we have used this technique in some signal processing applications[13].

6. Conclusions

Model-Integrated Computing shows the following advantages in the software and system development process: (1) It establishes a sofware engineering process that promotes design for change. (2) The process shifts the engineering focus from implementing point solutions to capturing and representing the relationship between problems and solutions. (3) It supports the applications with model-integrated program synthesis environments which offer a good deal of end-user programmability. We have found that the critical issue in system acceptance has been to facilitate domain specific modeling. This need has led us to follow an architecture-based tool development strategy that helps separate the generic and domain/application specific system components. The Multigraph Architecture has proven to be efficient in creating domain specific model-integrated program synthesis environments for several major applications.

7. Acknowledgements

The SSPF project has been supported by Saturn Corporation. The general MGA research has been supported, in part, by USAF, NASA, Boeing, DuPont, Sverdrup, and Vanderbilt University. Their support is greatfully acknowledged.

REFERENCES

[1] Abbott, B., Bapty, T., Biegl, C., Karsai, G., Sztipanovits, J.: "Model-Based Approach for Software Synthesis," IEEE Software, pp. 42-53, May, 1993.

[2] Childers,C.A., Apon,A.W., Hooper,W.H., Gordon,K.D., Dowdy,L.W.: "The Multigraph Modeling Tool", Proc. of the 7th International Conference on Parallel and Distributed Systems, Las Vegas, Nevada, October 5-8, 1994.

[3] O. R. Fonorow: "Modeling software tools with Icon", Proc. of the ICSE-10, pp. 202–221, Apr., 1988.

[4] T. Gallo and G. Serrano and F. Tisato: "*Ob-Net*: An Object-oriented Approach for Supporting Large, Long-lived, Highly Configurable Systems", Proc. of the ICSE-11, pp. 138–144, May, 1989.

[5] M. Ganti and P. Goyal and S. Podar: "An Object-oriented Software Application Architecture". Proc. of the ICSE-12, pp. 212–200, March, 1990.

[6] N. Iscoe and G. B. Williams and G. Arango: Domain Modeling for Software Engineering, Proc. of the ICSE-13, pp. 340-343, May, 1991.

[7] Karsai, G., Sztipanovits, J., Franke, H., Padalkar, S., Decaria, F: Model-Embedded Problem Solving Environment for Chemical Engineering, Proc. of IEEE ICECCS'95, pp. 227-234, Florida, 1995.

[8] Karsai,G.: "A Configurable Visual Programming Environment: A Tool for Domain-Specific Programming", IEEE Computer, pp. 36-44., March 1995.

[9] Ledeczi, A., Bapty, T., Karsai, G., Sztipanovits, J.: "Modeling Paradigm for Parallel Signal Processing," The Australian Computer Journal, vol. 27, No.3, pp. 92-102, August, 1995.

[10] Misra, A., Sztipanovits, J., Underbrink, A., Carnes, R., Purves, B.: "Diagnosability of Dynamical Systems," Proc. of the Third International Workshop on Principles of Diagnosis, pp. 239-244, Rosario, WA 1992.

[11] Misra, A. , Sztipanovits, J., Carnes, J. R., "Robust Diagnostics: Structural Redundancy Approach," Proc. of Knowledge Based Artificial Intelligence Systems in Aerospace and Industry conference at SPIE's Symposium on Intelligent Systems, pp. 249-260, Orlando, FLa, April 5-6, 1994.

[12] S. B. Ornburn and R. J. LeBlanc: "Building, modifying, and using component generators", Proc. of the ICSE-15, pp. 391–404, Apr. 1993.

[13] Sztipanovits, J., Wilkes, D., Karsai, G., Biegl, C., Lynd, L: "The Multigraph and Structural Adaptivity," IEEE Transactions on Signal Processing, Vol. 41, No. 8., pp. 2695-2716, 1993.

[14] Sztipanovits,J., Karsai, G., Biegl,Cs., Bapty, T., Ledeczi,A., Misra, A.: "Multigraph: An Architecture for Model-Integrated Computing", Proc. of IEEE ICECCS'95, pp. 361-368, Florida, 1995.

[15] CIMPLICITY MMI and MES/SCADA Products: User Manual, GE Fanuc Automation, February 1996.

a considerable amount of data was available. We also were able to get a time-stamped production data dump from one actual shift which we used to test, verify and successfully demonstrate the prototype. There were two people working on this phase of the project.

In 1996, the prototype was moved towards a production release. This involved a Critical Design Review and considerable changes in all the SSPF components. Some of the changes in these were necessitated by the integration process, but most were just due to added functionalities. A large part of the effort since April 1996 was spent on building the complete models of Saturn. For this another person was required. The model-based approach used for this project facilitated the knowlegde acqisition well enough such that it was decided by Saturn that we would build plant models. The modeling phase involved consulting with Saturn personnel to find out the processes, data points being measured, etc., and then putting this information into the models.

SSPF was put into production release in the first week of August 1996. In this release, 108 process and about 600 buffers have been modeled.

We learned many lessons during our efforts to integrate SSPF with Saturn plant:

- A large part of the effort was required for modeling of the plant. This is not surprising since the application itself is generated from the models.
- Using the MIC approach helped us considerably in verifying and testing the application. Before going into production release, SSPF Beta release was on-line during the model building phase. Whenever the models were added to or changed in any way, the application was regenerated and tested. This allowed us to verify the application *and* the models. The turn-around time was just the time required to change the models and to regenerate the configuration files. It showed clearly how flexible and maintainable the application becomes through the use of models.
- Due to iterations on functional specifications for SSPF, many times during the integration phase, requirements and/or enhancements in the functionality of SSPF were added. We had a very quick turn-around time on these since all it required was a change in one configurable component followed by re-generation of the application, and the change would appear for all the processes/buffers of the plant. Without the use of MIC, trying to keep an application upgrade consistent for 108 processes (and about 600 buffers) would be a very difficult and costly task.
- Since the data acquisition systems in different sections of the plant were implemented by different people, we had to deal with the "idiosyncracies" of these implementations. Being able to capture this information in models also helped considerably.

One of the issues raised during the integration process related to the performance and robustness of the server components. The tremendous volume of data flowing through SSPF presented us with implementation challenges, particularly for SSPF Server components. In the future, the volume of data is expected to grow and will necessitate further changes, e.g., multiple servers for RTDS, databases, etc. Using MIC approach will aid considerably in experimenting and fine tuning different architectures.

Another issue raised was that of the adequacy of the modeling paradigm. Since the application is generated from the models, its functionality is strongly affected by the models. The first modeling paradigm was adequate for the original functional specifications. However, due to the revisions and added functionalities, it was realized that the modeling paradigm needed to be redesigned to be able to model the plant in more detail and capture more physical and behavioral characteristics of processes, etc., and to include many other features.

5. Other Approaches

The benefits of software modeling and generating software from models (to facilitate rapid development) has been known for some time[3]. The importance of domain modeling in the software development process has been recently recognized[6][5].

Many of the concepts developed in [6] are present in our toolset, MGA. Just like in [5], the domain-specific models play an essential role. What makes our approach different, however, is the explicit model interpreter component, that decouples the execution of the system from the models. In a sense, our model interpreters are similar to component generators, as described in [12]: they instantiate and configure generic components and templates. The differences are: (1) this process may be executed at run-time, and (2) in our system we explicitly allow re-interpretation. In the latter case, there can be a feedback from the executing system to the model interpreter, and thus components can be changed dynamically based on events appearing in the system. Our run-time system allows this, and

- *ODBC Interface*: The ODBC interface is used by RTDS to read shift information and to store detailed and summary production data. HTDS uses the ODBC interface to retrieve historical production data at client request.
- *MS/SQL Server and Databases*: There are two databases used by SSPF: (1) `SSPFData`, which is used to store current and historical data for one week period, and (2) `SSPFShift`, which is used to store shift information
- *Client GUI*: This component presents the current and historical data to users, provides navigation through various sections of the plant, etc.
- *Client Data Handler*: This component is responsible for interfacing to RTDS and HTDS and getting and maintaining current and historical data.
- *Shift Manager*: This is a utility for updating and maintaining the shift information in the `SSPFShift` database.

Interfaces and Communication

For communication between the various components, either APIs were used or communication protocols (transport and presentation layers) were developed. The interfaces and communication mechanisms used by various components of SSPF are:

- Cimplicity project SSPFPB receives point data from area nodes at Saturn Site.
- Cimplicity Interface receives the points from SSPFPB using the Cimplicity API.
- RTDS receives point data from Cimplicity Interface over a TCP/IP connection using a stream socket.
- RTDS multicasts data and information packets to SSPF Clients using datagrams. These packets are used by the clients to update their own copies of production data. This manner of transmission of production data between the RTDS and client was used to allow any number of clients to be running at any given time and to keep RTDS independent from the clients. Due to this design, synchronization is needed between clients and RTDS, which is done using stream sockets as described below.
- RTDS opens a stream socket and listens for incoming client requests (for startup information, synchronization, manual input, etc.), services the request and closes the connection. This allows RTDS to service any number of clients (at different times).
- HTDS opens a stream socket and listens for incoming client requests for historical data. It sends back the requested data on the same socket.
- RTDS and HTDS make calls to the functions in the ODBC Interface (no communication involved).
- The ODBC Interface uses the ODBC API to interface to the SQL Server.

3.6. Model Interpretation and SSPF Component Configuration

The Application Generator (AG) "transforms" the SSPF models into the run-time system. This was accomplished in the following manner:

1. Configurable run-time libraries and programs were developed, which get their configuration information from configuration files produced by the AG.
2. Schemas for storing production data were defined. At this time, this is a manual process. In the future, these schemas will also be generated from the models.
3. AG traverses the model database, extracting the relevant information and produces a number of configuration files and SQL script files.
4. The configuration files are read by the SSPF components to build internal data structures, thus reflecting exactly what is in the models.
5. The SQL scripts are executed by the SQL/Server. The SQL scripts fill in the rows for the tables with essential information about the processes, buffers, etc. in the plant. Filling in the tables from information in the models guarantees that RTDS, HTDS and ODBC Interface (which are themselves configured from the models) will be consistent with each other and will log/retreive the data to/from the proper places. In addition, model-configured database browsing tools can be developed, allowing easy and consistent access to production data on a site-wide scale.

If the models are changed to reflect changes in the plant, only the last three steps need to performed. If a change in the functionality of SSPF is desired, the first two (and possibly the last three steps also) need to be performed.

4. Experiences

The SSPF project was started in September of 1995, with an Engineering Study and preliminary design. By the end of the year, a prototype was developed. The prototype included the modeling paradigm, AG, RTDS, HTDS, Client, ODBC Interface, database schemas and Cimplicity Interface. About one-third of the plant was modeled. This was the section where

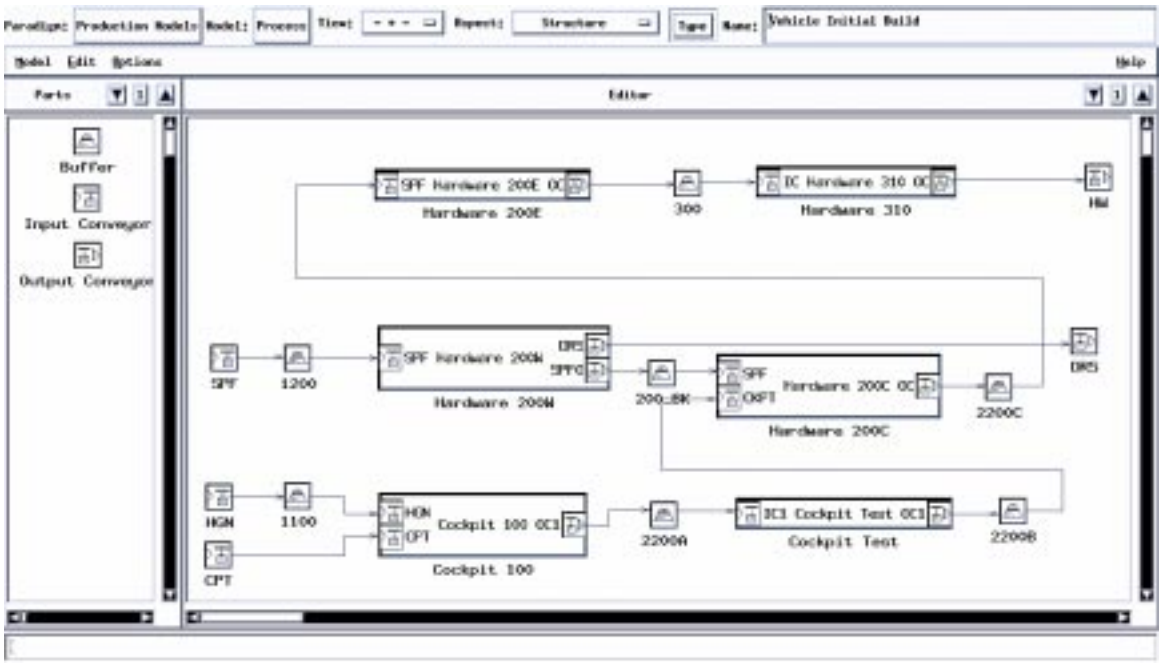

Figure 4: Production Model for Vehicle Initial Build

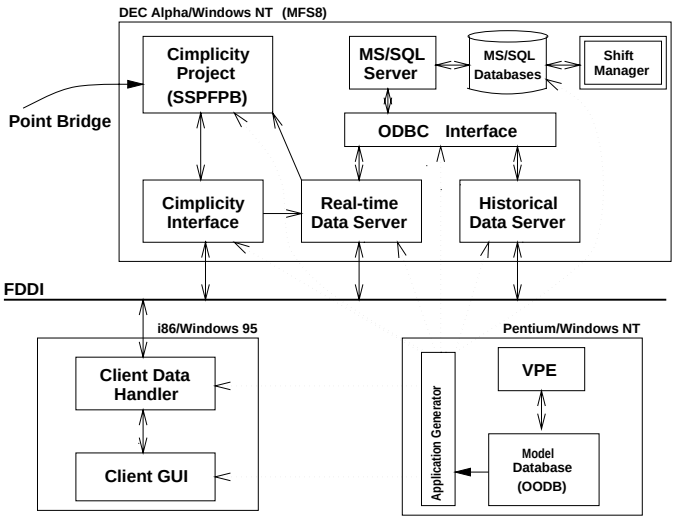

Figure 6: SSPF Architecture

Data Server (RTDS), Historical Data Server (HTDS), Cimplicity Interface, a Cimplicity project (SSPFPB), ODBC Interface and one or more MS/SQL Server and MS/SQL Database; (3) *SSPF Client* which consists of the Client Data Handler and the Client GUI.

The SSPF Server and SSPF Client together are called the SSPF Application. The SSPF Server components run on a DEC Alpha/Windows NT Server. The SSPF Clients run on a number of Intel workstations distributed throughout the Saturn Site. The various components and their functionalities are:

- *Visual Programming Environment (VPE)*: The VPE is used to build graphical models of the Saturn plant.
- *Model Database*: This is an object-oriented database which is used to store the models.
- *Application Generator*: This is the model interpreter which configures the various run-time components of SSPF. The dotted lines from Application Generator to the various components in Figure 6 indicate this fact.
- *Real-Time Data Server (RTDS)*: The RTDS receives real-time point data, processes the data, multicasts the data to clients, logs the raw computed real-time data and shift and day summaries, reads shift information for the processes from the shift tables, provides clients with startup and synchronization information, etc.
- *Historical Data Server (HTDS)*: HTDS services client requests for historical data.
- *Cimplicity Interface*: This component gets point data from the SSPFPB point bridge project and forwards the data to RTDS.
- *Cimplicity project SSPFPB*: This is a point bridge Cimplicity project that gets the plant data from various PLCs at Saturn Site.

plant undergoes changes (the extent and degree of the change varies). SSPF needs to be able to incorporate these changes with minimal effort and no disruption of the plant operations.

Evolution. In its initial incarnation, SSPF is a data collection, logging, retrieval and display service. In the future, however, SSPF will have many more components added to it that will provide simulation, bottleneck analysis, diagnosis, decision making support, etc. These components will face the same issues outlined above. Thus SSPF had to be designed for easy extensibility.

3.3. SSPF Modeling Paradigm

The SSPF application offers a structured view of the data representing the state of the manufacturing processes. This structured view and the related visualization services create a tight conceptual relationship between the plant and the SSPF software. In this section, we summarize the key modeling concepts that are used for defining the SSPF application and that are also provided for the users of the system.

Background. The manufacturing plant is viewed as an aggregate of processes and buffers. Processes represent the operations required for making a car, such as casting, machining, welding, assembly, etc. Associated with each process are certain measurements that relate to the productivity of the process. Examples of such measurements are : cycle-time, production count (how many parts were machined, assembled, etc.), Work In Process (WIP) (how many parts are currently being worked on), production downtime (equipment breakdown, etc.).

Buffers (or banks) lie between processes and hold parts that are produced by an upstream process before they are consumed by a downstream process. The information about banks that is relevant to production is: bank count (number of parts/sub-assemblies in the buffer) and the minimum and maximum capacities of the buffer.

The inter-connectivity of processes and buffers captures the sequence of operations required to produce a car. The concept of production flow is concerned with the flow of material (raw materials, parts, subassemblies, etc.) through the processes and buffers, and encompasses all the production related entities of processes and buffers (e.g. production count, WIP, bank count, starving, blocking).

To model Saturn Site in terms of its production processes and business organizations, the SSPF modeling paradigm was developed. The modeling paradigm utilizes four kinds of models: (1) Production Models, (2) Organization Models, (3) Activity Models and (4) Resource Models. Production models (an example iis shown in Figure ??) are used to represent the production flow at Saturn Site. The Organization models are used to represent the business units at Saturn and to establish relationships between business units and production units. Activity models are used to configure the SSPF activity(ies) while resource models describe the allocation of SSPF activity(ies) to workstations.

```
...
models
 Process compound {
  StructuralAspect "Structure" {
   icon rect { left  : InConveyors;
               right : OutConveyors; };
   attrs { attr PartNames; attr LineSpeed;
           attr JPHLow; attr JPHLowLow; }
   conns { JobFlow { 1 solid line arrow } :
    { InConveyors -> Buffers }
    { InConveyors -> SubProcs InConveyors }
    { Buffers -> SubProcs InConveyors }
    { SubProcs OutConveyors -> Buffers }; }
   parts { SubProcs : Process hierarchy;
           Buffers : Buffer;
           InConveyors : In_Conveyor link;
           OutConveyors : Out_Conveyor link;}
 }
...
```

Figure 5: Fragment of the SSPF paradigm definition

3.4. Model Builder/Editor Environment

The model editor was generated by customizing the MGA VPE for SSPF. VPE is customized through a paradigm specification file, that contains a declarative description of the modeling paradigm. This file is used to generate the model database schema and the specific database interface code for the model builder [8]. A fragment of the SSPF editor configuration file is reproduced in Figure 5: it shows the definition of the stuctural aspect of the processing models.

3.5. SSPF Architecture

There are three main parts to the SSPF system as shown in Figure 6: (1) *Model-Integrated Programs Synthesis (MIPS) Environment* which consists of the Visual Programming Environment (VPE), Model Database and the Application Generator (AG); (2) *SSPF Server* which consists of the Real-Time

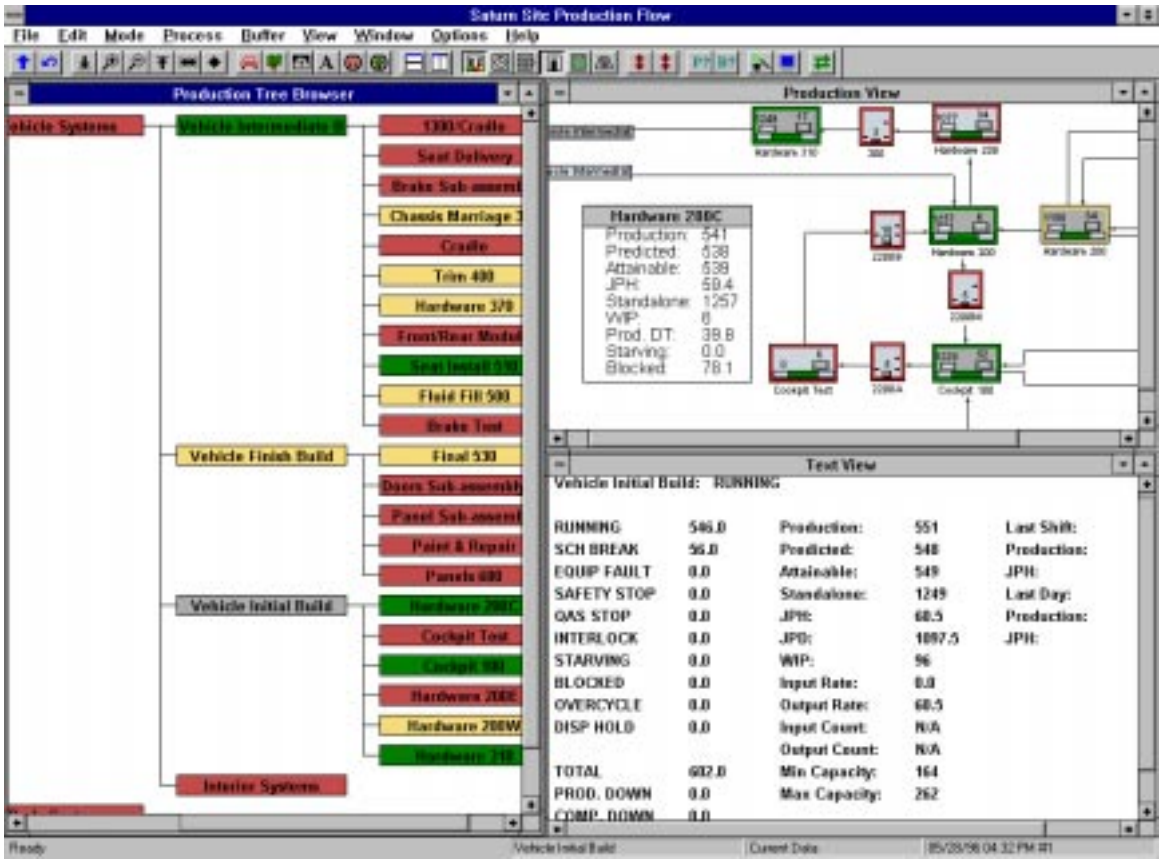

Figure 3: SSPF GUI

client/server architecture. It supports a multitude of users (about 300) with low load on network, server, or clients. The number of users is high because any person in the workforce at Saturn would be expected to be an active user.

- *Production database* : The raw plant data and processed data need to be logged to and retrieved from a relational database, in a structured and flexible manner.
- *Heterogeneous platforms* : The components of SSPF run on different hardware and software platforms.
- *Sophisticated GUI* : The SSPF GUI needs to show real-time and historical data and results of analyses, etc., using the same interface, provide easy navigational facilities, etc.
- *Robustness* : Robustness against corrupted data is a very important criteria. Since the PLCs are distributed throughtout the plant, the data provided by PLCs is frequently incorrect. Many times the connection to PLCs also is lost. SSPF is required to be able to handle such situations.

Integration. The issues faced during integration of SSPF with Saturn systems are:

- *Data collection* : The data collection methods used in different sections of the plant were implemented by different people. As a result, there was considerable heterogeneity in the data points with respect to the engineering units used, reset conditions, etc. This presented significant challenge for SSPF since its purpose is to provide a homogeneous view of the plant to the user.
- *Zero disruption* : The SSPF integration process was not allowed to disrupt the plant operation and other software packages in any way.
- *Pre-existing hardware and software platforms* : SSPF had to be designed to run on the hadrware/operating systems already in use at Saturn and to interface with existing software packages (Cimplicity, MS SQL/Server, etc.).

Maintenance. Due to the continuous, ongoing improvments in the product, the production processes and the engineering and business practices, the Saturn

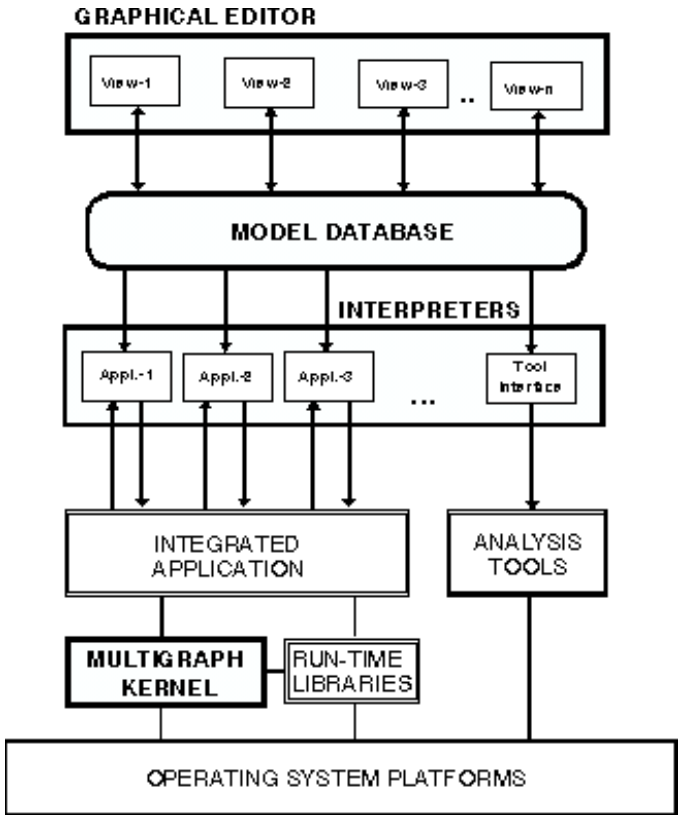

Figure 2: MGA Components

shifts, weeks, or months of history. To understand the dynamics of production flow, storage of detailed and summarized data in a structured format is required. The current databases at Saturn are difficult to use due to the lack of structure or a framework for the data.

SSPF stores the raw data (received from the plant through Cimplicity) and processed information in a structured manner using the a relational database (Microsoft SQL/Server). SSPF also includes tools to retrieve stored data for use in analysis and decision support tools. The database schemas and the interfaces to the database are configured from the plant models, thereby providing the framework for easy access and maintenance of the production database.

Graphical User Interface. The primary purpose of SSPF is to provide the users with current (real-time) and historical production data, which can be used for various purposes – monitoring, anlaysis, etc. For presenting the data SSPF includes a Graphical User Interface (GUI) (see Figure 3), which is configured from the plant models. The GUI has many features, some of which are:

- navigation through the plant allowing the user to examine the production flow through any section of the plant.
- detailed and aggregate views of the plant.
- alternate views.
- drill-down capabilities.
- textual reports

Bottleneck Analysis A key concept that is central to production flow is that of a bottleneck. A bottleneck is that process that is limiting the overall production flow. The overall concept of production flow is built on focusing attention on bottlenecks and potential bottlenecks. Bottlenecks may at first appear to be obvious, which is true if the process is simple enough and can be completely observed from a single point. In the case of a large discrete manufacturing plant such as Saturn, this is not possible. SSPF is intended to provide a total virtual view of the production flow across the plant and thus, aid in clear identification of bottlenecks.

Problem Solving Activities. Simulation, predictive techniques, and decision support will be provided in a common framework using the same models that are used for real-time monitoring, data storage and retrieval. Using traditional techniques for analysis is facilitated through application of MIC. Tools will be readily integrated into the problem space, seamlessly mixed as appropriate.

3.2. Design Challenges and Issues

The design issues and challenges faced by SSPF are common to most large-scale software systems. They can be categorized in the following manner: (1) Design and development, (2) Integration, (3) Maintenance, and (4) Evolution.

Design and Development. The issues faced in design and development of SSPF are:

- *Tight integration with the plant* : Since SSPF is "embedded" into the Saturn plant, the representation of the plant as used by the SSPF software component needs to match exactly the actual physical processes, buffers, machines, etc.
- *Real-time performance* : There are a large number of data points that are measured from the plant and provided to SSPF. The points are asynchronous and necessitate an event-driven architecture for SSPF. Every measurement that arrives needs to be processed, logged and supplied to clients for display.
- *Distributed system* : SSPF has a distributed

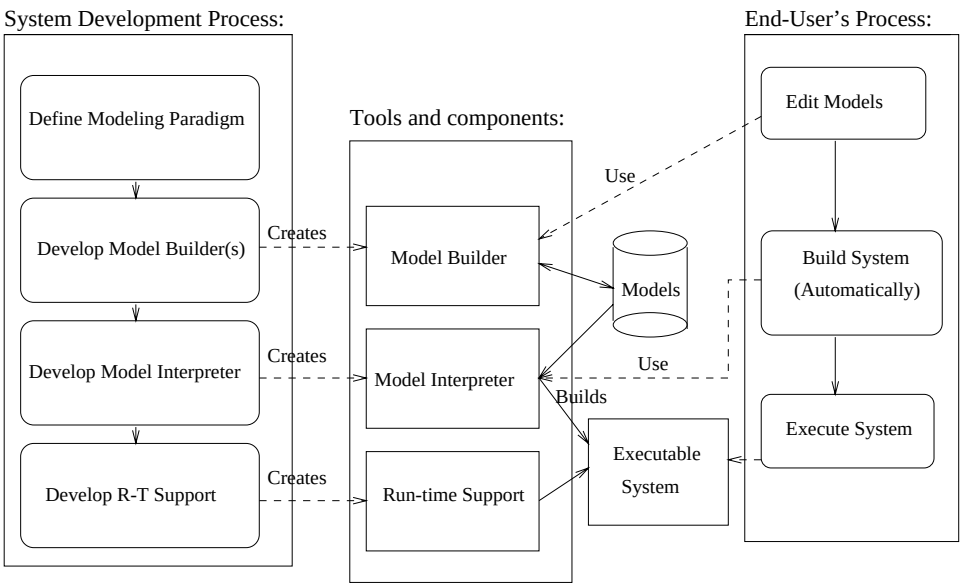

Figure 1: Model-Integrated System Development Process

dataflow based run-time kernel: the Multigraph Kernel (MGK). MGK facilitates the dynamic creation of networks of computing objects, even across processors, and the scheduling of those objects.

The flexibility with which MGA can be adopted to various application domains has enabled us to use it in widely different projects during the last 10 years, e.g., [1, 2, 7, 9, 10, 11]. In the following we describe the SSPF systems, which was developed using MIC.

3. A Practical System: SSPF

In this section, we describe the functionalities, requirements, design and development of the application of MIC towards providing a poblem-solving environment and decision support tools in the context of discrete manufacturing operations at Saturn Corp. The Saturn Site Production Flow (SSPF) system is a client-server application designed to meet an initiative within Saturn Manufacturing to increase the number of cars built utilizing existing manufacturing facilities and processes. The primary focus of tools and services provided by SSPF is the flow of material throughout the production facility (site-wide production flow). SSPF is intended to provide an integrated problem-solving environment needed for informed decision making by the team members and leaders within Saturn.

3.1. SSPF Functionalities

SSPF is designed to be an application that evolves easily. The functionalities described below represent the capabilities of the system that have already been identified and have been implemented or are currently under development. In the future, the requirements and functionalities of the system are expected to grow considerably.

Data Acquisition. SSPF functions involve real-time collection, presentation, storage, retrieval, and analysis of data. There is a data rich environment at Saturn based on traditional process monitoring and control (PM&C). There are a large number of points of live data that are monitored which consist of production counts, downtimes, bank counts, etc. The majority of these data points are collected from the Programmable Logic Controllers (PLCs) of machines using a data acquisition package called Cimplicity[15]. In the absence of any plant models to guide the data collection, logging and presentation, the enormous volume of data presents considerable difficulties in monitoring site-wide status and performing simulations and other decision making analyses.

SSPF uses Cimplicity's data acquisition as its interface to the PLCs. This interface is configured from the models, as described later, thereby affording considerable ease in maintenance and upgrade of the plant data interface as the plant itself changes.

Data Storage and Retrieval. Time is an essential dimension in understanding the dynamics of production flow. There are some dynamics that are within the bounds of a given shift. Others involve multiple

ther software engineers must become plant engineers (up to a certain degree, of course) or plant engineers must become software engineers.

On a deeper level, one can recognize that one source of the problems is the lack of end-user programmability. If the plant engineer, as an end-user, would be capable of describing changes in the plant - which in turn would result in the required changes in the software system, the problem would be more easily manageable.

In this paper we describe the Model-Integrated Computing approach as a solution to this problem. First we describe the approach on an abstract level, next we present a set of tools that support the approach. The major part of the paper describes the process the we followed to develop an actual application for a large-scale manufacturing operation. We discuss our experience with the process and give an evaluation of the work. Finally, we compare our approach to other development approaches.

2. Model-Integrated Computing

Recognizing the need for software systems that evolve with their environment, we propose the extensive use of models in the development process. The use of models in software development is not a new idea. Various analysis and design techniques (especially the object-oriented ones) very frequently build models of the system before realization, and model its environment as well. However, we propose to extend and specialize the modeling process so that the models can be more tightly integrated into the system development cycle than in traditional techniques. The process supporting this activity is called Model-Integrated Computing (MIC), and it results in a model-integrated system (MIS).

In an MIC process the models describe the system's environment, represent the system's architecture *and* they are used in generating and configuring the system. These models are indeed integrated with the system, in the sense that they are active participants in the development process, as opposed to being mere passive documents.

When MIC is used in developing a system, models are involved in all stages of the life-cycle. To support this, the initial step is the building of tools that support model creation and editing, used by the end-users when they want to customize the final application. The model editing tools are typically graphical, but more importantly, they support modeling in terms of the actual application domain. This domain-specific modeling is essential for making end-user programmability feasible.

There are two interrelated processes in MIC: (1) the process that involves the development of the model-integrated system, and (2) the process that is performed by the end-user of the system (in order to maintain, upgrade, reconfigure) the system – in accordance with the changes in its environment. Figure 1 shows the processes schematically. The first process is performed entirely by the system's developers (i.e., software engineers), the second, usually, by the end-users.

To summarize, in MIC the system is created through the development of the following: (1) a modeling paradigm, (2) the model builder (editor) environment, (3) the model interpreters and (4) the run-time support system. The product of this process is a set of tools: the model builder, model interpreter and generic run-time support system. Using these, first the developers, but eventually the end-users can build up the application itself by going through the following steps: (1) develop models, (2) interpret the models and generate the system (this step is automatic), and (3) execute the system. The key aspect of the development process is that domain-specific models are used in building the application, and thus it can be re-generated by the end-users.

It may seem that MIC necessitates a bigger effort than straightforward application development. This is true if there is no reuse and every project has to start from scratch. In recent years we have developed a toolset called the *Multigraph Architecture*(MGA)[14] that provides a highly reusable set of generic tools to support MIC. We claim that the tools provide a *meta-architecture*, because instead of enforcing one particular architectural style for development, they can be customized to create systems of widely different styles. Figure 2 shows the components found in a typical MGA application. The shadowed boxes indicate components that are generic and are customized for a particular domain.

In MGA we use a generic Visual Programming Environment (VPE)[8] for model building. Models are stored in an object-database: also a customizable component. The domain-specific customization of these components determines how the visual editor behaves, and how the database schema is organized. The model interpreters are typically highly domain-specific, although some of their components are general (e.g., low-level database access mechanism). For run-time support purposes we have successfully used a macro-

Model-Integrated Development of Complex Applications*

Amit Misra, Gabor Karsai and Janos Sztipanovits
Department of Electrical and Computer Engineering
Vanderbilt University
P.O. Box 1824 Station B
Nashville, TN 37235 USA
+1-615-322-2771
{misra,gabor,sztipaj}@vuse.vanderbilt.edu

Abstract

Many large distributed applications are tightly integrated with their physical environments and must be adapted when their environment changes. Typically, software development methodologies do not place a large emphasis on modeling the system's environment, and hence environmental changes may lead to significant discrepancies in the software. In this paper we argue that (1) the modeling of the environment should be an integral part of the process, and (2) to support software evolution, wherever possible, the software should be automatically generated. We present a model-integrated development approach that is capable of supporting cost effective system evolution in accordance with changes in the system's environment. The approach is supported by a "meta-architecture" that provides a framework for building complex software systems using COTS and custom developed software components. This framework has been successfully used in various projects. One of these projects, a site production flow visualization system for a large manufacturing operation, will be analyzed in detail. First, we show how the model-integrated process can be generalized and used to build families of model-integrated tools that support the development of specific systems. Next, we describe how the generic architecture was customized for the particular domain. Next, we discuss how specific components were implemented, and present a detailed experience report (both from developers and end-users).

*This work has been supported in part by the DARPA/ITO EDCS Program, Contract #F30602-96-2-0227

Keywords
model-integrated computing, end-user programming, component-based software integration

1. Introduction

Every software practitioner knows the difficulties of maintaining a large software system which is tightly coupled to a physical environment that is frequently undergoing configuration changes. In fact, it is probable that such systems are rather the rule than the exception. There are many causes for this problem, but some are more prevalent than others. Consider an example system that monitors and controls a large-scale manufacturing operation. The system collects data from thousands of sources (PLC-s, microswitches, etc.), archives the data values, makes the data available to operators and managers (after processing), and is also involved in making automatic decisions which enforce some level of production control. It is a fact of business that the plant changes and evolves, which will necessitate changes and upgrades in the software system. One has to change the database schema, recompile applications, reconfigure data acquisition systems, change user interfaces, etc., just to maintain existing functionalities. These many-faceted activities involve diverse software engineering issues, and the upkeep of the system becomes a highly non-trivial activity. To understand and maintain the relationship between the plant configuration and the software configuration, ei-