entire wafer fab would be highly desirable. We found several papers following this approach. Horiguchi, Raghavani, and Uzsoy [2] evaluated a new planning system for a wafer fab by using a discrete event simulator. Toba [3] focused on the evaluation of certain rescheduling strategies in a modern wafer fab, again using a discrete event simulator for emulation purposes of the shop floo .

We have participated in the project "Scheduling of Semiconductor Wafer Fabrication Facilities" of the Semiconductor Research Corporation (SRC)/International Sematech–funded FORCe (Factory Operation Research Center), which is working toward developing and implementing a modifie shifting bottleneck heuristic [4] for scheduling semiconductor wafer fabs. To evaluate and test this algorithm, we also need an appropriate simulation environment.

The semiconductor manufacturing domain currently lacks a simulation framework that allows evaluations of different planning, scheduling, and control software in an easy and unifie way.

In this paper, we present a simulation concept, including the description of various design decisions and a prototype of such a framework in semiconductor manufacturing. We want to point out, however, that this architecture can also be used by other industries. Performance studies using our approach will be presented in a later publication, but we have included two sample studies to illustrate the implementation of our concept for real-world problems.

The paper begins by describing the design criteria used for the simulation framework, continues with a discussion of the overall concept of the framework, and then presents the application of the framework in performance assessment situations. We finis the paper by providing two case studies using the concepts presented.

2. Framework Design Criteria

In a real wafer fab, a production planning and control tool obtains its information via the message bus of the manufacturing execution system (MES), stores this information in one or more databases, and computes fab control information based on this information. It then stores the control information in a database for evaluation purposes and fi nally transmits it to the shop floo , where the control actions are executed. The control information typically consists of dispatching lists for a particular tool or tool set or an automated loading device. The information is used to determine which lot to process next. At the moment, there are two approaches to create these dispatching lists: dispatching and scheduling software. In a dispatcher, a set of rules is applied to fin the appropriate sequence of lots for a given tool. A simple example of this type of control is firs in, firs out (FIFO) dispatching where the lots are sequenced in the order of their arrival at the tool under consideration. In a scheduler, a complex algorithm solves the problem of maximizing fab throughput while keeping cycle times low and on-time delivery percentages high. Again, the results are dispatching lists for the tools on the shop floo .

Therefore, the framework should support the following tasks:

- mimic the behavior of a real shop floo that communicates with a shop-floo control system over a message bus,
- provide a generic interface/data layer to plug in an arbitrary shop-floo control system (e.g., a planning, scheduling, or dispatching software system for semiconductor manufacturing),
- allow for user-specifi backups of the content of the data layer into a database. This database is required in real control systems to reset the controller to a reasonable state after a breakdown. In addition, the database can be used for statistical studies of the controller behavior.

The data layer works as the integrating part of the framework and allows for fast access to the data to run the planning, scheduling, or dispatching algorithms. The use of data layers of this type, also known as blackboards, is quite common in the artificia intelligence (AI) community [5, 6].

Before we provide a detailed description of the conceptual issues of the framework, we need to discuss the interoperability problem of the different applications (e.g., of the scheduler and simulation software packages). There are several approaches to solve this problem. The simplest way to exchange information is to use text files One application writes a data text file and the other application reads it. As soon as the fil format is specified the implementation of the read/write interfaces is straightforward. The main disadvantages of this approach are the low speed of the data exchange and the lack of a simple mechanism to synchronize applications.

A second approach is to have the data exchange performed via interprocess communication in the computers' memory. For this approach, all participating applications have to be able to share memory (i.e., it depends both on the operating system and the way the applications are implemented whether data can be transferred in this way).

The third approach requires that the applications communicate via network protocols. The applications may then even reside on computers with different operating systems. With this approach, we have to implement complex communication interfaces, causing a significan slowdown in the simulation execution speed.

The interoperability based on high-level architecture (HLA) functionality is becoming more and more common in simulation packages [7]. This does not hold, however, for shop-floo control systems or optimizer software packages in which interoperability functionality is not yet provided. Due to the lack of interoperability standards for this type of framework, significan work is required to defin

and implement appropriate data exchange interfaces for the participating applications.

3. Overall Concept of the Framework

3.1 Architecture for Interfacing between Shop-Floor Control Software and Simulation Model

In our design, we mimic the structure found in a real wafer fab (Fig. 1). The simulation model generates data, which is sent to a data model of the fab that stores this information. The message bus is replaced by a set of functions, providing a clear separation of data that reside inside the simulation model and data that are going to be transferred to the data model. This is necessary to facilitate the merging and splitting of the simulation model and controller with minimal effort. The shop-floo control software only makes use of the data stored in the data model to compute the schedule. No internal information from the simulation model is going to be used. The resulting schedule is stored in the data model for testing and serves as reference data that are required when a decision has to be made on whether to reschedule. The schedule is implemented in the fab by a set of functions.

3.2 Architecture for Interfacing Scheduler and Real Wafer Fab

The MES controls the shop floo in terms of lot progress in a real wafer fab. However, the MES systems core functionalities are product tracking and lot progress monitoring. Dispatching functionality is often limited in the state-of-the art MES software systems, and therefore many IC manufacturers use external dispatching software. The dispatching software contains rules that calculate a ranking for lots waiting to be processed at specifi equipment families (dispatching lists). The rules consider lot attributes, equipment status (setup information), and queuing information. However, the complete process fl w information of a lot is not required for the ranking process. In modern wafer fabs, there is a communication link between the equipment controllers (ECs) of the process equipment (tool) and the MES. Figure 2 shows a typical communication sequence between MES, ECs, and external dispatching software. Each time processing equipment has finishe the processing activities, the EC reports this event to the MES. The MES that takes control over the process fl w looks for the next process step of the lot and the required equipment family. Then the MES requests the dispatching software for lot ranking of the specifi dispatch list. As soon as the next tool of the equipment family becomes idle, the next lot is selected based on the dispatch list of the equipment family. In our simulation framework, the communication and the lot progress are tracked by the simulator itself. The ranking of the dispatch lists is performed in certain time intervals by the scheduler. In contrast to an external dispatching software, the algorithm of the scheduler requires the complete process fl w information (sequence, processing times, setup times, etc.) and further fab status information such as current equipment status.

The future MES system is a distributed system with several components. Core components such as work-in-process (WIP) tracking, machine management, process specificatio management, dispatcher, and so forth are necessary to control a wafer fab. The trend is that MES core components and external applications such as equipment control software communicate based on a standardized middleware. Future information technology (IT) architectures for wafer fabs must be open and comply with de jure or de facto standards such as Common Object Request Broker Architecture (CORBA) specification [8], CORBA services specifications several Semiconductor Equipment and Materials International (SEMI) standards, and COM/DCOM/.NET.

Not all information required for running advanced scheduling or even dispatching algorithms is available in state-of-the-art core MES components. For example, planned wafer release is typically provided in planning tools such as enterprise resource planning (ERP), supply chain management (SCM), or order release planning software; availabilities and current locations of reticles (photolithography masks) are handled by reticle management systems. Locations of lots in the fab, particularly in next-generation 300-mm fabs, are handled by automated material handling and storage systems and their control system framework or by equipment control systems and so forth.

To allow seamless integration to stand-alone scheduling or dispatching software, the functions interfacing to the simulation have to be easily replaced by functions interfacing to factory IT systems such as MES and ERP. This design principle requires no further changes in the software itself while allowing interface changes to be implemented with little additional work to other control and planning applications. In a wafer fab, the scheduler and the data model have to provide standardized interfaces for middleware communication.

As specifie by Semiconductor Equipment and Materials International (SEMI) E-105-0701 [9], the following interfaces have to be established:

- *Scheduling service interface:* this is the interface for services by the scheduler in response to requests from, for example, the MES dispatching component or some other client.
- *Scheduling factory input interface:* this is the interface used by other components to provide updated static or dynamic factory state information to the scheduler data model. The updates are typically lot-tracking information, order release information, current machine status information, and process fl w specificatio information.

3.3 Simulation Environment

The main purpose of the simulation environment is to emulate the behavior of a real wafer fabrication facility for

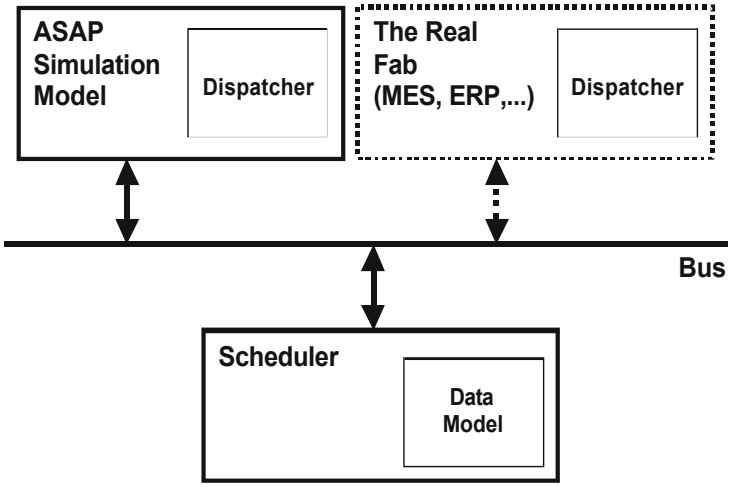

Figure 1. Structure of the simulation and performance access environment. MES = manufacturing execution system; ERP = enterprise resource planning; ASAP = AutoSched AP.

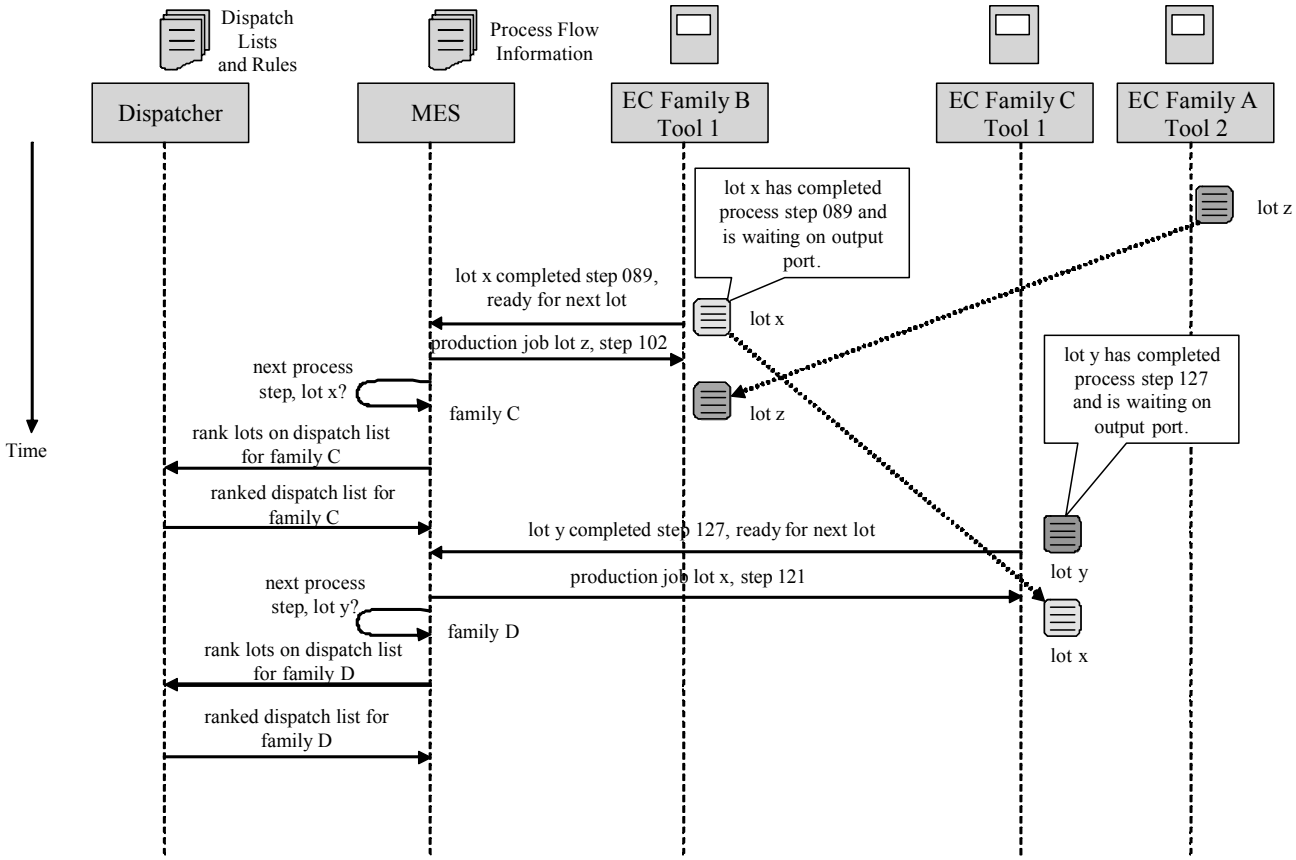

Figure 2. Simple communication scenario between manufacturing execution system (MES), equipment controller (EC), and external dispatching software

the performance assessment of shop-floo control software. Therefore, the simulation software used in the framework has to provide the functionality to model specifi characteristics of wafer fabs in the semiconductor industry. These characteristics include large process fl ws, multiple products, a high number of different manufacturing equipment types (resources), availability models (mean time to repair, mean time to failure, preventive maintenance), and semiconductor-specifi process and equipment timing models (batch processing, single-wafer processing, lot-based processing, etc.).

In addition, an event-based online communication between the simulation and the shop-floo control software has to be established to realize the control of the material fl w in the simulator based on decisions of the shop-floo control software. This spontaneous data exchange capability is a considerable problem for most of the current simulation packages. If the packages provide interfaces, they are most likely implemented in proprietary data objects and communication protocols. The same generally holds true for shop-floo control systems.

3.4 Shop-Floor Control System Triggering

The framework allows for an event-driven and a time-driven triggering of the shop-floo control system. In the case of event-driven triggering, events of the shop floo trigger the generation of control information. These triggers include bottleneck equipment breakdowns, exceeding a given limit for the deviation of a planned schedule, and lot movements. The shop-floo control system has to be called to compute a new schedule if these events occur. Event triggering is also intended for all rescheduling activities (i.e., minor changes of the schedule to adapt to state changes of the factory).

In the case of time-driven triggering, a timer starts the control system (e.g., at the beginning of a shift). This approach allows for the implementation of rolling-horizon approaches.

After finishin the action that launches the control algorithm, a delay can be set before the results of the algorithm are sent to the simulation model. Thus, we mimic the time-consuming behavior of a shop-floo control algorithm.

3.5 Data Layer Design

The data model stores manufacturing-specifi information required by the shop-floo controller to compute new schedules. A cache such as data storage can be found in most of the modern advanced planning systems (e.g., SAP's "Advanced Planner and Optimizer (APO)"). Advanced planning systems require a large amount of data from traditional ERP systems and/or MESs. This type of cache was introduced in such systems to avoid performance problems that occur while waiting for the data from a large number of database queries.

We segmented the data model to facilitate an appropriate storage of the types of data. We distinguish between static and dynamic data types in the data model. The static data include data such as information about process fl ws (technological routes), required reticles, setup information, existing tool sets, tool dedications, and data for calculating processing times. The second type of data, termed *dynamic data*, includes lot release information, lot states, tool states, setup state of a certain tool, and reticle states.

The data model is used in three different situations:

1. initialization of the data model at the beginning of the simulation,
2. update of the objects during the simulation run (or during the runtime of the shop-floo control algorithm in a real environment) in an event-driven manner,
3. reading of data model information by the shop-floo component.

The data model consists of classes representing business objects and classes that are used to encapsulate technical functionality. We focused on applying object-oriented modeling techniques because these techniques allow for an easy integration of association and aggregation relations between the real-world objects found in a wafer fab into the software.

The suggested class hierarchy is shown in Figure 3. Double-linked lists, containing pointers of the represented objects, are the basic abstract data structures that are used in the data model.

4. Performance Assessment

The suggested framework allows testing the performance of different shop-floo models and also of different shop-floo control systems. It is possible to compare new shop-floo control systems against built-in rules of the simulator (rules that mimic dispatch rules commonly used to control the real shop floor)

To that end, we implemented a data structure that collects historical information of all lots. For each step/operation of the lot's route, we collect the following information:

- simulated lot ready time,
- simulated lot completion time,
- planned lot ready time (if a scheduling algorithm is going to be assessed),
- planned lot completion time,
- waiting time experienced by the lot caused by tool unavailability since the last schedule generation instant.

As soon as we turn off all stochastic elements of the simulation model (e.g., machine failure), there should be very small differences between simulated and planned time stamps. This provides a method to check the implementation of the scheduling algorithm and to validate/verify

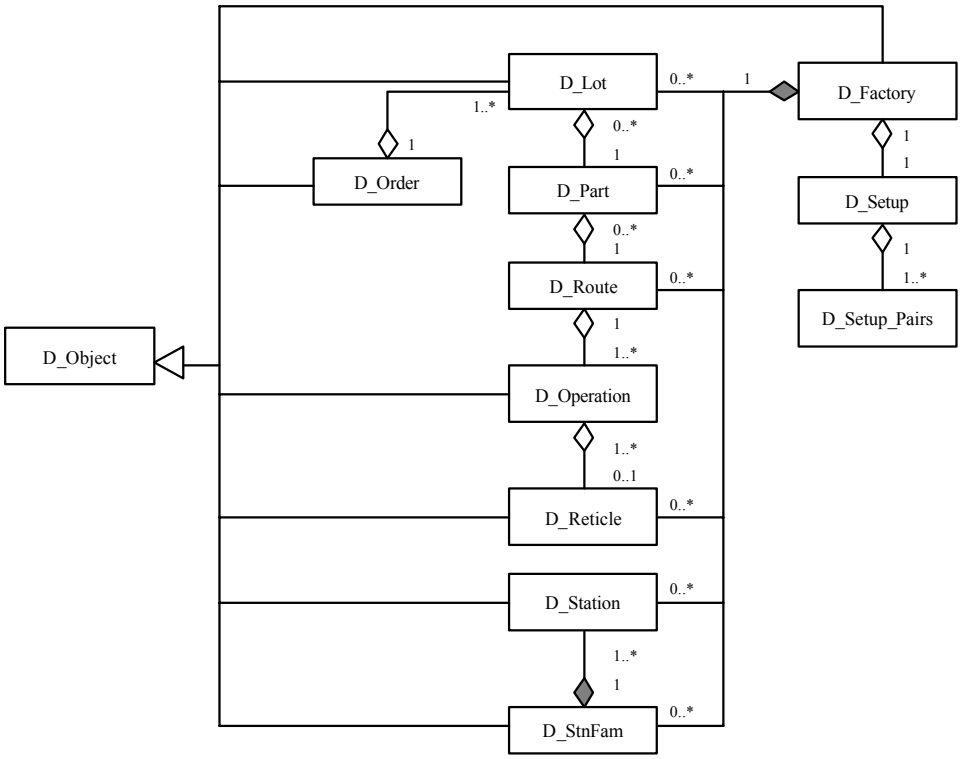

Figure 3. Class hierarchy of the data model in UML notation

the implementation. In a stochastic fab model, this difference in time stamps can be used to determine the need for rescheduling. We expect that most of the difference is caused by tool downtimes; therefore, we measure the amount of time each lot is delayed due to nonproductive equipment.

4.1 Implementation Language

The design, requirements, and implementation details of the framework suggest a simulation language that is robust, familiar within semiconductor manufacturing, and capable of interfacing with the many scheduling and control systems discussed. AutoSched AP (ASAP) 7.1 (Brooks–PRI Automation) was therefore selected as the simulation environment to be used in the prototype implementation. This decision was based on the evaluation results for general semiconductor modeling applications and the framework-specifi requirements, including the following:

- interprocess communication via memory with external software systems (here the shop-floo control software),
- ease of model customization with C++ code by developing our own dynamic link libraries (DLL),
- general acceptance and familiarity in semiconductor manufacturing, and
- user-friendly building of large-scale models.

The interprocess communication and the opportunity to integrate our own DLL distinguish ASAP as different from other simulation tools and was probably the most signifi cant influenc on our fina decision.

4.2 Prototype Implementation

In this section, we report on the usage of the framework in two different situations. First, we use the framework for testing the shifting bottleneck heuristic of the FORCe project (for details on this project, see Fowler et al. [10]). The second example is the performance assessment of a production-scheduling algorithm for lots in a wafer fab on a coarser level of granularity.

4.2.1 Scheduling Based on the Shifting Bottleneck Heuristic

The following parts of the framework are already realized in the "Scheduling of Semiconductor Wafer Fabrication Facilities" FORCe project (2001-NJ-880) [10].

Initialization of the static information of the data model. We parse the ASAP fab model text file to obtain the static data portion of the data model, such as tool set or product routes. The parser copies textual information from the model text file to the appropriate data structures of the

data model. Each time a new fab model is going to be customized, the parser must be reviewed to determine if new constructs were used to model the fab and therefore require parser extensions.

Update functions for the data model. The data model update is event driven. As soon as one of the following events takes place, data are transferred from the simulation to the data model: lot enters fab, lot leaves fab, lot enters operation, lot enters load port, lot leaves load port, reticle new location, reticle new state, and station new state. These events are mapped to ASAP internal events. ASAP has a notification/subscription/publicatio mechanism that facilitates calling of C++ code upon ASAP internal events. At the beginning of the simulation run, the custom code subscribes to the publication routines of all events of interest. As soon as such an event takes place, the custom code is started to update the data model.

Starting the scheduler and delayed reception of its results. We use custom lots with custom action lists to start the scheduler. For instance, to achieve one scheduler run per shift, we generate a custom lot every 8 hours. These lots possess action lists that only contain the single action of calling the external scheduler or dispatcher. After the action that runs the scheduler, a delay can be set before the scheduler results are sent back to the simulation model. This delay mimics the computation time of the schedule in a real wafer fab. In addition, the scheduler results are added to the data model.

Implementation of a scheduler ASAP rule. To make use of the external schedule, we implemented a custom rule that dispatches lots based on a lot list and start times that are provided by the scheduler. Each tool uses this rule for dispatching.

Testing of the implemented framework. We wrote an external FIFO dispatcher for testing the above modules and interfaces. This dispatcher simply scans the list of waiting lots at each tool and writes this list of lots back to the simulation model after a given delay. It turned out that our external FIFO dispatcher leads to the same performance results as the internal ASAP dispatching rule if the interval between external dispatcher runs is less than 30 minutes. From this result, we concluded that our implementation has no serious bugs.

Adaptation of the FORCe scheduler. As expected from our conceptual considerations above, only the data input and output interfaces of the FORCe scheduler had to be adapted to the data model. The implementation of these data-mapping procedures was straightforward because both the ASAP customization and the FORCe scheduler are implemented in C++.

4.2.2 Scheduling Based on a Beam-Search Algorithm

Very often, only rough schedules are required in semiconductor manufacturing (i.e., start and end dates have to be calculated only for a set of consecutive process steps and not for each single process step). Habenicht and Mönch [11] suggested the use of a finite-capacit beam-search algorithm to solve this task. The data model was extended to store additional information such as the segmentation of the routes and the rough schedules. The beam-search algorithm was developed using the commercial ILOG libraries [12]. The scheduling algorithms were integrated into the suggested framework with little effort because only the interfaces to the data model had to be implemented. Thus, the test of the beam-search algorithm in a dynamic environment was considerably simplifie compared to developing a new simulation test environment just for this particular purpose.

4.3 Customization Effort and Runtime Performance for ASAP Fab Models

To use our framework, the following customization steps are required:

- adapting the data model (as a consequence, the data model initialization function, the event processing, and the parser have to be updated),
- replacing the given dispatching rule by our custom rule,
- adding custom lots for starting the external scheduler or dispatcher, a custom action list, a few external definitions and a few user attributes.

Only the firs point requires a moderate amount of work because most of the data structures that are typical for the semiconductor industry are already available in the prototype. The remaining steps are copy actions from a given sample fab model and can be performed in only a few minutes.

The framework has only a minor effect on the runtime of the simulation model. The parsing of the model data file takes less time than the time ASAP consumes to initialize the same model without our framework. We were not able to detect an increase in simulation runtime due to the event-based data model updates or our custom dispatching rule.

The framework leads to an increased memory consumption of a fab model during runtime because of the data model. Its size depends on the size of the factory, the simulated interval, and the number of runs of the external scheduler or dispatcher. The size of the data model, however, grows during simulation runtime only if the tracking of statistical data is switched on. Then, historical information for each lot and each scheduler/dispatcher run is kept in the data model for performance assessment purposes after the simulation run.

5. Conclusions

In this paper, we present a framework allowing performance assessment of shop-floo control systems. We suggest a framework based on emulation of the shop floo via simulation that allows the test of different shop-floo models and different control systems in a unifie manner. We

discuss the design decisions for the framework and report on its implementation. We discuss the application of the framework for two real-world scenarios.

6. Acknowledgments

The authors gratefully acknowledge the support of the Semiconductor Research Corporation (2001-NJ-880). They thank Detlef Papst, Jens Zimmermann, and Maryam Zehtaban for their valuable programming efforts within the FORCe project.

7. References

[1] Schömig, A., and J. W. Fowler. 2000. Modeling semiconductor manufacturing operations. In *Proceedings of the 9th ASIM Dedicated Conference Simulation in Production and Logistics*, pp. 55-64.

[2] Horiguchi, K., N. Raghavani, and R. Uzsoy. 2001. Finite capacity production planning algorithms for a semiconductor wafer fabrication facility. *International Journal of Production Research* 39 (5): 825-42.

[3] Toba, H. 2000. Segment-based approach for real-time reactive rescheduling for automated manufacturing control. *IEEE Transactions on Semiconductor Manufacturing* 13 (3): 264-72.

[4] Mason, S., J. W. Fowler, and M. W. Carlyle. 2001. A modifie shifting bottleneck heuristic for minimizing the total weighted tardiness in a semiconductor wafer fab. *Journal of Scheduling* 5:247-62.

[5] Fordyce, K., and G. Sullivan. 1994. Logistics management system (LMS): Integrating decision technology for dispatch scheduling in semiconductor manufacturing. In *Intelligent scheduling*, edited by M. Zweben and M. S. Fox, 473-516. San Francisco, CA: Morgan Kaufmann.

[6] Henoch, J., and H. Ulrich. 2000. HIDES: Towards an agent-based simulator. In *Proceedings of Agent Based Simulation: International Workshop 2000*, Passau, Germany, pp. 259-63.

[7] McLean, C., and F. Riddick. 2000. The IMS mission architecture for distributed manufacturing simulation. In *Proceedings of the 2000 Winter Simulation Conference*, pp. 1539-48.

[8] Object Management Group. 2001. CORBA/IIOP specificatio 2.6. Retrieved from www.omg.org

[9] Semiconductor Equipment and Materials International (SEMI). 2000. *SEMI E105-1000, provisional specificatio for CIM framework scheduling component*. Austin, TX: SEMI.

[10] Fowler, J. W., S. Brown, M. W. Carlyle, E. S. Gel, S. M. Mason, L. Mönch, O. Rose, G. C. Runger, and R. Sturm. 2002. A modifie shifting bottleneck heuristic for scheduling semiconductor wafer fabrication facilities. In *Proceedings of the 12th International Conference on Flexible Automation and Intelligent Manufacturing*, pp. 1231-6.

[11] Habenicht, I., and L. Mönch. 2002. A finite-capacit beam-search-algorithm for production scheduling in semiconductor manufacturing. In *Proceedings of the 2002 Winter Simulation Conference*, pp. 1406-13.

[12] Le Pape, C. 1994. Implementation of resource constraints in ILOG SCHEDULE: A library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering* 3:55-66.

Lars Mönch *is an assistant professor in the Institut für Wirtschaftsinformatik, Technische Universität Ilmenau, Ilmenau, Germany.*

Oliver Rose *is an assistant professor in the Lehrstuhl für Informatik III, Universität Würzburg, Würzburg, Germany.*

Roland Sturm *is a project manager and scientist in the Department Cleanroom Manufacturing, Fraunhofer Institut Produktionstechnik und Automatisierung (Fraunhofer IPA), Stuttgart, Germany.*