Model Integrated Computing-Based Software Design and Evolution

Greg Nordstrom, Gabor Karsai, Michael Moore, Ted Bapty, and Janos Sztipanovits
Institute for Software Integrated Systems
Vanderbilt University
Hill Center Addition, Room 201
230 Appleton Place
Nashville, TN 37203-5701
Phone: (615) 343-7521
Fax: (615) 343 7440
{gnordstr, gabor, msm, bapty, sztipaj@vuse.vanderbilt.edu}

Abstract

Among the most significant technological developments of the past 20 years are computer-based systems (CBSs), where functional, performance, and reliability requirements demand the tight integration of physical processes and information processing. Because complex component interactions exist in these systems, we must construct the software and its associated hardware such that they can evolve together.

Model integrated computing (MIC) is an effective and efficient method for developing, maintaining, and evolving large-scale, domain-specific CBS applications. MIC is model-based, allowing the synthesis of application programs from models created using customized, domain-specific, multi-aspect model integrated program synthesis (MIPS) environments. Integrated models explicitly represent dependencies and constraints among various design views. Because engineers can input design information at appropriate levels in the design hierarchy, and are freed from low-level implementation details, true end-user programmability is achieved.

This paper discusses MIC technology and presents two large-scale MIC applications currently in use—the Saturn Site Production Flow (SSPF) system and the Integrated Test Information System (ITIS). The SSPF is a manufacturing execution system used by General Motors' Saturn division to model, monitor, and analyze throughput characteristics of the plant. SSPF has been deployed in two Saturn plants and has helped Saturn to increase throughput in the Spring Hill, TN plant by nearly 10%.

The ITIS, used to support Department of Defense aerospace testing at Arnold Engineering Development Center, integrates diverse sets of information from distributed, heterogeneous data sources into a seamless real-time, on-demand data system. The ITIS allows for rapid generation and customization of test information systems that track changing user requirements, allowing secure, uniform access to search metadata and retrieve test data from geographically distributed engineering teams. Using MIC technology, the system can respond to rapidly changing requirements for the interconnection of a wide variety of data sources and analyses.

I. Introduction

Developing and maintaining software for large-scale systems where a high degree of coupling exists between the software and its environment, and where the environment changes with time, is a difficult and time-consuming task. Many causes contribute to the difficulty, with some being more prevalent than others. For example, consider a manufacturing execution system that monitors the state of a large-scale manufacturing facility. The system collects and stores data from hundreds or thousands of points within the plant, and reduces that data into various real-time and historical production and status reports for use by production control, process and plant management, and business-oriented decision makers. Such a software-based system, while designed to respond to normal fluctuations in production flow, must also respond to changes in the business itself. These changes may involve database schema modifications, performance tuning, modifications to the data processing algorithms, etc. to maintain the capabilities of the system. Also, changes in the business model often necessitate additional features in the software—features previously unnecessary or unavailable due to the plant environment (i.e. the software's operating environment). To ensure that such software development and evolution efforts are performed effectively, software developers must become "experts" in many areas—the overall operation of the plant, the business process models, and, of course, company-adopted software development methodologies. Also, to a lesser degree, the non-software development personnel (production engineers, business managers, and other support system developers) must become intimately familiar with the issues and concerns of the software development team. This leads to inefficiencies and errors in the software development cycle.

We advocate an approach to software development and evolution that addresses these problems. We describe the approach on an abstract level, and present a set of tools to support the approach. Next we discuss two applications of this approach—one used to enhance the throughput of the General Motors Saturn automobile manufacturing facility in Spring Hill, Tennessee, and another used to integrate heterogeneous, legacy test support data systems in U.S. Air Force test facilities at Arnold Engineering Development Center, Arnold Air Force Base, Tennessee. We begin with a discussion of model integrated computing.

II. Model-Integrated Computing

We propose the use of models in the development and evolution of large-scale software systems [1]. Of course, the use of models in software development is not a new idea. Many analysis and design techniques (especially the object-oriented approaches) use models to describe the necessary class and inheritance relationships that must exist in the software. Such techniques can be extended to the modeling of external interfaces to the environment. However, such models remain loosely coupled to the actual system development cycle. We propose to extend and specialize this approach, forming a tightly coupled environment where the software, the environment, and the integration constraints are all modeled, with the resulting models used to generate and/or configure the necessary software components of the actual system (see [2] for some background on component generation). This process is called

Model Integrated Computing (MIC) [3]. The MIC-based development cycle is shown in Figure 1 below.

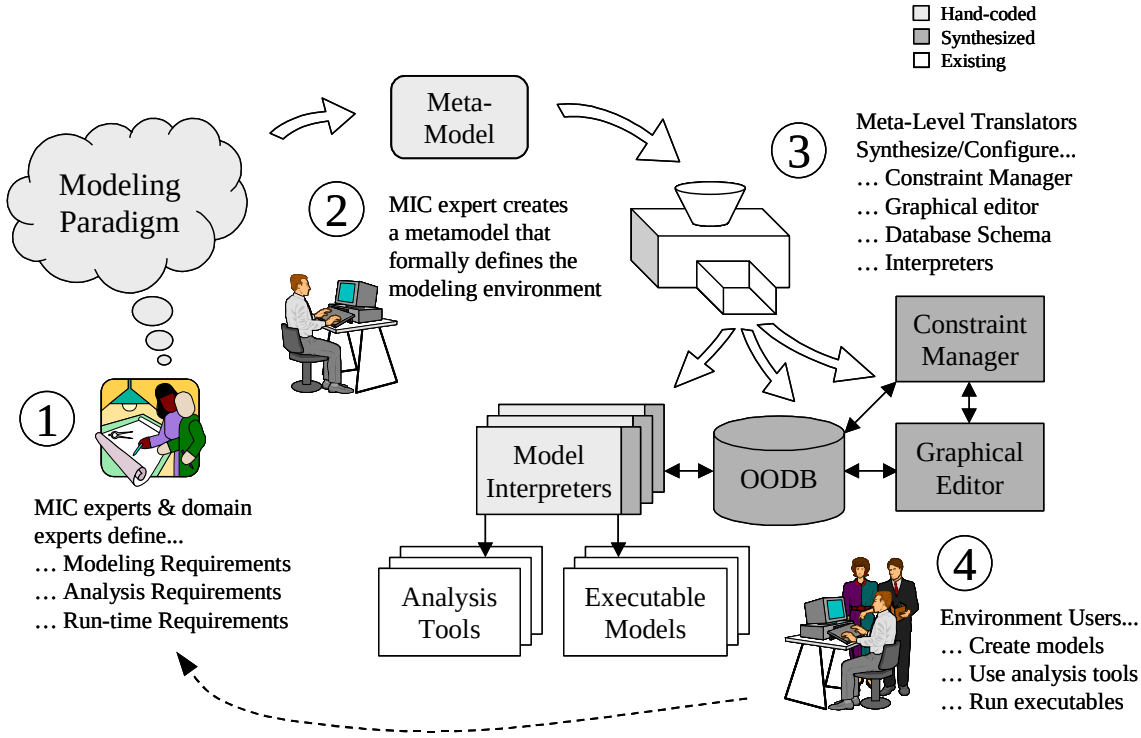

Figure 1. Model integrated computing-based development

Initially, a *modeling paradigm* is developed. The modeling paradigm defines the overall modeling effort—what is to be modeled (which aspects of the system, its behavior, its environment, etc.), how the necessary models are to be constructed (the syntactic, semantic, and presentation specifications of the resultant modeling language), and how the models are to be used (the model translation and interpretation requirements). Modeling paradigm definition requires both MIC- and domain experts. These experts work together to define the domain's modeling, analysis, and run-time requirements.

Once a modeling paradigm has been established, the modeling environment requirements are formally stated in a *metamodel*. The metamodel specifies the domain-specific, graphical modeling language (i.e. the domain-specific modeling environment) to be used when constructing models within the particular domain in question [4]. The domain-specific modeling environment (DSME) is synthesized from the metamodel by a meta-level translation process. The output of the meta-level translator is used to customize a suite of configurable model editing tools. These editing tools are typically graphical, but more importantly, they support modeling in terms of the actual application domain. Domain-specific modeling is essential to enable end-user programmability.

Once created, domain models are stored in a model database. In order to use the models effectively, one needs (at least) two more components beyond model editors: (1) tools for transforming abstract models into executable systems, and (2) run-time support libraries for the executable system. The transformation is done by a component called the *model interpreter*. A model interpreter traverses the model database, analyzes the models, and "creates" the executable system. Model interpreters can be implemented using various strategies, depending on what the run-time system looks like. For instance, if the run-time system includes a relational database, model interpreters can generate the SQL definitions for the schema; if it is a multi-tasking kernel, model interpreters generate the code skeletons performing synchronization; if it is a Petri-net simulator, they generate the configuration tables for use by the simulator. In the most general terms, the **model interpreters are responsible for mapping domain-specific models into run-time components**. Often, run-time systems contain "generic" components that are specialized according to need (as derived from the models). They form the run-time support libraries mentioned above. Model interpreters perform an automatic system generation by instantiating and customizing the generic components. Note that the models are domain-specific, and do not (necessarily) include software concepts, even though the end solution is software-based.

At the process level, MIC has two interrelated processes: (1) the process that involves the development of the model-integrated (i.e. modeling) system, and (2) the process that is performed by the end-user of the system in order to maintain, upgrade, and reconfigure the domain-based system (i.e. application programs), in accordance with the changes in its environment.

To summarize, with MIC the system is created through the following steps: (1) definition of a modeling paradigm, (2) creation of a metamodel describing the DSME, (3) synthesis of the model builder (editor) environment, development of the model interpreters, and development of the run-time support system, and (4) use of the modeling and application synthesis tools by domain-aware end users. The key aspect of the development process is that *domain-specific models are used in building the applicatio*n, and the application can be regenerated by the end-users.

The MIC approach can be contrasted with current development practices as follows. As opposed to developing a highly specialized product, in MIC we want to understand an entire *class* of problems related to a particular domain. As opposed to developing a specific application, we try to develop first the domain-specific tools to model (i.e. specify) the solution, then use these models to generate the actual application. Many of these ideas can already be found in other large-scale packages [5]. What is different here is that, in addition to making the models themselves available for the end-users, we want to make explicit use of the models in generating applications. See the Acknowledgments section for more on related software development technologies.

It would appear that MIC necessitates a bigger effort than straightforward application development. This is true only if there is no reuse and every project has to start "from scratch." In recent years we have developed a toolset called the *MultiGraph Architectur*e

(MGA) [6] that provides a highly reusable set of generic tools with which to do 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 design environments that embody many different styles, as required by domain practitioners.

In the MGA we use a Generic Modeling Environment (GME) [20] (formerly called the Visual Programming Environment (VPE) [7]) for model building. Models are stored in an object-database; another customizable component. The domain-specific customization of these components determines how the visual editor behaves, how the database schema is organized, and how domain-specific constraints are enforced. The model interpreters are typically highly domain-specific. Model interpreters transform models into executable code and/or to the input language of various, domain-specific analysis tools. For run-time support purposes we have successfully used a macro-dataflow based run-time kernel that facilitates the dynamic creation of networks of computing objects, even across processors, and the scheduling of those objects. The flexibility with which the MGA can be adapted to various application domains has enabled us to use it in widely different projects during the past 12-15 years [12]. Three of these projects are briefly described in the paragraphs below, followed by a detailed discussion of two large-scale projects—SSPF and ITIS—in sections III and IV, respectively.

RDS and **DTool**: The MGA was used as the software framework for a robust, model-integrated real-time diagnostic system (RDS) [8], and a diagnosability and testability analysis tool (DTool) [9]. Both tools are used by Boeing on the International Space Station Alpha (ISSA) program to evaluate detectability, distinguishability, and predictability of faults given on-line sensor allocation and built-in-test coverage (BIT). In the case of RDS, the models are interpreted and the software for the RDS is generated automatically. The RDS implements algorithms that provide timely diagnosis for complex systems, even in the case of sensor failures.

IPCS: The Intelligent Process Control System (IPCS) is an on-line problem solving environment and decision support tool for process and production management. The central concepts of IPCS are models of the plant and the process engineering activities. Plant models include a variety of modeling views, including process flow sheets, static and dynamic process equations, finite-state models, failure propagations, equipment structure, etc. Activity models cover a wide range of tasks related to process and production management (e.g. analysis of process operations). The activity models are automatically translated into executable software. The IPCS system is actively used at the DuPont Old Hickory, TN, plant for the development of commercial applications, including monitoring, sensor data validation, on-line process simulation, and process diagnosis [10].

CADDMAS: The MGA is the underlying software technology for the Computer Aided Dynamic Data Monitoring System (CADDMAS) developed in close cooperation with the USAF Arnold Engineering and Development Center (AEDC). CADDMAS provides real-time vibration analysis for 48 channels of 50 kHz bandwidth using a heterogeneous network of nearly 100 processors [1][11]. In the CADDMAS application, the modeling environment

supports the hierarchical modeling of signal flow graphs, hardware resources, and resource limitations [1]. A model interpreter synthesizes the complex executable program and configures the parallel computing platform.

III. The Saturn Site Production Flow System

The Saturn Site Production Flow (SSPF) system is a manufacturing execution system designed for the General Motors Saturn Automobile manufacturing facility in Spring Hill, TN. The SSPF acquires production data from the underlying plant instrumentation system, computes a standard set of throughput measurements from that data, stores that information, presents reduced results in real time to the production floor, and aids in the management reporting and analysis (i.e. back office) business processes.

A. SSPF Functionalities

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). The data being measured consists of production counts, downtimes, bank counts, and other production related information. Data can be presented on screens, using an existing data acquisition and display package. However, in the absence of any structured plant models to guide the data collection, logging and presentation, the enormous volume of data presents considerable difficulties in using the system for monitoring site-wide status and for performing simulations and other decision making analyses.

Data Storage and Retrieval. Time is an essential factor in understanding the dynamics of production flow. The stored data contains detailed histories for every process and buffer in the plant. Furthermore, summarized information for a shift, day, week, and month is maintained. Even though Saturn currently has systems in place that log and retrieve the production data, the practical usability of this data is limited—access to data is very difficult. SSPF stores the raw data and processed information in a structured manner using a relational database (Microsoft SQL/Server). The database schemas and the interfaces to the database are generated automatically from the plant models, thereby providing the framework for easy access and maintenance of the 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, analysis, etc. For presenting the data, SSPF includes a Graphical User Interface (GUI), which is configured from the plant models.

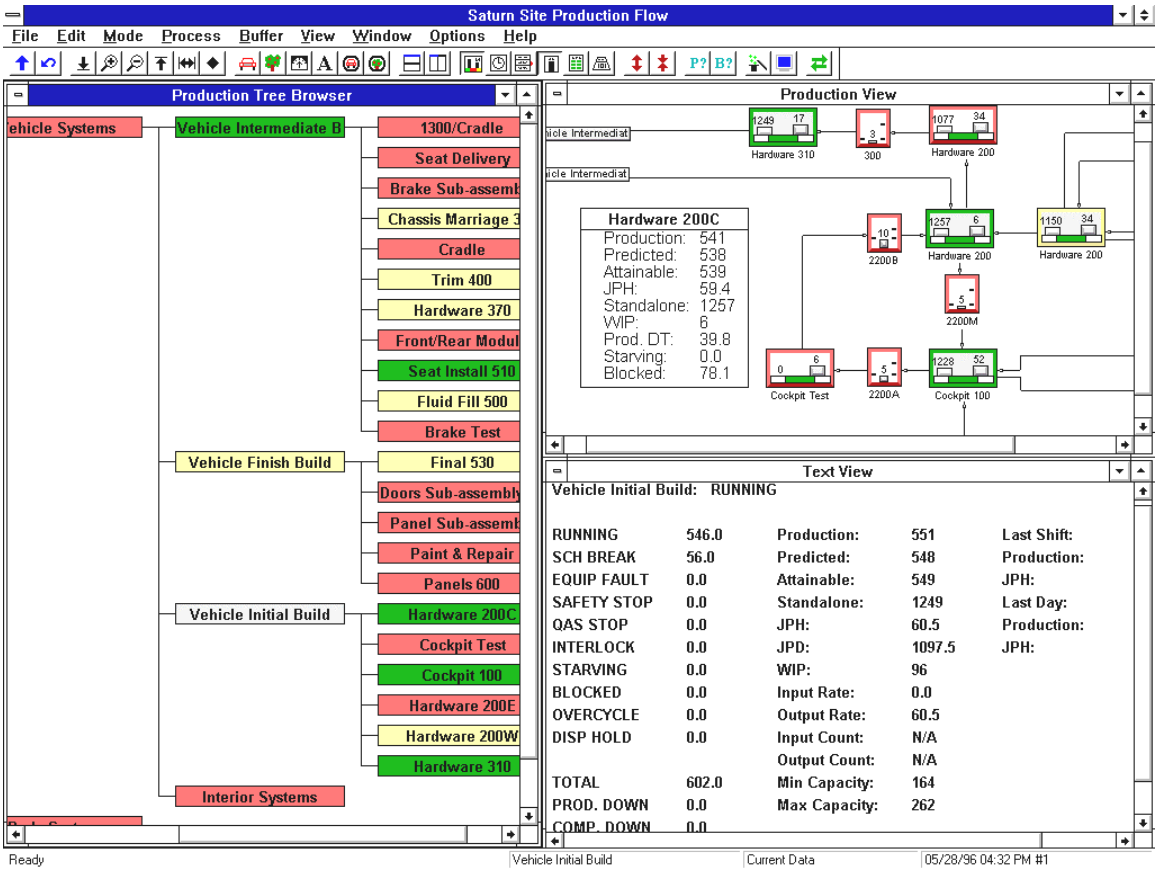

Figure 2. SSPF graphical user interface

Figure 2 shows the SSPF GUI with hierarchical navigation, drop down boxes and detailed textual report. On the left, the process hierarchy is shown, which allows the user to go to any section of the plant and examine it. On the right the GUI layout for `Vehicle Initial Build`, as synthesized from the models, and the textual report of production related data is shown.

B. 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.

The manufacturing plant is viewed as an aggregate of processes and buffers. Processes represent the operations required for making a car. 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 assembled), 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 and/or sub-assemblies that are produced by an upstream process before they are consumed by a downstream process.

To model the Saturn site in terms of its production processes and business organizations, a special modeling paradigm was developed that utilizes four kinds of models: (1) Production Models, (2) Organization Models, (3) Activity Models, and (4) Resource Models. Production models are used to represent the production flow at Saturn. 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 activities while resource models describe the allocation of SSPF activities to workstations. Here we briefly describe the first kind of models only.

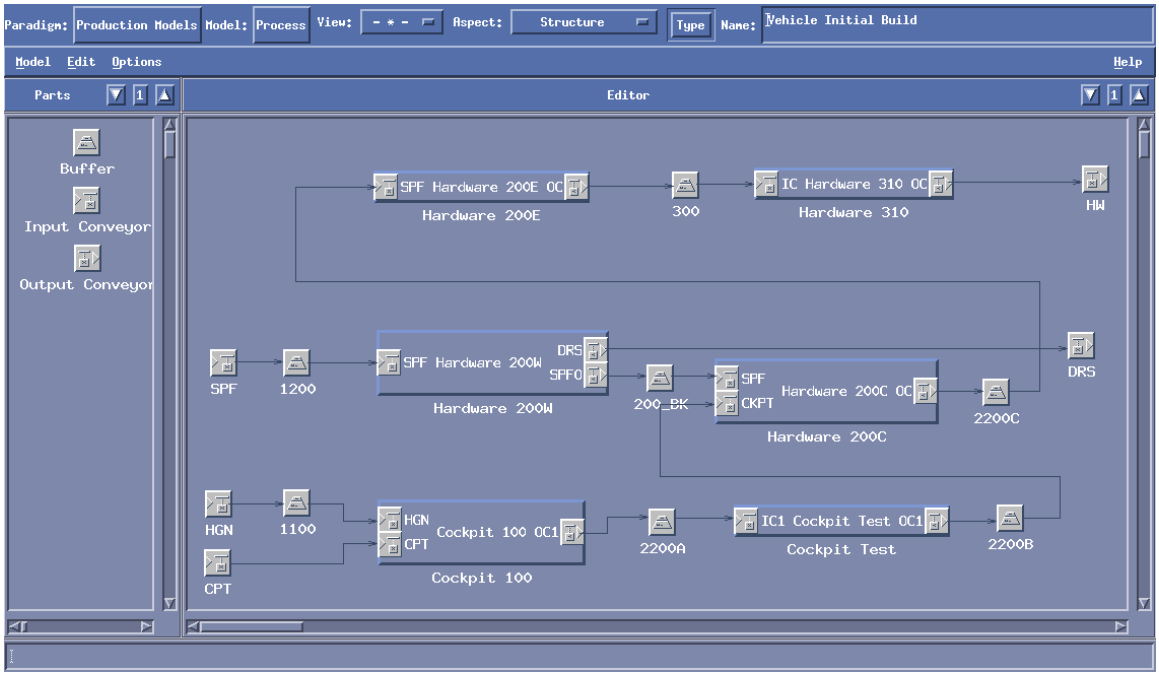

Figure 3. Structural aspect for Vehicle Initial Build

Production models hierarchically describe the production flow of Saturn Plant in terms of processes and buffers. A process represents a production entity in which production and/or assembly operations are performed. A process can be a leaf process, e.g., `Hardware 310`, or it may be an aggregate process, e.g., `Vehicle Initial Build`. Aggregate processes consist of other (leaf and/or aggregate) processes and buffers. Buffers represent the banks between processes, e.g., `300`, which is the bank between processes `Hardware 200E` and `Hardware 310` (see Figure 3). A process may have input and output conveyors. These act as interfaces to buffers and other processes and are used to connect banks and processes together.

C. SSPF Architecture and Component Generation

There are three main parts to the SSPF system: (1) The *Model-Integrated Programs Synthesis (MIPS) Environmen*t consists of the Visual Programming Environment (VPE) also referred to as the Graphical Model Builder (GMB) , Model Database and the Application Generator (AG); (2) the *SSPF Serve*r consists of the Real-Time Data Server (RTDS), Historical Data Server (HTDS), Cimplicity Interface, a Cimplicity project bridge (SSPFPB) (for PM&C), ODBC Interface and one or more MS/SQL Server and MS/SQL Database; and (3) the *SSPF Clien*t that consists of the Client Data Handler and the Client GUI. The Application Generator (AG) translates the SSPF models into the executable application. This is accomplished as follows.

1. Configurable run-time libraries and programs were developed, which get their configuration information from configuration files produced by the AG. These components implement generic functionalities (e.g. interface to the data acquisition system, user interface, etc.), that can be instantiated according to the contents of models.
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. 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 need to be performed.

D. SSPF 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, with about one-third of the plant being modeled. During 1996, the prototype was moved towards a production release. Some of the changes were necessitated by the integration process, but most were due to added functionalities. A large part of the effort after April 1996 was spent on building the complete models of the plant. The modeling phase involved consulting with Saturn personnel and putting this information into the models. SSPF was put into production release in the first week of August 1996.

We learned many lessons during our system integration efforts:

- 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. This allowed us to verify the application *and* the 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 that was required was a change in one configurable component followed by regeneration of the application. Without the use of MIC, trying to keep an application upgrade consistent for all the processes (and the buffers) would have been 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 idiosyncrasies of these implementations. Being able to capture this information in models also helped the integration effort considerably.

IV. The Integrated Test Information System

The Integrated Test Information System (ITIS) addresses the need to integrate diverse sets of information from distributed, heterogeneous data sources into a web-based metadata search and data retrieval application. (Here, metadata is defined as data used as index or search criteria when collecting sets of archived test data.) The approaches used allow for rapid generation and customization of test information systems that track changing user requirements. MIC technology is used to automatically synthesize complex test information systems using high-level models. Using MIC, the system can manage rapidly changing requirements for the interconnection of a wide variety of data sources, analyses, and their associated restrictions. The ITIS allows secure, restricted data access to geographically distributed engineering teams.

A. ITIS Modeling Paradigm

The ITIS modeling paradigm allows the creation of configurations that describe a group of users' view and access to archived data. The paradigm consists of two categories: Configurations and Library. Configurations contain all configurations currently included in the ITIS configuration database for access by authorized users. Library contains all components of a configuration and past configurations that are no longer active. Modelers create active configurations by dragging references to the Library's components into a Configuration model.

The GME graphical modeling tool, developed by ISIS, creates a visual representation of the paradigm's components and their relationships for creation and editing. Figure 4 depicts a Configuration model that contains the elements available when a user selects this configuration. These elements describe the location and names of archived data files, data

fusion modules and their operations, available search types, search parameters, any default search values, and output types available to user groups. Figure 5 illustrates a fusion model that contains an FFT operation. Figure 6 shows the selection of search parameters that will be displayed when a search of type “Test Engineer” is selected. Figure 7 details the specification of parameters from the metadata schema and any default values they may take on.

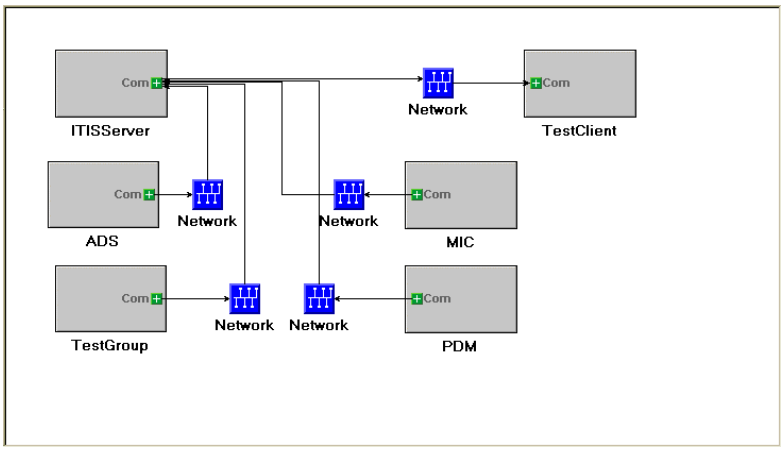

Figure 4. Configuration model contents

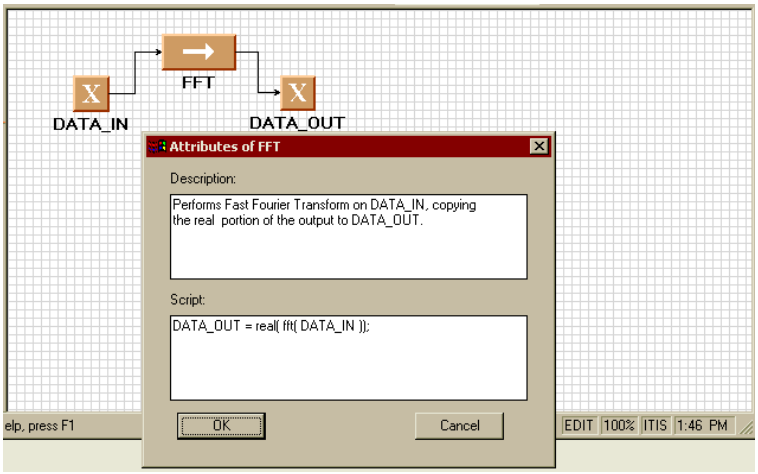

Figure 5. Fusion module with FFT operation

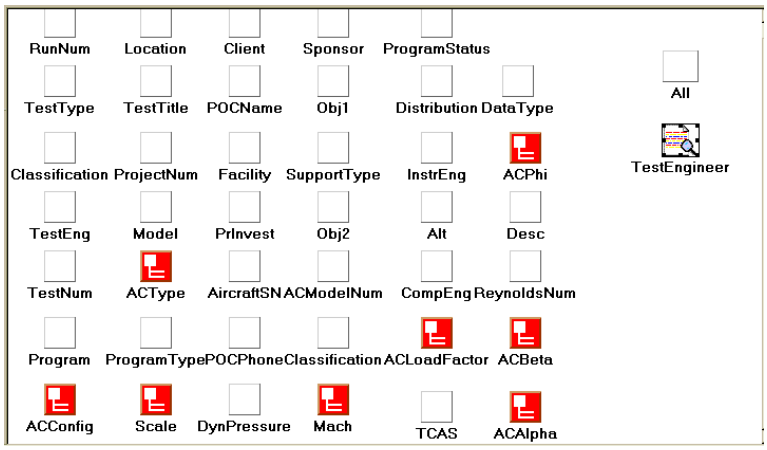

Figure 6. Search parameter selection

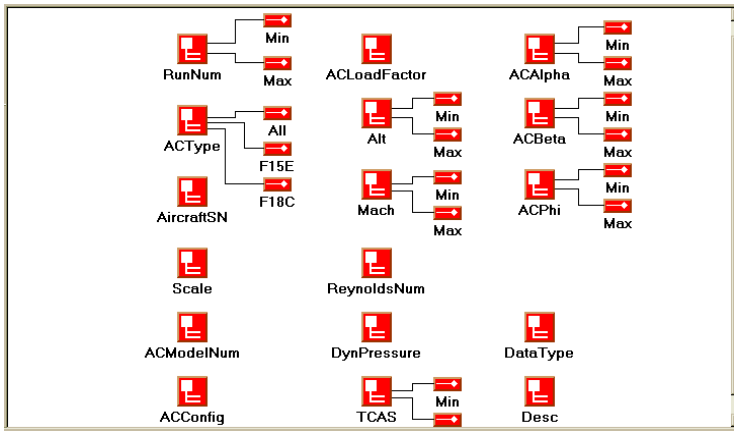

Figure 7. Parameter specification with default values

B. ITIS Configuration Database

By building a set of models corresponding to configurations of the ITIS application, a modeler populates a database schema corresponding to information captured in the models. This configuration database is then utilized by the server to verify any user requests against their modeled roles as well as generate HTML content that is configuration-specific.

C. ITIS Meta Database

Based on research by ISIS, AEDC designed a database schema to capture all test metadata that is generated for each test. The schema uses relational database design methods to link all information in a hierarchy of tables corresponding to project, test, and each AEDC business area. Through these relationships, searches may be restricted based on the actual archived data files. So, a user authorized to see data file X is unable to see matches to his/her criteria that correspond to data file Y. These access restrictions are modeled in the ITIS paradigm.

D. Server

The ITIS application utilizes Microsoft's Internet Information Server (IIS) with Active Server Pages (ASP) technology to deliver dynamic HTML content based on each user and the current request. The application is configured to only accept 128-bit Secure Sockets Layer (SSL) connections for encrypted client/server communications. The flexibility of ASP allow the creation and use of any Component Object Model (COM)-compliant object for use in generating client content.

A user must first authenticate himself to the server before entering the application. At AEDC, external customers access the site using SecurID where AEDC on-site employees use a simple username/password combination. Once authenticated, the server creates a session for the user that is used to track their use of the system. The user is first presented a list of configurations that his group has been modeled to access. For every page submission or request that the user makes, his user authentication, assigned session number, and selected configuration are matched to ensure he can only access permitted data.

E. Client

Users of the ITIS web-based application are required to access the website using either Microsoft Internet Explorer (version 4.0 or higher) or Netscape (version 4.6 or higher).

F. Data Sources

A variety of proprietary file formats have been created for storing test data in each AEDC business area. ITIS allows the plotting of data from different test data files on the same plot. This is accomplished by utilizing Microsoft's Universal Data Access strategy involving OLE DB technology. OLE DB is a set of COM interfaces to enable applications to have uniform access to data stored in DBMS and non-DBMS information containers. Using the ActiveX Template Library, we created custom OLE DB data providers that incorporate legacy file access code from AEDC. Through these data providers, we are able to retrieve and manipulate archived data in a similar manner for all file formats.

G. Data Services

Each proprietary file format also has a set of tools or routines normally used to analyze the data. In a similar manner, these routines may be wrapped as OLE DB data services to perform operations on the supplied data. For example, we have wrapped the MATLAB application to be a component in the ITIS application. Now, users may request operations modeled in the system to be performed on test data and view the output.

H. Data Output

Users may request test data in a variety of formats. The modeling environment allows the specification of which output modules are available for each particular configuration.

Each module is an ActiveX DLL that runs on the server. It retrieves the archived test data and then does any formatting or manipulation necessary to generate the output requested by the user. In the current system, output modules allow the plotting of data using a commercial-off-the-shelf JAVA-based charting package. Also, either ASCII-based tab-delimited files of selected data, or an entire archive file, may be downloaded to the client machine.

I. ITIS Status

The ITIS system was installed July 1999 at AEDC in Tullahoma, TN. Since installation, eight datasets have been added to the metadata for viewing through modeled configurations. Beginning in mid-September 1999, the Air Force Information Warfare Center (AFIWC) began evaluation of our design, and the system was installed at AFSEO/Eglin Air Force Base in Ft. Walton, FL. The system was also demonstrated to Boeing in October, and was installed at their St. Louis facility in early December, 1999.

V. Conclusions and Future Work

Model Integrated Computing shows the following advantages in the software and system development process: (1) It establishes a software engineering process that promotes designing 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 offering 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.

Using the metamodeling technology described in this paper, domain-specific modeling tools have been created, and have been in constant use for many years, in many engineering application areas and domains [12]. Our future work will focus on more efficient methods for mapping the abstract syntax of a metamodel onto the graphical idioms of the GME and improving the GME constraint manager, as well as continued research into interpreter specification and generation [13].

VI. Acknowledgements

Many concepts and techniques in our approach are based on groundwork done by a large community of modeling experts, along with academic and industrial researchers. The use of metamodels for defining modeling concepts and domains can be found in many engineering standards. For example, CDIF [14] proposes the use of the four-layer modeling approach. The static semantics of UML is specified using a similar approach, using UML as its own metalanguage [15]. Metamodeling is an idea that has been addressed in many research workshops and projects (e.g. [16] and [17]). Some of the relevant research activities and

industry efforts are related to integrating data from various sources (e.g. MetaData coalition [18]) as well as creating domain-oriented tools for building original types of models (e.g. DOME [19]).

The SSPF project has been supported by General Motors Corporation, Saturn Division. The ITIS project has been supported by the U.S. Air Force, Arnold Engineering Development Center. General MIC/MGA research has been supported, in part, by the DARPA/ITO EDCS program (F30602-96-2-0227) , the U.S. Army, NASA, Boeing, DuPont, Sverdrup, and Vanderbilt University. Their support is gratefully acknowledged.

VII. References

[1] Abbott, B., et al. Model-based approach for software synthesis. *IEEE Softwar*e, pages 42–53, May 1993.

[2] S. B. Ornburn and R. J. LeBlanc. Building, modifying, and using component generators. *Proc. of the ICSE-1*5, pages 391–404, Apr 1993.

[3] J. Sztipanovits, G. Karsai, “Model-Integrated Computing,” IEEE Computer, pp. 110-112, April, 1997.

[4] Nordstrom G.: "Metamodeling - Rapid Design and Evolution of Domain-Specific Modeling Environments", Ph.D. Dissertation, Vanderbilt University, 1999.

[5] Ganti, M., et al. An object-oriented software application architecture. *Proc. of the ICSE-1*2, pages 212–200, March 1990.

[6] Sztipanovits, J., et al. Multigraph: An architecture for model-integrated computing. *Proc. of IEEE ICECCS’9*5, pages 361–368, 1995.

[7] Karsai, G. A configurable visual programming environment: A tool for domain-specific programming. *IEEE Compute*r, pages 36–44, March 1995.

[8] Misra, A., et al. Robust diagnostics: Structural redundancy approach. *Proc. of Knowledge Based AI Systems in Aerospace and Industry, SPIE’s Symposium on Intelligent System*s, pages 249–260, 1994.

[9] Misra, A., et al. Diagnosability of dynamical systems. *Proc. of the Third International Workshop on Principles of Diagnosi*s, pages 239–244, 1992.

[10] Karsai, G., et al. Model-embedded problem solving environment for chemical engineering. *Proc. of IEEE ICECCS’9*5, pages 227–234, 1995.

[11] Ledeczi, A., et al. Modeling paradigm for parallel signal processing. *The Australian Computer Journa*l, 27(3):92–102, August 1995.

[12] Model-Integrated Computing documents. http://www.isis.vanderbilt.edu/projects/

[13] G. Karsai, et al., “Towards Specification of Program Synthesis in Model-Integrated Computing," Proceedings of the IEEE ECBS’98 Conference, 1998.

[14] CDIF Meta Model documentation. http://www.metamodel.com/cdif-metamodel.html

[15] UML Semantics, v. 1.1, Rational Software Corporation, et al., September 1997.

[16] Metamodeling in OO (OOPSLA'95 Workshop) October 15, 1995, http://saturne.info.uqam.ca/ Labo_Recherche/Larc/MetamodelingWorkshop/metamodeling-agenda.html

[17] 43rd Annual Meeting of the International Society for Systems Sciences, at the Asilomar Conference Center, Pacific Grove, California, June 26 to July 2, 1999, http://www.isss.org/1999meet/sigs/sigmodel.htm

[18] MetaData coalition. http://www.mdcinfo.com/

[19] Dome Official Web Site, Honeywell, 2000, http://www.src.honeywell.com/dome/

[20] Generic Modeling Environment documents, http://www.isis.vanderbilt.edu/projects/gme/Doc.html