Proceedings of the 2018 Winter Simulation Conference
M. Rabe, A.A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, eds.

AUGMENTED GENERIC FLOW VISUALIZATION AND AGGREGATION FOR A MULTI-PRODUCT SEMICONDUCTOR FAB

Thomas Winkler
Ralf Sprenger

GLOBALFOUNDRIES Dresden
Module One LLC & Co. KG
Wilschdorfer Landstrasse 101
Dresden, 01109, GERMANY

ABSTRACT

Modern semiconductor foundry business involves dozens of different products sharing the same production line. With each of them having distinct flows with potentially different mask layers, process steps and dedicated tools, it is not easy to get a good overview. Furthermore, analyzing simulation results is difficult because the flow of material on the shopfloor inside the simulation is difficult to understand. We describe a dynamic approach of automated, interactive, aggregated flow visualization based on nodes and edges graphs, augmented by current state and simulation data like waiting and upcoming material, tool states and line holds. This allows for a profound analysis of the current state of the production line and the material flow anticipated by the simulation considering all constraints.

1 INTRODUCTION

Semiconductor manufacturing nowadays typically requires hundreds of different process, measurement and other steps to be executed on a single wafer of a given product. As a foundry business typically manufactures a large number of diverse products, product flow is highly complex and difficult to visualize.

Simulation of this complexity requires modelling many restrictions that occur. Any process step for every single product has its own machines that are allowed or blocked due to process releases and inhibits. Different time link constraints (Winkler et al. 2016) occur that lead to different behaviour of the material flow.

Evaluation of the quality of the simulation model beyond simple measures like the total number of moves or the speed of single products is nontrivial. It is however essential to continuously verify and validate the quality to get a valid simulation result. Often, if a product does not behave as expected, e.g. gets stuck somewhere, simulation input files (the model) are analyzed to find the root cause. This could be caused by a real issue in the line or just by a weakness in the simulation model. Analyzing tabular text files is usually a tedious process, greatly improvable by a suitable graphical representation.

Beside of an understanding of how all of the process steps are connected and where different products can use which machines, the described approach allows for a graphical analysis of how the material will flow within the simulation and what the model looks like.

While a handful of commercial simulation tools does provide a graphical representation of the simulated model (Cimino et al. 2010), such visualizations are usually insufficient to understand the entire complexity of the fab; they show the machine park, but they fail to provide a flow-based visualization. In contrast, our solution allows:

- The visualization and comparison of flows and the deep connections between machine groups, process steps and layers.
- A profound analysis of the current fab state including where material is standing, current tool states, restrictions, dedications, inhibits, lineholds and time link constraints.
- An assessment of how the WIP will flow using a discrete-event simulation upcoming WIP forecast.

In comparison, most works (Chen et al. 2008, Niedermayer and Rose 2003) analyze only either the overall line or a single point of interest (e.g. a single cluster tool) but do not provide such a comprehensive graphical insight with interactive drill-down into the details of the model.

In Section 2 we will show in detail how we utilize graphs to visualize the flow along process steps. Section 3 describes the interactive augmentation of the graph by relevant and useful context details. In Section 4 we will discuss the architecture, performance and scalability of our solution. Finally, in Section 5 we sum up and elaborate on potential future use cases and improvements. We did an extensive search for related literature but to our knowledge nothing similar has been published.

2 VISUALIZATION OF PRODUCTION FLOWS

This section describes the requirements to the graph visualization and the different types of node aggregation that are used.

2.1 Graph-based visualization

Production flows in semiconductor fabs describe the sequence of steps for a wafer to become a functioning product. Typically, flows are broken up into several consecutive mask layers according to the actual subsequent stack-up of the structures on the wafer. Each layer then consists of dozens of process, measurement and other operations which have to be followed along in order to create and verify the actual structures and assist the wafer handling during production.

In order to effectively visualize such flows, we have chosen a graph-based representation with complexity folding, where a node represents operation(s) or a layer and the edges represent the flow from one to the other. This approach offers several advantages:

- A graph is a versatile and universal modelling and visualization tool. Many users are already familiar with this concept and there is a lot of existing knowledge we can refer to.
- An existing generic graph rendering framework can be used, where we enter only the nodes and edges and the actual visualization work will be performed for us. There are numerous solutions on the market, ranging from free open-source software to more expensive commercial libraries on different platforms and programming languages.
- Some graph rendering frameworks offer built-in automatic graph layouting. Good layouting of complex graphs is not at all trivial to implement. For example, the layout algorithm of our choice consists of more than 5,000 lines of code, where similar algorithms are described in theses each of several dozens of pages in length, e.g. Tušla (2017), Reynolds (1997). By finding a suitable framework, we can save the enormous efforts and concentrate on our actual problem.
- Nodes in a graph can easily be hierarchical, i.e. there may be a sub-graph inside each node, making it expandable and foldable. This is perfectly suitable for complexity folding, e.g. by collapsing entire layers into single nodes until the user shows interest about them.

We will present the selected graph rendering framework and the supplementary architecture in detail in Section 4. For the rest of this Section, we will focus on the logical representation of such graphs from a user's point of view.

2.2 Node Aggregation

By default, each graph node represents an entire layer. This generates a compact overview of the investigated flows (Figure 1, topmost graph). By selecting layers, the process operations in these layers are shown as individual nodes instead of the aggregated layer node (Figure 1, center graph). Consecutive measurement and/or other operations are foldable to single nodes ("non-process operation chains") in order to reduce complexity, or may be shown in full detail instead (Figure 1, bottommost graph).

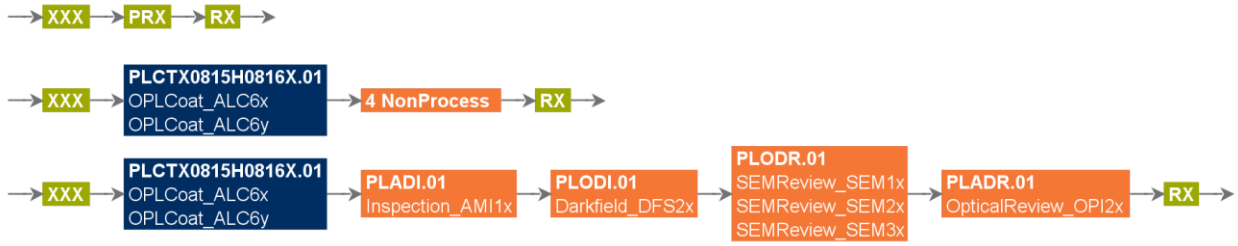


Figure 1: Simple graph with complexity unfolding.

To reduce the number of nodes and focus on tool groups instead of individual operations, the aggregation may be switched to show one node per machine group. This is especially useful for observing bottlenecks in the context of their predecessors and successors. However, because of the re-entrant nature of semiconductor manufacturing, machine groups usually appear several times along processing flows. Thus, having only one node per machine group often leads to edges going backwards, seemingly against the flow. Depending on the analysis, this may or may not be desired. Therefore, we also implemented the possibility to avoid loops by automatically splitting entities to multiple nodes where necessary (Figure 2).

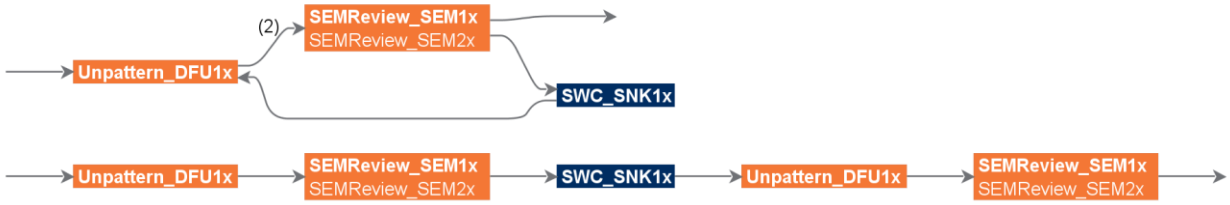


Figure 2: Graph aggregated by machine group with optional loop avoidance.

Aggregating by generic operation descriptions instead of full operation names is an approach that also can be used to graphically inspect differences in flows. By utilizing the automatic graph layouting of the graph rendering framework, this automatically delivers a common sort order for operations and/or operations of multiple, unequal flows.

3 ANALYTICS

We start by describing how we extended the graph with additional information and how the user is interacting with the graphs. Furthermore, we built in a history so that the user is also able to analyze situations in the past. Finally, analytic use cases are described.

3.1 Augmented Nodes and Edges

While a generic graphical representation of arbitrary combinations and sections of production flows is already valuable by itself, in addition to that, we augmented the nodes and edges of the graph by useful associated information.

It is most important for a semiconductor manufacturing foundry to have timely, accurate and contextual information about the following measures which are all contained in the node visualization shown in Figure 3:

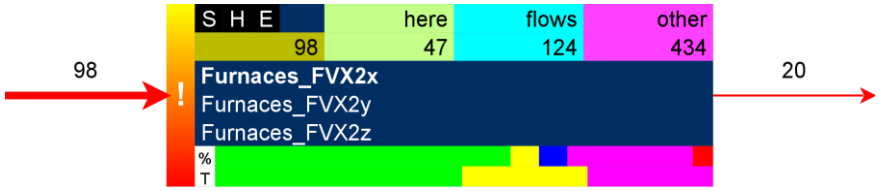


Figure 3: Augmented graph node.

- WIP (number of wafers) waiting or being processed at the given operations, layers or tool groups (green, "here: 47").
- If aggregating by tool sets: WIP waiting or being processed at different operations in the selected flows, as this will share the same tool capacity with the currently inspected operations' WIP (cyan, "flows: 124").
- WIP waiting or being processed in flows not currently selected. While we want to focus mainly on the user's selection, it is still highly useful to have this contextual information as this WIP will share the same tool capacity with the currently inspected operations' WIP (violet, "other: 434").
- Upcoming WIP (number of wafers per time frame, e.g. wafers/h). For accurate forecasting of these numbers, we use the results of a discrete-event fab simulation. The number can be shown on the edges. The sum of the incoming edges is also shown in the yellow/ochre segment in the upper left of the node. In addition, the thickness of the drawn edge is adjusted by the amount of upcoming WIP, i.e. edges where a high amount of upcoming WIP is predicted will be drawn thicker than edges carrying low or no upcoming WIP ("98", "20").
- Tool states (productive, standby, engineering, scheduled/unscheduled down, non-scheduled). We chose to display those graphically in the form of stacked bars where the node's width denotes 100% of the time (when looking at the last hour) or 100% of the tools (when looking at the current state). This quickly gives an overview about the number of tools currently available / having been available on average during the last hour (or any other configurable time frame) (stacked bars: "%", "T").
- Line holds, i.e. operations where material is currently held for yield reasons (e.g. because a measurement result must be awaited) or for WIP flow control reasons (e.g. because some material is moving too fast / other material too slow). We mark such operations with a yellow-to-red gradient exclamation mark which immediately attracts the user's attention ("!").

3.2 Graph interactivity and tabular details

In addition to the information shown directly in the nodes and on the edges, hovering over any element will show a tooltip with a description and additional insights (Figure 4). Furthermore, it is possible to get detailed data by clicking on the displayed values. For example, clicking on a WIP indicator will show a list of the individual lots leading to that number, while clicking on a line hold emblem immediately yields the list of associated line holds with introduction date, type and reason description, whereas clicking on an edge with upcoming WIP will show the split of products and customers. A click on the 'S' button on the upper left of a node displays a list of steps aggregated by this node (similar to the lot list at the bottom of Figure 4), declaring also the quantitative split of WIP and upcoming WIP between operations, products and customers.

The lot list presents basic lot information such as lot type, product, customer, carrier, flow, current operation, wafer quantity, priority, global rank, initial and current committed fab out date, current location, current operation waiting duration and due date, whether the lot is processing, waiting, blocked

or on hold and if so the hold reason. For lots currently in a time link area, the enter timestamp and how much time is left until the control limit will expire is also shown. For immediate attention, a few important columns like the priority, the global rank and the remaining time to control limit are highlighted with colors. By selecting a lot in the lot list, the corresponding edges will turn into a green tint in order to visually guide the path of the lot (Figure 4).

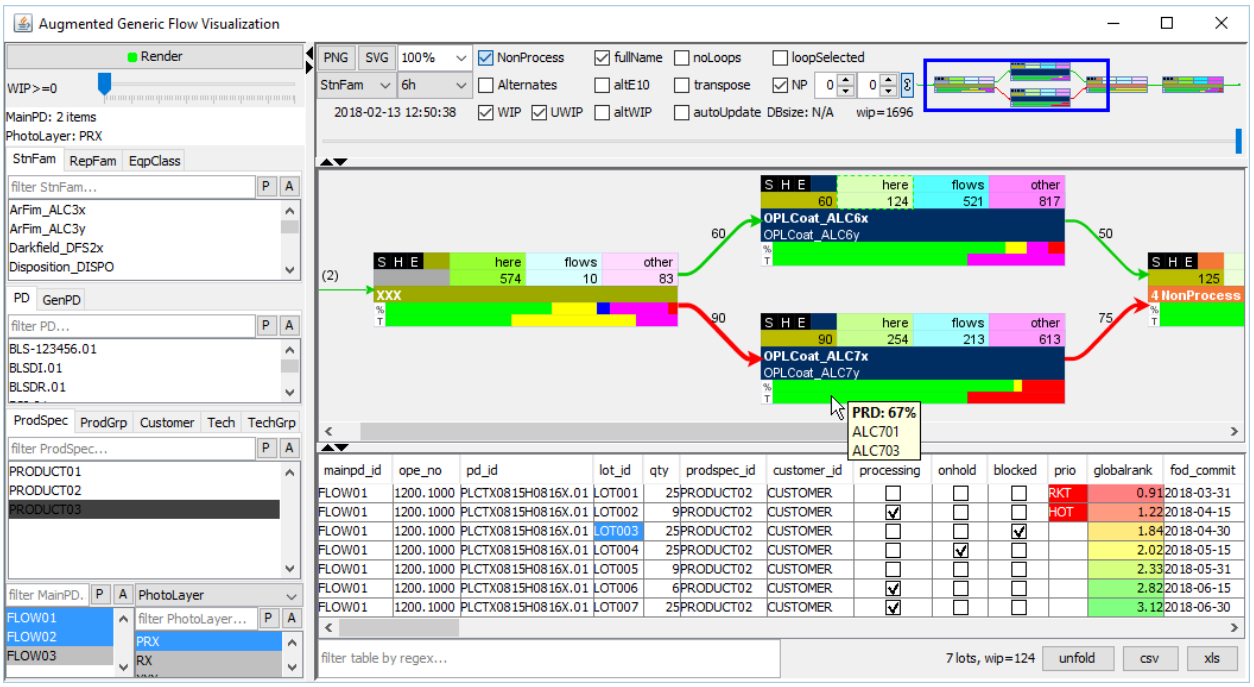


Figure 4: Entire user interface (sidebar, top panel, graph, lot list, tool tip).

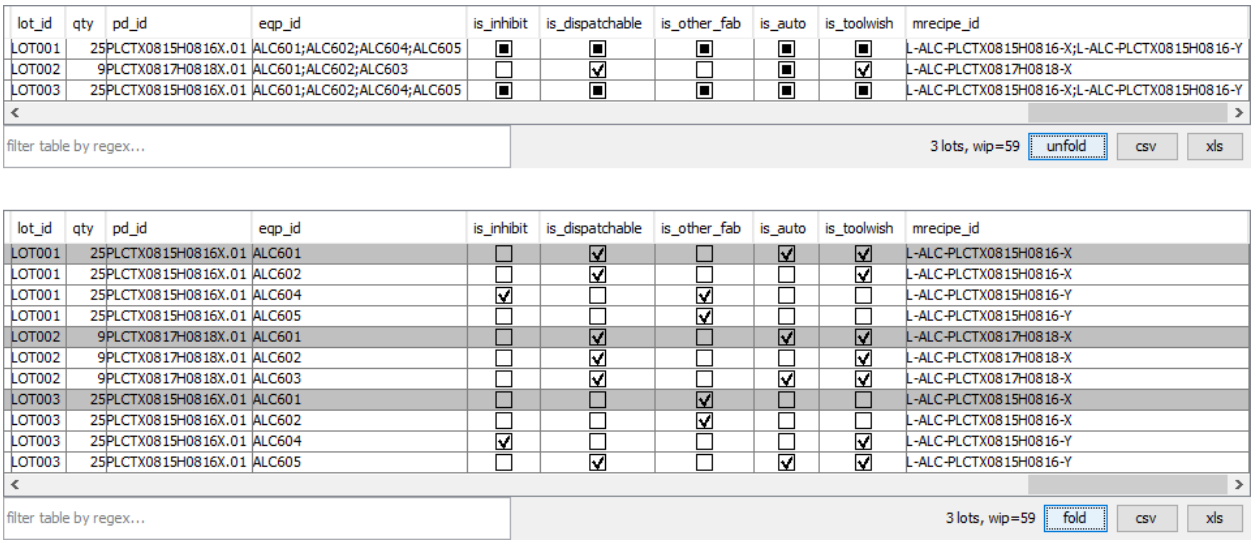
Some lot details are specific to the tool the lot would run on. By default, these details are aggregated so that only one row per lot has to be shown. If necessary, the user can ‘unfold’ the aggregated rows into individual ones, one for each lot-tool combination (Figure 5). For example, some tools may be located in a different fab building, requiring an inter-fab transport for processing the lot there. As this leads to additional cycle time and an inter-fab bridge has limited capacity, these tools are usually only chosen for a good reason, e.g. if all other tools are inhibited. For further investigation these tables are exportable into arbitrary spreadsheet software in csv or xls format.

3.3 Data Historization

As the situation (high/low WIP, tool downs, etc.) of the fab is highly dynamic, it is often desirable to have a reporting reaching back into the past. We therefore enriched the system with a time slider (Figure 6, above the graph) to select an arbitrary snapshot recorded during the last few days for visualization.

In order to easily find the most interesting snapshots in the past, the WIP of any node can be viewed over time in a line chart (Figure 6, below the graph). Usually, as problems appear in the line, WIP piles up, leading to an easily spottable spike in the chart. This snapshot can then be specifically navigated to for further investigation using the time slider. Picking a past snapshot via the time slider will also show the upcoming WIP forecast of the simulation run related to the selected snapshot (Figure 7).

While we chose to execute and archive one snapshot per hour, this could be easily adapted to a tighter or looser schedule, or even an event-based schedule (e.g. on tool down). The system is flexible enough to allow arbitrary non-equidistant snapshot dates, for example it could be reconfigured to have a tight schedule for the last week and a loose schedule for another three weeks.



lot_id	qty	pd_id	eqp_id	is_inhibit	is_dispatchable	is_other_fab	is_auto	is_toolwish	mrecipe_id
LOT001	25	PLCTX0815H0816X.01	ALC601;ALC602;ALC604;ALC605	■	■	■	■	■	L-ALC-PLCTX0815H0816-X;L-ALC-PLCTX0815H0816-Y
LOT002	9	PLCTX0817H0818X.01	ALC601;ALC602;ALC603	☐	☑	☐	■	☑	L-ALC-PLCTX0817H0818-X
LOT003	25	PLCTX0815H0816X.01	ALC601;ALC602;ALC604;ALC605	■	■	■	■	■	L-ALC-PLCTX0815H0816-X;L-ALC-PLCTX0815H0816-Y

filter table by regex... 3 lots, wip=59 unfold csv xls

lot_id	qty	pd_id	eqp_id	is_inhibit	is_dispatchable	is_other_fab	is_auto	is_toolwish	mrecipe_id
LOT001	25	PLCTX0815H0816X.01	ALC601	☐	☑	☐	☑	☑	L-ALC-PLCTX0815H0816-X
LOT001	25	PLCTX0815H0816X.01	ALC602	☐	☑	☐	☐	☑	L-ALC-PLCTX0815H0816-X
LOT001	25	PLCTX0815H0816X.01	ALC604	☑	☐	☑	☐	☐	L-ALC-PLCTX0815H0816-Y
LOT001	25	PLCTX0815H0816X.01	ALC605	☐	☐	☑	☐	☐	L-ALC-PLCTX0815H0816-Y
LOT002	9	PLCTX0817H0818X.01	ALC601	☐	☑	☐	☑	☑	L-ALC-PLCTX0817H0818-X
LOT002	9	PLCTX0817H0818X.01	ALC602	☐	☑	☐	☐	☑	L-ALC-PLCTX0817H0818-X
LOT002	9	PLCTX0817H0818X.01	ALC603	☐	☑	☐	☑	☑	L-ALC-PLCTX0817H0818-X
LOT003	25	PLCTX0815H0816X.01	ALC601	☐	☐	☑	☐	☐	L-ALC-PLCTX0815H0816-X
LOT003	25	PLCTX0815H0816X.01	ALC602	☐	☐	☑	☐	☐	L-ALC-PLCTX0815H0816-X
LOT003	25	PLCTX0815H0816X.01	ALC604	☑	☐	☐	☐	☑	L-ALC-PLCTX0815H0816-Y
LOT003	25	PLCTX0815H0816X.01	ALC605	☐	☑	☐	☑	☑	L-ALC-PLCTX0815H0816-Y

filter table by regex... 3 lots, wip=59 fold csv xls

Figure 5: Folded and unfolded version of lot list.

3.4 Use Cases

From a machine group point of view, the augmentations previously described allow the following questions to be answered:

- Does the machine group have a suitable amount of WIP waiting/processing? What share of the total machine group's WIP is related to the currently observed flow / operation? This is especially useful for bottlenecks, where a lack of WIP and running tools empty represents a capacity loss and too much WIP might lead to higher Cycle Time.
- How much WIP is expected to be arriving at this machine group and where is it coming from? Is the upcoming WIP hindered by some line hold at a predecessor operation?
- What are the details of the material currently standing in front of a machine group, including dispatch ranking and dedications?
- Into which chains (to which machine groups) will material be fed based on the current dedication setup and tool situation?
- Is the productivity of the tools sufficient (enough productivity/standby time)? Are there enough tools assigned / released?

Also, because of the complexity folding, it is easy to get an overview of how much material is in which layer. The user can then drill down to operation level for desired layers (e.g. where there is a lot of WIP).

Finally, the approach is valuable for visually comparing flows or flow sections against each other. An example of this feature has already been shown in Figure 4, where one operation requires different machine groups for the individual flows, whereas the rest of the flows are similar. The graph will automatically be combined where the flows are equal and split where they differ. This allows to comfortably compare flows on layer or process step level.

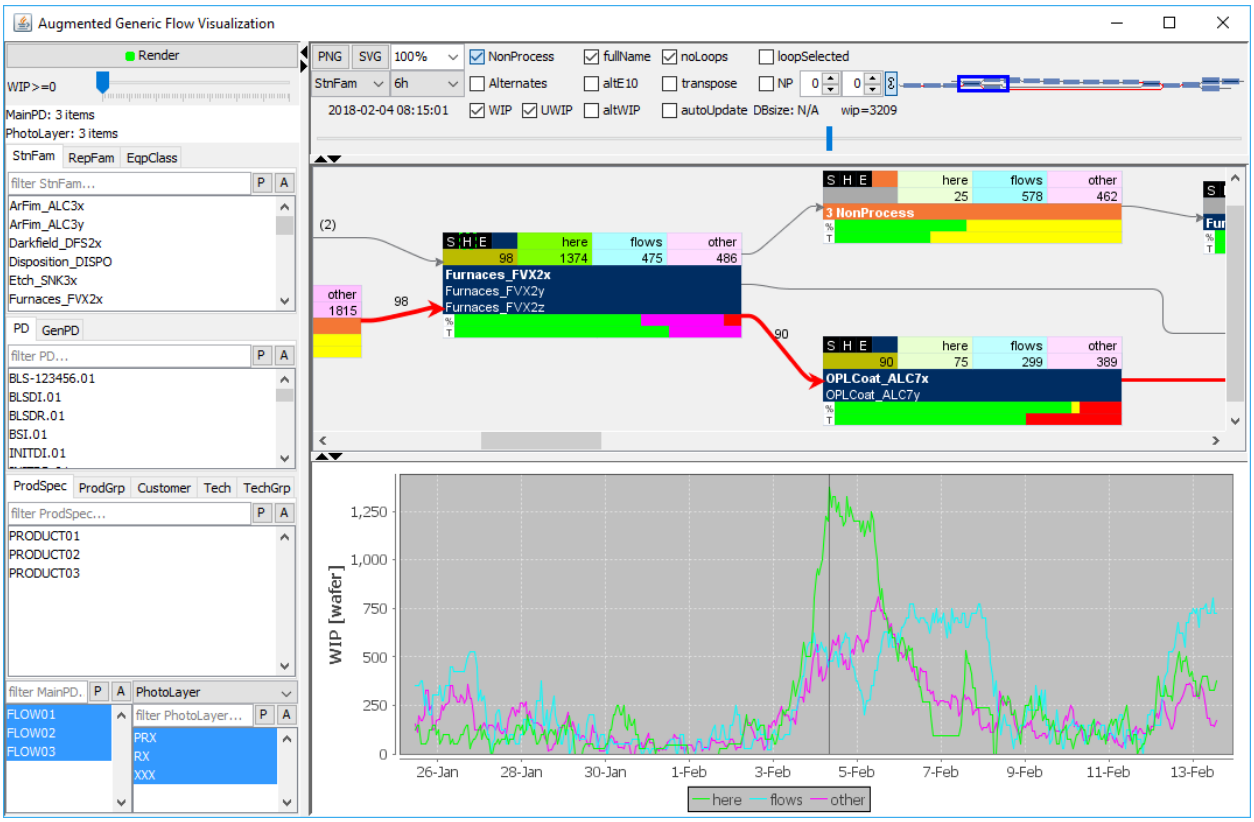


Figure 6: WIP history chart with highest peak selected by time slider.

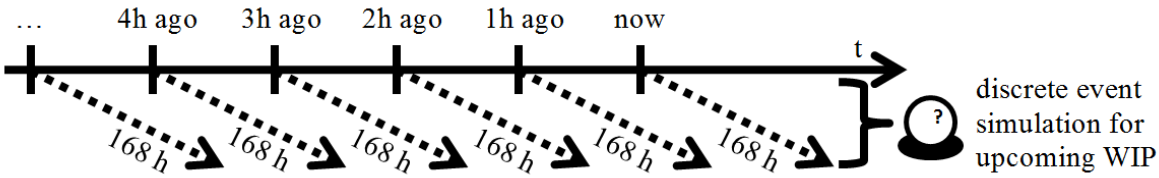


Figure 7: Integration of simulation into time slider for hourly upcoming WIP forecast.

4 SOFTWARE AND SYSTEM ARCHITECTURE

We start by describing the architecture of the underlying software, its scalability and how it complies with the company's regulations. While the software was created as a customized in-house development specifically meeting the company's needs, its core and user interface are well-structured and highly modularized, allowing to be easily adapted for other data models or different use cases.

4.1 Architecture

As described in Section 2, utilizing an existing powerful graph rendering framework avoids being detracted from the actual problem by having to implement such underlying functionality ourselves. Therefore, we researched and tested numerous frameworks and found JGraphX (Bagger and Heinz 2001; JGraph 2017) to work best for us. It is licensed under the permissive 3-clause BSD license and provides fully interactive and customizable graph rendering including a number of sophisticated automated graph layouting algorithms. For our purposes, we found the Hierarchical Layout to work best.

As JGraphX is written in Java, a language which the authors are generally familiar with, it seemed logical to use it for implementing the application. Also, JGraphX integrates well into Java Swing, which was chosen as a portable, powerful, highly adaptable yet relatively simple to use User Interface Framework, allowing to freely integrate additional components like the line chart of JFreeChart shown in Figure 4. This makes the application very flexible for changes and adoption of additional use cases.

In order to be used on any computer without requiring an explicit installation, the application is embeddable into a web page as a Java Applet. We have chosen to deploy it on a dedicated JBoss (Fleury and Reverbel 2003) server, offering:

- Middleware: by choosing RESTEasy as a reliable communication framework, the applet can easily communicate with the JBoss server via secured HTTP. The server will respond only to known queries of known users (whitelist approach). Requests may be logged. Only the JBoss server has access to the database server. This greatly enhances security in comparison to direct database connections from clients.
- Stateless communication: Holding the session state in the client applet only, the server is stateless, responding directly to each query without having to consider a context. This makes the server simple and robust, i.e. there are potentially fewer possibilities to end up in an undefined state.
- Isolation: Even though our implementation has been thoroughly tested, there is still a risk of an unknown bug crashing the application. We generally expect it to be sufficient to reload the applet on the client side, as the state is held only therein. If, however, the server crashes for any unexpected reason, it could be rebooted without influencing further services. This could be automated by a watchdog.
- Database connection pooling: The JBoss server automatically reuses database connections in a smart way. It will open only a reasonable and controllable number of connections. For example, ten database connections may be sufficient for serving dozens of clients, as they do not query data permanently. If the JBoss server reaches the maximum number of allowed database connections and further queries arrive, it will queue them, serving them one after another instead of flooding the database server with too much load at once.

By choosing a three-tier architecture utilizing a clustered Oracle Database and multiple JBoss Servers with Load Balancing as well as doing the computationally most expensive rendering work on the client machines, we get a scalable and fail-safe system architecture potentially usable by hundreds of users at once (Figure 8).

Apart from the disadvantages mentioned above, using a stand-alone Java Client directly accessing the database would also be undesired by corporate policy. Large companies often have regulations regarding programming language, user security (e.g. requiring user single sign-on), fail-safe mechanisms, scalability and maintainability of the system and whether executables are allowed to be deployed on individual client computers. The described architecture fulfills all requirements of GLOBALFOUNDRIES. We had no significant failure of the system and required no considerable user support or maintenance effort since the first deployment five years ago. Currently, we have more than 200 trained users, with 10 to 30 distinct users using this software every day. Depending on where in the production line problems occur, these are different users on a daily basis.

The simulation is loosely coupled solely via a single text file listing the amount of upcoming WIP for each product, customer, flow, operation and time frame. This allows the simulation framework, methodology, level of detail as well as time horizon to be freely chosen. At the moment, we use a non-deterministic discrete event simulation with simplified dispatch rules for the next 168 hours, but this may be changed at any time.

4.2 Usability, speed and scalability

The described architecture and used framework is able to draw flow charts that include all flows of a few hundred products at the same time if desired, as shown in Figure 9. Very complex charts like that take at maximum a few seconds to render. After the user has selected more specifically what he wants to see, the chart is regularly drawn within < 0.1s.

There are four kinds of data that are retrieved:

- All flow related data is loaded when the application (browser window) is opened.
- The latest snapshot with WIP levels, E10 states and lineholds is loaded afterwards. When switching to a different snapshot it takes around 5s to retrieve the new data from the database.
- Detailed WIP information is loaded from the database when a WIP bucket is clicked. This takes <1s.
- WIP charts over time are loaded upon request and take ~5s. This takes so long as data loaded needs to be on process step detail level to be able to exactly consider the filter setting chosen by the user.

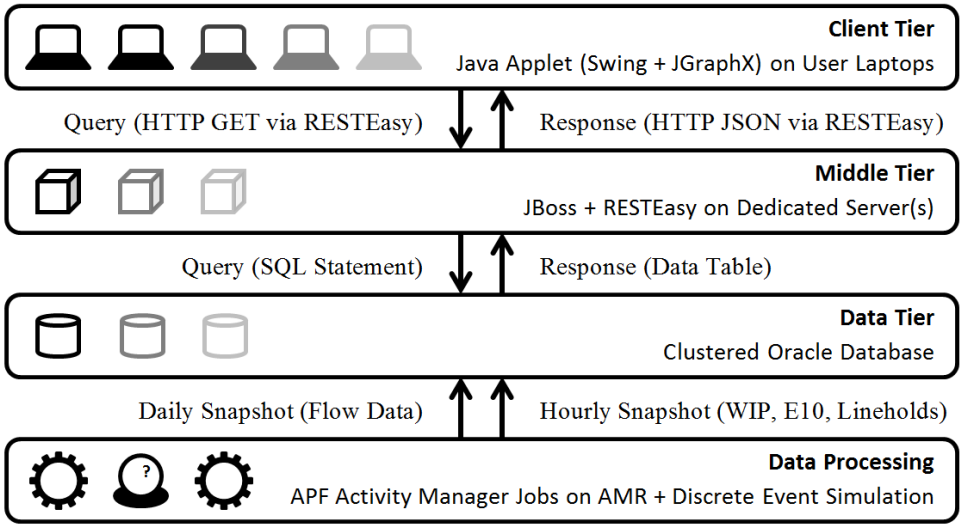


Figure 8: Scalable System Architecture.

Another advantage of the chosen architecture is that everything is rendered on the user's computer. This allows fast response to any user input on the one hand and on the other hand also scalability. If a single server needed to render all requested visualizations, the maximum amount of users would be limited severely. Utilizing the client computer allows fast responses while still being able to provide hourly snapshots of the last two weeks.

Finally, we hold a total of ~50GB of data in the database. Besides the current flow setup it contains hourly snapshots of all lots, lineholds, upcoming WIP expected by the simulation, etc. for a total horizon of two weeks. The described architecture is able to hold the data currently relevant for the user inside the Applet with a total of ~0.5GB memory consumption on the client side, whereas all other data is loaded upon request.

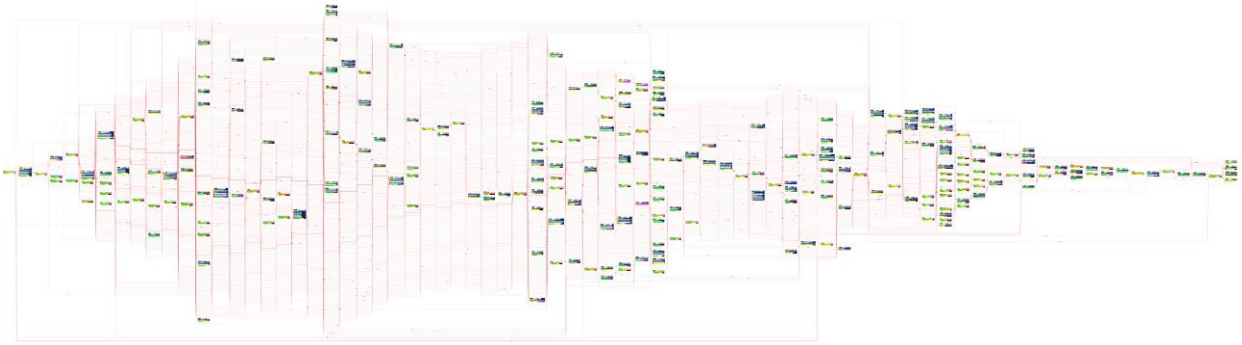


Figure 9: Complex graph of a many flows selected at the same time.

5 SUMMARY AND OUTLOOK

The described software enables the flow-related visualization of the shop floor. This enables the simulation expert to see the behaviour of the simulation model along the production flows and directly analyze the root cause for disturbances and WIP pile-up. Furthermore, it allows to analyze the situation in the fab as current state but also to analyze situations in the past.

The described architecture is suitable to handle a lot of data, even in a very complex foundry with hundreds of different products. Furthermore, it fulfills common requirements of a company's IT environment including user single sign-on and other security concerns.

One thing we are thinking about for the future is to extend the time slider to show the anticipated situation in the future based on the current state. This will allow a more detailed analysis of the simulation model (see Figure 10 in comparison to Figure 7).

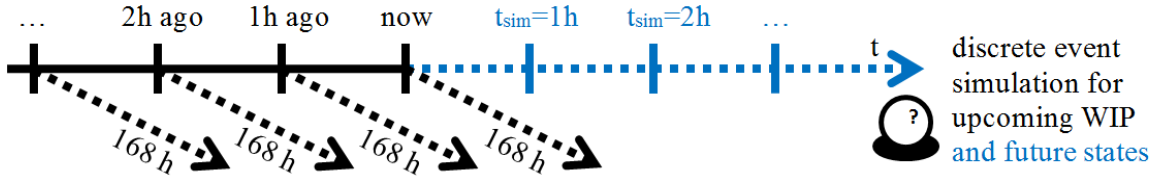


Figure 10: Possible time slider extension to show simulated future situation.

In addition, we are thinking about direct simulation model modification through the user interface. If an obstacle like an inhibit is identified as not persisting any longer, it could be disabled and the change fed into the next simulation model run. The JBoss architecture allows to feed back data into a database and the simulation software could use it.

REFERENCES

Bagga, J and A. Heinz. 2001. "JGraph – A Java based System for Drawing Graphs and Running Graph Algorithms." In *Proceedings of the International Symposium on Graph drawing*, edited by P. Mutzel et al., 459-460. Heidelberg, Germany: Springer.

Chen, T., Y. Wang, and H. Tsai. 2008. "Lot cycle time prediction in a ramping-up semiconductor manufacturing factory with a SOM–FBPN-ensemble approach with multiple buckets and partial normalization." In *Proceedings of The International Journal of Advanced Manufacturing Technology*, 42(11), edited by A. Y. C. Nee et al., 1206-1216.

Cimino A, F. Longo, and G. Mirabelli. 2010. "A General Simulation Framework for Supply Chain Modeling: State of the Art and Case Study." *International Journal of Computer Science Issues* 7(2), 1-9.

Fleury, M. and F. Reverbel. 2003. "The JBoss Extensible Server". In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, 344–373. New York, USA: Springer.

JGraph Ltd. 2017. "JGraphX (JGraph 6) User Manual", https://jgraph.github.io/mxgraph, accessed February 10th, 2018.

Niedermayer, H. and O. Rose. 2003. "A Simulation-based Analysis of the Cycle Time of Cluster Tools in Semiconductor Manufacturing". In *Proceedings of the 15th European Simulation Symposium*, edited by A. Verbraeck et al., Delft, Nederlands, ESS.

Reynolds, J. 1997. *A Hierarchical Layout Algorithm for Drawing Directed Graphs*. Master dissertation, Department of Computer and Information Science, Queen's University, Kingston, Ontario, Canada.

Tušla, T. 2017. *Layout of hierarchical flow charts*. Ph.D. thesis, Department of Computer Graphics and Interaction, Czech Technical University, Prague, Czech Republic.

Winkler, T., P. Barthel and R. Sprenger. 2016. "Modeling of Complex Decision Making using Forward Simulation". In *Proceedings of the 2016 Winter Simulation Conference*, edited by T. Roeder et al., 2982–2991. Piscataway, New Jersey: IEEE.

AUTHOR BIOGRAPHIES

THOMAS WINKLER is an engineer in the Line Analytics department (part of Industrial Engineering) of GLOBALFOUNDRIES Fab 1 in Dresden. He received a master's degree in Applied Information Technologies at Dresden University of Applied Sciences. His research interests include capacity planning, heuristics for multiple-constraint problems, data visualization and simulation. His email address is thomas.winkler@globalfoundries.com.

RALF SPRENGER is manager of the Line Analytics department (part of Industrial Engineering) of GLOBALFOUNDRIES Fab 1 in Dresden. He received a Ph.D. from the Department of Mathematics and Computer Science at the University of Hagen, and a master's degree in computer science at Dresden University of Technology. His research interests include industrial engineering in semiconductor manufacturing and optimization. His email address is ralf.sprenger@globalfoundries.com.