Increasing Productivity at Saturn

Automaker Saturn drew from academic research to integrate information across disparate systems within its manufacturing site. The new information system gave team members across the plant access to real-time and historical throughput data. This information helped Saturn identify changes that led to a 10 percent increase in plant throughput.

To remain competitive, manufacturing enterprises must increase throughput and simultaneously reduce costs. This requires daily and long-term examination and analysis of a plant’s functions and operations. Using this data, an enterprise can identify production flow bottlenecks and analyze capacity and other factors, which in turn helps identify improvement opportunities. While such information is critical, it is often a challenge to obtain in a complex manufacturing environment.

The automotive industry provides some prime examples of complex manufacturing environments. At Saturn Corp., we turned to academic research—work involving model-integrated computing (MIC)1—as a framework to organize the diverse types of data our information system must deal with. In MIC, you build domain-specific models to capture information relevant to a system under design. Tools developed to work with MIC then automatically and quickly generate applications.

Because MIC depends on domain-specific models, we must first explain some basics about the process Saturn uses to build cars.

THE BUSINESS OF BUILDING CARS

Manufacturing automobiles involves many disparate operations: stamping, molding, fabrication, casting, machining, assembly, and so on. Saturn brings these operations together in an integrated manufacturing system designed around just-in-time principles.

The Saturn manufacturing site, in essence, is a network of processes and buffers. Processes represent the operations required to build a car. An operation, in turn, can involve actions (such as the casting, machining, and welding of car parts) or assemblies (such as for transmissions, engines, and the final car). Each process has associated measurements that indicate its productivity. These measurements—critical information in making business decisions—include entities such as cycle time, production count, work in process, and downtime.

Buffers (or banks) lie between processes. A buffer holds parts and/or subassemblies produced by an upstream process for consumption by a downstream process. In different sections of the plant, buffers take on different physical forms: They may be parts contained in portable containers or on conveyors, for example. Despite their different physical manifestations, buffers have common measurements that are pertinent to production:

- bank count (the number of parts or subassemblies in the buffer), and
- minimum and maximum buffer capacities.

The interconnectivity of processes and buffers captures the sequence of operations required to produce a car. It also defines the interdependence of processes on each other and on buffer capacities, and how one
We can attribute some tangible results to SSPF: Using SSPF-provided data, a throughput task force instituted changes that led to a 10 percent increase in the throughput of Saturn’s Spring Hill, Tennessee, plant in three months.

SUPPORTING SATURN’S BUSINESS PROCESS

The business process Saturn employs to continuously improve throughput consists of:

- gathering relevant plant data,
- visualizing plant operations and analyzing each operation’s capacity,
- identifying bottlenecks, and
- formulating engineering and business plans to improve throughput.

To better support that business process, Saturn designed and implemented a new production flow concept in 1993. As part of that effort, we had to select key measurements from the available data for all processes and buffers. Saturn team members identified these key indicators through experience and observation. Over the years, buffer indicators had proven extremely useful, and they received more attention than in the past.

Beginning in late 1994, Saturn initiated a manual evaluation process that involved calculating and posting stand-alone capacity: Stand-alone capacity is what a process’ production rate would be without obstructions (without starving or blocking). This became a key measurement in identifying bottlenecks for potential investment opportunities.

Saturn’s traditional information system and engineering process supported the manual update and posting of the stand-alone capacities only once a month. The system also did not make postings available to all team members and leaders who participated in business decisions. The company determined that it would take too much effort to perform this manual process more frequently.

A NEW PROCESS AND SYSTEM

To solve these problems, Saturn team members and Vanderbilt University researchers began collaborating on the Saturn Site Production Flow (SSPF) engineering process in 1995. The primary focus of SSPF is to provide tools and services that help improve production flow. SSPF was a direct request from Saturn’s manufacturing leader to address timeliness, automatic data collection, and universal accessibility.

SSPF provides an integrated problem-solving environment that presents consistent and pertinent information. It also provides analysis and decision support services to team members and leaders. The SSPF engineering process is supported by an information system called the SSPF system. In this article, SSPF refers to the engineering process as well as the system.

The SSPF engineering process makes real-time and historical data from across the site available to all team members and leaders. In addition, the SSPF system calculates and updates stand-alone capacities, production rates, work in progress, blocking and starving information, and other measurements in real time.

We can attribute some tangible results to SSPF: Using SSPF-provided data, a throughput task force instituted changes that led to a 10 percent increase in the throughput of Saturn’s Spring Hill, Tennessee, plant in a three-month period. The responsiveness of SSPF to changing business needs, engineering requirements, and the physical environment has provided a significant degree of flexibility. This flexibility permits a business process based on continuous, ongoing improvement by providing the production data to make day-to-day and long-term business decisions, a key success factor to the overall business process.

SSPF SYSTEM

The overall SSPF system consists of components in two distinct groups, as shown in Figure 1:

- Application. The software components in this group comprise what industrial engineers traditionally think of as the runtime application, such as a manufacturing execution system. These components gather, log, visualize, and analyze real-time data.
- Modeling and program synthesis environment. We use the components in this group to model plant processes and buffers and to automatically configure and synthesize components in the application group.

A set of configurable tables, explained later, provide the bridge between runtime components and models.

Application group

The client-server-based SSPF application consists of many commercial, off-the-shelf and custom components, as Figure 1 shows. These components are all highly configurable through configuration methods—
examples of which are plaintext, comma-separated value (CSV) files or, in the case of the SQL database, through SQL queries. We made the custom SSPF components as configurable as possible. The high configurability of SSPF application components is exploited in the overall SSPF system, as we will discuss later. The sidebar “SSPF Application Components” briefly describes these components.

Configuration methods

We use the generic term configuration method to refer to the external configuration information. In some cases, a programmer might perform configuration through a text file with a predetermined format. A program could then read and modify its data structures or I/O, for example, according to information in the text file.

In some cases, as with the SQL database, configuration might take the form of SQL queries that modify the database content, thus ensuring that any other component or program that accesses the database will find the right data. Or it might take the form of class diagrams, as with systems based on the Unified Modeling language. In some cases, as with the real-time data handler (RTDH) component, configuration might take place through modification of a software library (or libraries) used by the component. For example, we configure RTDH (in part) by changing an RTDH-loaded DLL responsible for transforming the raw plant data into uniform SSPF data.

Whatever the means used to configure a component, the important fact is that highly configurable components make it easier to integrate, change, maintain, and evolve an application. In fact, most software packages come with their own configuration management tools—tools with varying degrees of sophistication.

Using configuration management tools in itself does not alleviate the problems associated with large software applications. Large packaged applications still work at the level of their own configuration information (even when they use an object-oriented approach or modeling languages such as UML), irrespective of which application they are part of and which plant they are running in.

Our method of software development recognizes that configuration information is directly and completely dictated by the plant itself. Thus we take the following approach:

- Develop and use configurable custom and COTS software components.
- Identify the entities in the plant, and the relationships and properties thereof, that dictate the configurations.
- Choose a formal representation for the plant that is as close to the representation used by industrial engineers as possible.
- Develop tools to model the plant using the formal representation.
- Develop model interpreters to translate the models into the required configuration.
- Model the plant.
- Generate/configure the runtime system.

We accomplish these tasks using the Model-Integrated Program Synthesis Environment.
SSPF Application Components

The SSPF application combines off-the-shelf and custom components. The key to making these components work together is their high degree of external configurability, which is achieved through configuration methods.

Data acquisition software. SSPF uses Cimplicity, a package developed by GE Fanuc Automation, as its front-end data acquisition package.

Data acquisition interface. This component acquires the raw, real-time data from the plant via Cimplicity and passes it on to the real-time data handler.

Real-time data handler. A custom component, RTDH mainly performs two functions. First, it transforms (using unit conversion, and so on) the raw data from the plant into a uniform, homogenous set of data. Second, it multicasts the transformed data to any “clients” interested in the data. Some clients are what we commonly understand to be clients in a client-server application; others are components that comprise the server.

SQL database. SSPF uses Microsoft’s SQL/Server database to store its data.

Real-Time Data Logger (RTDL). This component is responsible for logging the real-time detailed and summarized data into the SQL database. It “listens” for the SSPF data packets multicast by RTDH, summarizes or otherwise manipulates them, and logs the data.

Historical data server. A separate server provides access to the historical data stored in the SSPF database.

Client data server. As SSPF evolved, users requested many new services. This component provides those services, which include start-up information and manual data entry, for example.

Process viewer. Also called the SSPF viewer, this component is one of the clients for SSPF data. It is used to visualize real-time and historical production flow, look at hourly trends, manually enter data, and so on.

Data browser. Another client of the SSPF system, the SSPF data browser provides structured access to any and all the data (real-time and historical) collected and stored by SSPF. It is an OLE-linking and OLE automation server, making it easy to link or import SSPF’s data into other COTS components, such as Microsoft Access, Microsoft Excel, and Web browsers.

Bottleneck analysis tool. This component (under development) observes the Saturn plant (by listening to the data multicast by RTDH) and analyzes the production flow to identify bottlenecks. It provides information to clients in real time and logs summaries of bottleneck analysis results.

What-if tool. Saturn team members use this component (under development) to evaluate the effect of changes to the plant or the production flow. This is a client of the historical SSPF data only.

Other COTS. Used in conjunction with custom SSPF components like the SSPF data browser, these components do miscellaneous tasks, such as generating charts and reports and presenting data on the Web.

Program synthesis environment

We developed the Model-Integrated Program Synthesis Environment within the framework of the Multigraph Architecture MGA. MGA has evolved over the past decade as a software framework, tool integration architecture, and infrastructure for MIC. MGA comprises generic, customizable tools for constructing domain-specific models, analyzing models, and synthesizing programs.

The main functional components of MGA are

- a metaprogrammable graphical model builder (M etaGM B),
- an object-oriented database for storing and accessing models, and
- model interpreters that generate and configure software components or translate models into the input data structures of analysis tools.

A current Vanderbilt research project has introduced a metaprogramming interface and metalevel tools in MGA. (This work is supported by the DARPA/ITO Evolutionary Design of Complex Software program.) The interface and tools help build and evolve domain-specific, model-integrated, program-synthesis environments. Using the MGA infrastructure, we took three steps to design and implement the SSPF environment.

Integrated modeling paradigm and modeling environment. First, we specify an integrated-modeling paradigm and synthesize a modeling environment. The modeling paradigm defines concepts, model-structure principles, and model integrity constraints, as represented in the MGA metalanguage. MGA metalevel translators translate these metamodels and automatically customize the graphical model builder and model database.

The SSPF modeling paradigm defines a structure for the information, a necessary requirement for synthesizing the SSPF application. The paradigm includes multiple interacting views of the production flow, organizational structure, data points, and system resources; we specify the paradigm in the MGA metalanguage. Saturn modifies SSPF models frequently and regenerates the system almost every week in response to lessons learned from analysis of the data. In contrast, the modeling paradigm changes only if the SSPF tool configuration undergoes a major upgrade due to additions or changes to modeling concepts.

Analysis tools. We must next add analysis tools to the synthesized environment. For example, the SSPF modeling environment includes a constraint manager, which checks the consistency and completeness of models against constraints represented in the modeling-paradigm specifications.

Model interpreters. Finally, we need to specify and synthesize model interpreters. Interpreters traverse the model database, collect application-relevant information, and generate configuration information (tables, or executable specifications or code) for the synthesized applications. In this step, we used the MGA metalevel tools to specify model interpreters and to translate specifications into model interpreter
code. The MGA metalevel tools also support the rapid construction of model interpreters.

The primary advantage of applying MGA in SSPF (as well as several other projects in the manufacturing, aerospace, and defense industries\textsuperscript{11}) is twofold. First, it supports the rapid change and adaptation of applications. Second, it also supports the evolution of the M Model-Integrated Program Synthesis Environment. We change SSPF applications by modifying domain-specific models using the graphical modeling tool. We change the Model-Integrated Program Synthesis Environment via the metalevel tools. Saturn team members modify the SSPF models frequently and regenerate the system almost every week. In contrast, the modeling paradigm changes only if the SSPF tool configuration undergoes a major upgrade. The success of other applications of MIC has led to the establishment of the M Model-Integrated Computing Alliance. (See \url{http://mcsl.vuse.vanderbilt.edu} for more information.)

**SSPF ENGINEERING PROCESS**

The SSPF engineering process is an ongoing, cyclical process that begins by modeling production data. Modeling production data defines what data should be collected and how it should be stored. The process then calls for visualizing and analyzing current and historical data, which supports business decisions. At this point, more data collection, visualization, and analysis might be required before changing the plant or process. If a decision changes business practices to increase throughput, such changes could affect the data model, as in the case of changes to production flow or plant configuration.

**Data modeling**

Saturn's data-rich environment is based on traditional process monitoring and control, and the Spring Hill site (consisting of four plants) monitors and controls an estimated 80,000 points of live data, including production counts, downtime, and bank counts. The majority of these data points come from programmable logic controllers (PLCs) on machines within each plant.

The traditional information system collects, logs, and presents data to users via status screens configured using the Cimplicity data acquisition and display package. This architecture has been in place since the Spring Hill plant started production in 1990. However, the enormous amount of data cannot be directly used for throughput analyses because

- only a subset of the data pertains to production flow, as the rest is used for other purposes such as control or diagnostics;
- the data must be transformed into production flow measures; and
- the engineering practices (engineering units, sampling rates, and so on) used in different sections of the plant make the data quite heterogeneous, requiring additional data transformation.

Once the system identifies, collects, and transforms the appropriate data into production flow measures, it presents the data to users, analyzes it, and logs it for later retrieval and visualization. Obviously, a (naive) brute-force method to implement these tasks, in which everything is hard-coded, is unacceptable. Such an approach would fail simply because of the enormous scale and complexity of the tasks, let alone the fact that the software system must exist in a continually changing environment and evolve with the business.

At the minimum, we need some sort of a "formal" data model that will exist outside of the software and capture the specifics of data collection, transformation, storage, and visualization. For example, a description that specifies certain tasks—how to fetch some specific bytes of data from a specific PLC, what formulas to use in transforming it, and where (on what users' screens) to display it—can be considered a data model. We can use such a model to configure and/or guide a system developed to collect and manipulate data. The type of data models employed by such a system will have a strong bearing on development, integration, and maintenance efforts as well as on the system's robustness.

Many systems fall short of delivering low-effort, low-cost solutions because they do not provide a comprehensive, plantwide framework for the data models and tasks. The data models remain fragmented, which makes them difficult to integrate and maintain, no matter how sophisticated the user interface.

In SSPF, we take an approach based on the observation that modeling the data itself is not enough. Any model must account for the context of the production data, which is the plant. That is, the model must account for a plant's manufacturing processes, buffers, and so on.

For the SSPF engineering process, we used a MIC approach to model the plant and its data. We identified the entities in the plant and their relationships and properties that have a bearing on the production flow and on the software used to measure and analyze that flow. Our modeling paradigm describes these entities and relations.

We modeled plant processes hierarchically, allowing the abstraction of relevant production flow information at higher levels in the hierarchy. Figure 2 shows the model for Vehicle Initial Build, a section of the Saturn general assembly plant. This process has six subprocesses: Cockpit 100, Cockpit Test, Hardware 200W, Hardware 200C, Hardware 200E, and...
Hardware 310. The icons for the subprocesses appear in the figure. The interface points on the icons represent conveyor systems that deliver parts or subassemblies from buffers. The buffers between processes also appear (labeled 2200A, 2200B, 300, and so on). The connectivity between buffers and processes represents the production flow.

There are many more types of models, aspects, properties, and attributes that together comprise the SSPF modeling paradigm. However, for brevity we do not describe them here.

Data acquisition, storage, and retrieval

Once modeled, the data must be collected in real time. Using models to guide data acquisition facilitates the implementation by providing a structured view of the data. We use the plant models to configure the interface to Cimplicity and, in turn, the PLCs. The use of models helps bring structure to the chaos of plant data.

Time is an essential dimension in understanding the dynamics of production flow. Some production flow dynamics remain within the bounds of a single production shift. Others involve multiple shifts, weeks, or months of history. Thus, to understand the dynamics of production flow, data collected in real time must be stored for later retrieval and analysis. The stored data contains detailed histories for every process and buffer in the plant.

To retrieve and analyze data more easily, we need to store it in a structured manner, according to the structure of the plant and the production flow, which the plant models capture. We use plant models to generate the underlying schemas for storing data, to provide data retrieval mechanisms in custom and COTS components, and to ensure interoperability.

Visualization

The throughput task force used many different types of visualization techniques to analyze throughput for both real-time and historical data. Rather than being set in stone, we expect the visualization process to evolve with time. To provide a visualization tool that presents a uniform view across the plant and also evolve easily and changes with the plant, we use the plant models to configure the visualization. The models provide the structured, uniform view of production flow that facilitates better visualization.

Saturn team members use the visualization methods daily to make decisions, so the system presents data to users in a succinct and problem-oriented manner. For example, if users wish to examine downtrend trends for a process or set of processes over a few weeks, they can readily access the information in...
visual form, using the SSPF system and tools such as Microsoft Access and Excel. Users aren’t overwhelmed by the data’s enormity and complexity. Instead, they can easily navigate to the desired data. Users need only specify the desired data for the desired processes and buffers, and the tools make it available.

**Real-time data.** For real-time visualization, the SSPF engineering process uses a custom-developed visualization tool, the SSPF viewer, and COTS tools combined with the custom-developed, interoperable SSPF data browser.

Figure 3 shows the SSPF viewer. This graphical user interface shows a production flow view, drop-down boxes, a detailed textual report, and legend boxes. In addition to the SSPF viewer, we use the SSPF data browser (in conjunction with Microsoft Excel) to provide real-time charts on the Web.

**Historical data.** The throughput task force used SSPF historical data extensively to analyze production flow. Other team members use the summary shown in Figure 4 every morning to review the previous day’s performance. In addition, SSPF generates summaries every week, three weeks, and six weeks.

**Analysis and decision support**

Decision-support tools help managers design and verify changes in a plant’s operation. Decision support may come in the form of data processing and visualization to recognize problems; bottleneck analysis to understand a problem’s cause; and prediction to assess the impact of a planned intervention. In this sense, decision-support tools must help define and execute integrated analyses.

Many systems are available for this work; however, these systems are usually isolated from each other. Many times, the plant representation each system uses is significantly different, making it difficult to use the tools together or to combine their results.

SSPF provides an integrated problem-solving environment, which allows Saturn personnel to use various tools together and make more informed business decisions. Some of these tools are Microsoft Excel, Microsoft Access, and the SSPF data browser.

**SSPF DEPLOYMENT AND OPERATION**

SSPF has been in operation since August 1996. The initial version focused on collecting data, making it available to users, and logging data into a relational database. The most successful part of the initial release was in providing the data necessary to improve throughput. As the system collected and logged data, it became practical to provide continuous and regular feedback to Saturn’s leadership, including the throughput task force.

The primary measurement the task force used to identify areas requiring specific improvement programs (activation areas) and to measure progress was stand-alone capacity.

SSPF provided this information in the form of

- a daily report to management that showed the previous day’s performance.

Figure 4. SSPF provides a daily stand-alone chart, which is available each morning to all Saturn team members. The traditional information system that SSPF replaced was able to provide this information only once a month. ACT ST is the actual straight-time production count.

Starve Loss and Block Loss are the number of cars lost to starving and blocking obstructions. Run ration compares actual capacity to daily gross capacity. Capacity is the daily gross capacity—what could be produced with no production losses. The last item, 1218 JPD, is the daily production target.
Saturn started the SSPF project in September 1995 with an engineering study. Following the study, Vanderbilt developed a prototype, which it demonstrated in December 1995. We next developed the production version (SSPF Version 1.0) and put it into operation in August 1996. As Saturn personnel used the system, lessons learned led to new and altered concepts that required many upgrades and additional functionality. To accommodate the upgrades and new functionality, we overhauled the modeling paradigm. On the basis of this new paradigm, we developed SSPF Version 2.0, which has been operational since June 1997. We learned many lessons during this time.

- Modeling the plant is a large part of the effort. This is not surprising, since we generate the application itself from the models.
- Eliminating excess data and providing only key indicators is crucial. The validity of this principle became apparent to us when we first selected the data SSPF would collect and display.
- Verifying and testing the application was easier. Before production release, the SSPF beta release was online while we built the models. Whenever we added to or changed the models in any way, we regenerated the application and tested it, which allowed us to verify the application and the models. The turn-around time was only the time required to change the models and to regenerate the configuration information. This short turn-around time clearly showed how flexible and maintainable an application becomes through the use of models.
- Handling changes in user requirements was efficient. During development, changes only required modification to one configurable component followed by a regeneration of the application. The change would then appear for all the plant’s processes and buffers. Without MIC, trying to keep an application upgrade consistent for the large number of processes and buffers would have been difficult and costly.
- Responding to changing business needs became quicker. Once Saturn team members realized how responsive SSPF was to business needs, they started to frequently change the focus of their analyses, sometimes changing the models as often as once a week. Since changing the models is equivalent—in other techniques—to changing the software system, we were effectively changing the system every week without adverse impact to SSPF’s performance or robustness.

Table 1. Cost of developing various versions of SSPF and plant modeling.

<table>
<thead>
<tr>
<th>Task</th>
<th>Effort (staff months)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype</td>
<td>4.5</td>
</tr>
<tr>
<td>Version 1.0</td>
<td>18.0</td>
</tr>
<tr>
<td>Version 2.0</td>
<td>12.0</td>
</tr>
<tr>
<td>Modeling</td>
<td>12.0</td>
</tr>
</tbody>
</table>

Cost
Table 1 shows the cost of developing the two versions of SSPF.
At first glance, the effort saved may not seem significant. However, a more detailed analysis gives a better picture.

- The costs include the development not just of basic SSPF functionality but also of the many extensions and additions.
- A large part of the effort went toward modeling. Since models are highly reusable, the current and future cost savings are considerable.
- SSPF offers a high degree of end-user programmability, eliminating the need for highly paid software experts.
- The biggest cost savings from SSPF accrue from the fact that using models produces highly integrated software. This results in large cost savings in software maintenance, upgrades, and, most importantly, in software integration, as demonstrated by our experiences at a different manufacturing site.

Proof of concept
In September 1997, Saturn installed SSPF at another plant. The pilot installation modeled 45 processes and their buffers, and it took four staff weeks to complete this modeling and some minor changes in the SSPF runtime system. Software support personnel made only one trip to this plant to install the system.

It took three hours to install the system and bring it up. From that point onward, real-time data and historical data were available, which enabled team members at the plant to visualize and analyze the production flow. Since then, a total of about 100 processes have been modeled in about six staff weeks of effort. This effort did not require changes to the SSPF software or the services of software experts. This experience clearly shows the benefits, usefulness, and suitability of MIC for large-scale information systems.
IC technology provides a conceptually strong solution framework for the SSPF application. Considering the diversity of requirements and data sources, logic complexity would have been high with a traditional code solution because the solution environment is quite chaotic. Traditional code solutions would require a significant effort to order information from these data sources.

Through models, we can readily explain such a complex data structure in a virtual view of the manufacturing-plant floor. Diverse services are required for applications such as process monitoring and control, process simulation, statistical analysis packages, and other data manipulation tools. MIC integrates these tools into a common problem-solving environment.

Advanced features—for which we expect SSPF to display the greatest utility—are under development. These features focus on bottleneck identification, deterministic analysis, statistical analysis, process simulation, and decision support. With a solid platform of data collection, retention, and handling, we are positioned to proceed with these functions.

Through applying models, we expect to create algorithms that emulate soft rules and permit the seamless integration of third-party analytical tools. This integration supports decision making in a cost-effective manner.

Acknowledgments
This project was supported in part by Saturn Corp. and the US Defense Advanced Research Projects Agency/ITO contract number F30602-96-2-0227.

References

Earl Long is a data analysis manager at Saturn Corp. His research interests include model-integrated computing. Long received a BS in mechanical/electrical engineering from Kettering University, an MBA from Eastern Michigan University, and an MS in information systems from Eastern Michigan University.

Amit Misra is a research assistant professor at Vanderbilt University. His research interests include software engineering, large-scale integrated information systems, and their reliability, diagnosis, and simulation. Misra received a PhD in electrical engineering from Vanderbilt. He is a member of the IEEE and Tau Beta Pi.

Janos Sztipanovits is a professor of electrical and computer engineering at Vanderbilt University and the director of the Institute for Software Integrated Systems. His research interests include structurally adaptive systems, model-integrated computing, and software engineering. Sztipanovits received a PhD in electrical and computer engineering from the Technical University of Budapest. He is a senior member of the IEEE.

Contact Long at Saturn Corp., 100 Saturn Pkwy., Spring Hill, TN 37174; [misra, sztipa]@vuse.vanderbilt.edu.