The Role of Models in Semiconductor Smart Manufacturing

Alan Weber, Cimetrix Incorporated

Good afternoon. The reason I picked the title for this talk is that we've seen so many references to Smart Manufacturing, not only in this conference but in many leading up to it. Industry 4.0, IoT, all of these things seem to be swimming together in one collection of vocabulary, so one of the key points I wanted to make, and have made in a variety of recent presentations, is that semiconductor manufacturing has been an example of a "smart" domain for some time already. I will also explain how the models that are an inherent part of our latest integration standards bring us even closer to the full vision of Smart Manufacturing.

First of all, I'll pick a familiar definition for "Smart Manufacturing" rather than making up one of my own. I'll talk about how the SEMI Standards have evolved over time to support the needs of our process control and other operations management needs continuously since their inception, as this has been a 30-year evolution. Then I'll talk about some actual current examples of equipment models that are embedded in the standards, and give some application use cases that depend on those.

The key point is that with the sophistication and level of detail of the models that are now embedded in the industry standards, if your equipment is compliant to these standards , there is a great deal of application capability that can developed in a truly equipment- and process-independent way. Now when you get into fault detection and control algorithms that depend on specific process parameters, of course those go beyond the scope of the standard embedded model. However, a great deal of the events and states and parameters that are required do to the data framing necessary for feature extraction for these kinds of applications are all very much standard if the latest versions of the standard (especially SEMI E164 – EDA Common Metadata) have been adopted. OK, so that was the conclusion first…

From the Industry 4.0 Wikipedia page, “Smart Manufacturing” is defined as cyber-physical systems that create virtual copies… yada, yada, yada. The point is, that over the Internet of Things, these cyber-physical systems communicate and cooperate with one another to help achieve the factory objectives in real time, and we’ve seen these examples of that in most of the presentations, especially the factory-oriented presentations this week.

As a point of context, we’re in the connectivity business—Cimetrix doesn’t sell much directly to the fabs. We have our software at most production factories in the world, and we are quite happy if you don’t even know that. It’s like the plumbing in your house: you flush the toilet, you expect everything to move on down the line. The same thing is true with the data in these systems: you expect our software to let data come from the equipment and sensors into the servers and systems that require it. If we do anything to get in the way, that’s a bad thing.
In the connectivity business we see everything from that perspective.

So… in a future smart manufacturing environment, what are some of the fundamental requirements for all the “things” that will be collaborating? I’ve suggested a list of attributes that these “things” might need to have.

First of all, there is no way in the world you can maintain a static list of the thousands of these sensors and other things in a manufacturing environment. If it was correct in one second, it would be wrong a minute later. And so having it be *discoverable* somehow is the first key attribute. Secondly, it must be *autonomous*, because if would be impossible to implement a rigid command and control network for a system that included three thousand devices. So they need to be autonomous, while working within a set of guidelines that enable them to effectively collaborate. The point I’m emphasizing most in this particular presentation is that they should be *model-based*. This means that there should be some explicit description of their content, structure, behavior… all of those things you need to know to actually interact with them effectively. This should be expressed in some sort of explicit form, ideally a *standards-based* model. Of course, *communicative*, that goes without saying, since things can’t collaborate if they can't communicate. *Self-monitoring* – these things will have intelligence. They should know when they're healthy; they should know when they're not, and tell you about it. Again, there is no way in the world we'll have enough time and software to be monitoring the behavior of every one of these things. It would be better if they did that themselves, and raise the alarm that says "Hey, I'm not feeling so well; maybe you can take me off-line and put a replacement in until I'm better."

And then finally, *secure* is key. With all of this collaboration going on, there is certainly opportunity for malicious actors to be involved in all this, and so security goes without saying in this set of attributes.

Now, if you have a set of devices that meet these criteria, you can imagine what kind of collaborative behavior could emerge with a system made up of these things.

Let's cover the key messages here. First of all, models, as I've hinted already, are very useful things because they help you understand equipment and process and component behavior when you apply them to smaller things. They also allow factories and suppliers to communicate with one another, because, unlike the specifications which may have all sorts of interpretation issues, explicit models are just that—explicit—and should be able to unambiguously describe what something does. Actually, that is making my second point: explicit models, especially standard ones, are particularly useful, because they allow plug-and-play applications to exist, once you understand what the model is capable of.
The importance of events is my third key message. There was a slide in Gerhard Luhn's (from Systema) presentation in which he stated that the natural world is based on events. I mean, we are event-driven people—things happen and we react. No matter how good we thought our plan was, things always conspire to put us off course, and so being easily reactive to all the events that happen is a good thing. There is a lot of equipment event data available that is not being effectively used, which represents a lot of untapped potential in our manufacturing environments.

This is especially true since *time* is the one thing that we all share. We all get the same amount of it, and when it's gone, it's gone. You can never recover production time that you didn't take advantage of; you can't get 25 hours out of tomorrow, no matter how much some of your managers might expect.

And finally, models are the basis for true component interoperability. This gets back to that collaborative aspect. Roughly 20 years ago, we put a proposal together for an "agent-based manufacturing system," back when agent technology was being discussed in the late 1990's. One could imagine collaborative networks of things that exhibited "flocking" behavior and other emergent behaviors. There was a lot hype at the time, but the basic idea was that every "agent" (thing) had an objective and some basic interfaces. And if you put them together in the right way, they should be able to exhibit intelligent collaborative behavior. In today's world, explicit models are one of the ways of achieving this.

Note that this is not a new concept. There have been models in the semiconductor integration standards for a long time, even as far back as 30 years ago when SECS-I was first defined for basic messaging. Admittedly, these first models weren't very elegant. If you look at the natural language analogy, all we talked about at that level were data items. All we did was to agree on the format of data items—tantamount to providing a dictionary. This wasn't *really* useful until you could actually put data items together in some sort of grammar, which is where the SECS-II language came from.

However, until GEM came along in the early 90s, there was a dialect of SECS-II for every major semiconductor manufacturer, and the equipment suppliers had to provide a version of their interface software for Hitachi, a version for TI, a version for IBM, a version for Intel, and so on—it was just cacophony. And so the major end users got together in the early 90s and said "Let's define the sentences that we can speak across this language, and they were called 'capabilities' in the GEM parlance." Now ironically, although GEM stands for Generic Equipment Model, there is no explicit model in that document. It was basically just a list of capabilities. The authors actually had a mental model in their minds, but they never wrote it down. You had to infer that from the description of the capabilities.

Next GEM300 came along during the transition to 300mm manufacturing as the industry recognized that to support the kind of automation required by that technology node, it would have to get much more explicit about the automation sequences and how the state machines and events and all these capabilities weave together, and so they defined entire scenarios that were like conversations that would achieve a particular purpose. GEM300 has carried us for a long time, and to this day, supports 300mm production factories around the world.

In the mid-2000s, along comes EDA, because although the GEM300 interfaces were ideal for what they were intended for, once they are locked down, people are nervous about trying to pump too much data through them. This limited what we were able to do from a data collection standpoint. As a result, EDA was defined in the mid-2000s as a way of having much higher performance, dynamic data collection. However, even though the EDA standards feature an embedded equipment model, without any prescription for what was in those models, you again had the same diversity of interfaces that you had way back in the SECS-II era. This situation was reminiscent of Improv Theater. Now, I don’t know if any of you have been to Improv Theater, but it’s a pretty tricky thing for people to do, and there are a lot of good examples of *bad* Improv Theater. You give somebody a theme, and then they just seem to wander off the planet.

Now, when E164, the standard for EDA Common Metadata, came along to set guidelines for equipment metadata model structure and content, it was like Improv Theater with a point. They basically said “let’s actually agree on what the content of these models will be, on the naming conventions and the structures so that you can actually have true collaborative behavior between factory applications on the tool data.” And it’s anybody’s guess where this will go in the future… But for the time being, we have enough on our hands to fully adopt E164, which is the most recent stage of our connectivity standards evolution.

I'm going to spend the rest of the time talking about the models specified by E164, which is the SEMI specification for EDA Common Metadata. Now for you circuit design people who hear "EDA" and think "Electronic Design Automation," you'll just have to forget that association for the moment; in the context of SEMI Information and Control systems, EDA stands for "Equipment Data Acquisition." It is admittedly a poor choice of acronym, but it is what it is…

This is what an EDA model looks like. Note that it is formed using only a handful of different component types on the left side of this diagram, and they can be nested in a whole variety of ways. Now if you go look at the actual information model in the E120 standard, you will see that each of these component types can be nested within itself and one another (except for the equipment node), which means that the resulting models may be arbitrarily simple or complex. Notice also that you can decorate each one of those components with all of this other stuff on the right side of the diagram, so each node description may include state machines, parameters, simple events, exceptions, etc. Consequently, this standard structure allows you to have a really flat, ugly, impossible-to-use model, or a really deep, ugly, hierarchical model, or anything in between. To improve this situation, E164 came along to say "Let's agree on how these things should be structured, how they should be documented, what they should be named—for all the things that are common in our equipment." And so, that's where the E164 standard came from. I'll leave is as an exercise to the reader to go figure out what the components of an E164 model might look like…

Now, for tools that follow the E164 standard, it is really easy for you to upload a model and browse it, since you know ahead of time where to look for stuff. For example, if you need to know where the substrate processing location for this chamber is, you can easily find it, because it's always going to be appear at the third level underneath a component called a processing module. So it makes it very easy to discover what's available in that equipment.

Looking at the overall equipment model value chain diagram, it's important to note that it is the tool supplier on the left who defines what's in the model, *not* the end user over here on the right that is concerned with manufacturing KPI's. So unless you as an end user are fairly prescriptive about not only the list of standards you want, but the list of things you want in that model beyond what is standardized, you'll never get that information out of the tool.
It pays to pay attention to that very carefully at the outset – all the way down to the process parameter vectors that you expect to support various kinds of process control and equipment engineering applications that are important in running your factory. So there a lot that the standards do dictate, but when you get into process specifics that we need for fault detection or preventive maintenance, or a whole list of other applications, those are going to be extensions to the standard.

In summary, if the model is properly defined, you can see that it supports the entire value chain, from creating the equipment, supporting the applications, the addressing the careabouts of different factories, and a wide variety of KPIs. So this model value chain is a big deal.

There are a few additional points to make about equipment models. First of all, a well-designed model will exactly reflect the tool's hardware organization, so if your equipment engineering people know how they think about the tool, the model will look pretty much the same. This includes, of course, not only the physical structure, but also a few logical components that are required by the standard. These include a Performance Tracker, a Job Manager, and a Material Manager, so as you create dynamic (aka "transient") objects to represent the carriers and substrates that enter the system, and the process and control jobs associated with that material, you have standard places in the model to find those items.

Secondly, the model is not described separately in some data base that you have to find, or in a document that's buried on an integration engineer's desk… the models are resident in the equipment, and the standards define the mechanisms by which you request and receive that model. Therefore, it's always available—there's no additional documentation required to find it. It also serves as a good common point of reference among all the stakeholders that care about that equipment's data.

For those of you who want to structure your SQL data bases according to the structure of the equipment model, it could also be used as a way of auto-configuring a lot of the records and "tags" in your process data bases. And finally, to the degree to which your applications depend only on the standard content, they can be truly plug-and-play across a wide range of equipment types. So what I'm going to do now is cover a number of the factory applications that could truly be plug-and-play.

If you look at this list of applications, you can see that OEE calculation, substrate tracking, process module execution, and even lot completion estimation (which is becoming more and more important as inter-process times become an issue) could be done in a generic way. All four of these applications, in fact, even product time measurement (or Wait-Time Waste, as it has been called), can be driven entirely by the events, parameters, state machines, and other information that is entirely prescribed by the E164 model. Now the only partial exception to this statement is that there's some information in OEE that doesn't come directly from the equipment—namely, the *reason* something is waiting—and sometimes that may require operator input of data from other place. However, for the most part, knowing where substrates have been, knowing what's happening in the process module, knowing when the set of wafers currently on a given machine is going to complete and will be available for pickup—all of these things can be calculated, tracked and predicted, given the information in the equipment models.

Now let me illustrate this with a couple of examples.

For those of you familiar with the SEMI GEM300 standards, here are the state models for the two principal objects in E90, which is the Substrate Tracking standard. It has an object defined for a Substrate, which has a whole list of attributes, and a Substrate Location, which is much simpler. You can see in the diagram that the state model for the substrate is actually two parallel state models, one that deals with transportation (on the left), and another that deals with processing (on the right). Likewise, the Substrate Location state model is also part of the standard. It's far simpler because there are only two states it can be in: "occupied" or "unoccupied." What's key to understand is that every module in the equipment that either processes wafers or can store a wafer temporarily must have, according to the standard, a Substrate Location which will have these exact two states in its associated state model.

If you go look into a graphical representation of the EDA model, you'll see something like these treeviews. This happens to be Cimetrix' way of representing it, but many others are possible. The orange icons signify events, the blue icons are state machines, and the sort of greenish-red things are parameters. If you look at the two of the key events for substrate processing, knowing that a substrate "NEEDS PROCESSING" and is transitioning to being "IN PROCESS" is a key event; knowing when it's done ("IN PROCESS" to "PROCESSING COMPLETE") is likewise a key event. Notice that you can go find these specific events, named exactly the same as called for in the GEM300 standards, over in this model. Now every time one of these events fires, the E90 standard also requires that this entire list of context parameters be available (i.e., they have valid values) for collection. I know it's hard to read, but the list includes everything from the recipe that's being run, the lot it is part of, the substrate ID, and a lot of other stuff. All of this information is supposed to be available every time that event fires, so that, depending on what your application is doing, your data collection plans can be triggered by these events and include as much or as little of this information as needed for your application.

From the substrate transportation standpoint, the same thing happens. When a wafer arrives, this first event fires; when it leaves, the next event fires; and every time that happens, you can get the parameters indicated. It is not particularly complex—you just want to know which wafer it was, and which location it occupied/vacated. So you can see that this information allows you to completely track where a wafer has moved inside a piece of equipment.

Now if you also want to know what's happening to them, then you need to go to a different state machine. That state machine is the E157 Process Module Tracking state machine, and it's also fairly simple. A Process Module is either executing or it's not. And when it's executing, it cycles between a pair of states every time a recipe step changes. Note that this is *precisely* the kind of data that you need to know in order to "frame" the data for many FDC applications, and many predictive maintenance applications. Yesterday a number of the talks discussed "feature extraction" using curves of trace data in specific regions that have been either manually or automatically determined. The boundaries of each of these regions usually correspond with those of a specific recipe step, so the conditional triggering feature of the EDA standard allows you to pick the appropriate event, and then decide how many milliseconds you want to wait before the trace data collection starts, and how long you want it to run and/or what other event should terminate the sampling.

From the models in an EDA interface, you have all sorts of event and context information available to do very precise data framing, and can use this to design different DCPs (data collection plans) that sample, say, at a high frequency during critical process steps, and sample at lower rates during non-critical steps.

Now let's look at the details of the E157 state model. An instance of this state model is running for each process module while the equipment is active, and regardless of the tool type, the same four events are found over here with the same identical names, along with the context information you probably care most about for a process tracking application. At a minimum, this would include the step count (i.e., which recipe step am I on) and which substrate am I working on.

To this point, I've only discussed *tracking* applications. This is important to understand, because if you want to know what your average processing time is per recipe, per product, per substrate, per chamber, per whatever, you can calculate every bit of that in a standard, generic application from the data that I've shown you thus far.

But now we also want to move into the realm of prediction… For example, I may need to know when this lot is going to be done, because I have to schedule a carrier to pick it up and deliver it to the next process tool in less than two minutes. This application is not just important for operational reasons; it's important for quality reasons as well, because wafers in some of their states will deteriorate if they're exposed to the atmosphere for very long. Now, the way this is handled in most advanced wafer fab is you look at the equipment automation requirements spec and they'll say "well I want this, this, and this" and there's going to be this little section that says "special variables" and "special events." There'll be things in there like "I want an SVID that tells me how many minutes are left for this lot... and another that indicates how many minutes are left until the entire tool is free, so I can be putting that information into my scheduling and dispatching systems." Moreover, I may want an event that I can configure that says "When you [the tool] cross a ten-minute threshold, a five-minute threshold, a two-minute threshold, in case I haven't already scheduled that material, I want you to wake me up, send me an event that says 'hey this thing is about done.'" And so this kind of stuff is buried in the automation specs of a lot of these advanced users.

Implementing this usually falls to the tool suppliers who say "Gosh, that is not just a standard GEM Interface… I've got to do some custom calculations…" And once the tool is delivered and the algorithm changes, the chip maker has to go back to the tool supplier for a software update, unless they were clever enough to make it configurable. In contrast, with these sort of standards that include a rich equipment model, all that prediction can be done with the information that is generally available.

As an example, here is a summary of the algorithm. I'll leave the detailed execution of this to your imagination, but basically you want to know how many wafers there are in the lot, what the average processing time is for each of them, and how many are left to process so you can update the completion prediction. All of that information can be calculated by looking into the model. Knowing 1) which Process Job IDs are there, 2) how many wafers are in each job, and then 3) tracking the "AtWork to AtDestination" events for each of the wafers, and 4) keeping a running total of that time difference and 5) the average of those differences, you can estimate how much time you've got left. This algorithm can be completely implemented by first looking in the model, opening the MaterialManager—which is the logical component where all the information about substrates, wafers, and carriers is—finding the ObjID attribute from the carrier you're working on, and then finding out how many Process Jobs are associated with that object, and how many wafers are in each of those. Then you start watching these E90 Substrate Transport events to see when every wafer arrives at its destination, subtracting the previous wafer's arrival time from this one, summing all that up, and updating the average. Unfortunately the algorithm goes back and forth from one slide to another, so I'm not going to go through all of that, but when you look carefully at the presentation later, you can see where those little arrows go and understand exactly what parameters and events in the equipment model are being referenced.

A similar algorithm can be used for Wait-Time Waste analysis, and you don't actually need to do much prediction in this case. This is an example that came from NXP, who did a whole lot of calculation of exactly where each wafer was in a complex lithography track and scanner combination. Using only four event types—when each wafer arrives and departs, and when each process starts and stops—they were able to create this beautiful stacked Gantt chart. Notice that there is a grey area in the middle when the scanner was actually doing nothing. What NXP originally thought was an operator loading problem—putting material on the tool at the wrong time and leaving the tool a little bit starved—was actually an internal scheduling issue with the scanner. And NXP would never have figured this out without looking at the detailed events that drove that.

Another key characteristic of all these applications is that it's not an "all or nothing" thing. The more data you choose to collect, the more benefit you may realize. Moreover, it's not linear:

you may well get 80% of the benefit from 20% of the data. So it's important to first look at what you're trying to accomplish. Start with the decision you need to make, determine what insights are needed to drive those decisions, identify the data required to achieve those insights and the source(s) of that data. This basic process works for any application. And these standards don't suggest that you should collect and store everything; just do it on the basis of what you actually need.

I'll skip the Multivariate Fault Detection and Classification (MVA FDC) for now, except to point out that we see a lot of pictures like this (the trace diagram), and that the triggers for transitioning from one zone to the next are events buried in the equipment model that can be used to do the framing. The other key with the latest EDA standards is that the "conditional triggers" allow you not just to start a trace request on a single event, but a Boolean combination of events that can include algebraic combinations of conditions, resulting in very precise framing.

Another frequent topic in these conferences is external sensor integration. This gets a little deeper into the actual process specifics, but a key insight in the sensor integration domain is that the biggest part of the problem is *not* finding a sensor that works, it's all this other stuff: sampling synchronization, dealing with multiple timestamps, scaling and units conversion, and merging data into a database… The architecture I suggest using the EDA standards allows you to do a lot of that very close to the tool by using a shared model for the process tool and the sensor integration server to make the data look like it was collected from the same place, even though the sources may have been very disparate. So again, I leave the details as an exercise for the future…

If you go to our website, you'll see that we explain a lot about these applications with videos and app notes, but I especially wanted to highlight the breadth of things you could be doing with these standards *today*. Models have been at the core of our standards for decades, but they'll really help components of future manufacturing systems understand one another, and will play an increasingly important role as the number and variety of the devices we put into our factories increase.

So, that’s what I’ve got today. Please contact us with any questions or to discuss these ideas further.