Model-Integrated System Development: Models, Architecture, and Process

Gabor Karsai, Amit Misra, Janos Sztipanovits, Akos Ledeczi, Michael Moore
Department of Electrical and Computer Engineering
Vanderbilt University
Nashville, TN 37235 USA
{gabor,misra,sztipaj,akos,msm}@vuse.vanderbilt.edu

Abstract

Many large software systems are tightly integrated with their physical environments and must be adapted when their environment changes. Typically, software development methodologies do not place a great emphasis on modeling the system's environment, and hence environmental changes may lead to significant and complicated changes 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 model-based systems. This framework has been successfully used in various projects. One of these projects, a site-production flow visualization system for a large manufacturing operation, is analyzed in detail.

1. Introduction

Every software practitioner knows the difficulties of maintaining a large software system that is tightly coupled to a physical environment which is periodically undergoing configuration changes. In fact, it is probable that these systems are rather the rule than the exception.

Naturally, there are a lot of causes for this problem, but there seem to exist some aspects that are more prevalent than others. Let us look at 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 enforcing some level of production control. It is a fact of business that the plant changes and evolves necessitating changes and upgrades in the software system. One has to change the database schema, recompile applications, reconfigure data acquisition systems, etc. just to maintain existing functionalities. These many-faceted activities involve diverse software engineering issues, and the maintenance of the system becomes a highly non-trivial activity. It is interesting to note, that software engineers must become plant engineers (up to a certain degree, of course) in order to understand how the plant works. While it is important from a business point of view for the software engineers to become familiar with the problem domain, this results in a duplication of effort, since the plant engineers are already experts in their domain. Furthermore, it seems that the software configuration changes should be closely coupled to configuration changes in the plant, i.e. the environment of the software.

In this paper, we advocate an approach that addresses this problem. First we describe the approach on an abstract level, then 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 objective evaluation of the work. Finally, we relate to other development approaches and conclude with showing some future aspects of the development process.

2. Model-Integrated Development

Recognizing the need for software systems that evolve and are maintained in accordance with their environment, we propose the extensive use of *models* in the development process. The use of models in software development is not

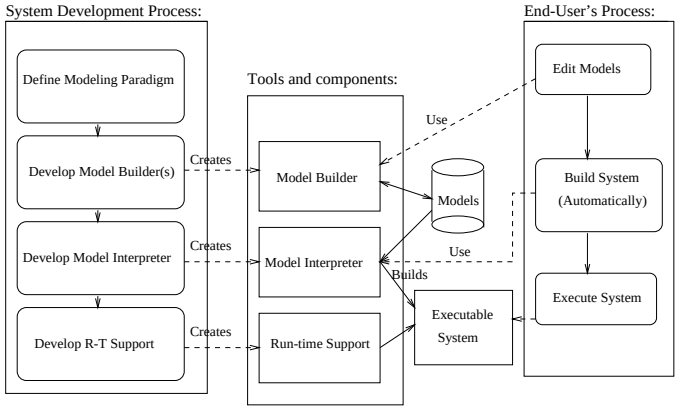

Figure 1. Model-Integrated System Development Process

a new idea. Various analysis and design techniques (especially the object-oriented ones) build models of the system before implementation, and model its environment as well. However, we propose to extend and specialize the modeling process, so the models can be more tightly integrated into the system development cycle than with traditional techniques. The process supporting this activity can be called Model-Integrated Development (MID), and it results in a model-integrated system (MIS).

In an MID process, the models play many central and essential roles, including:

- they describe the system's environment,
- they 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 MID 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. These model editing tools can later be 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. The result of the model editing is a set of domain-specific models, that are typically kept in a 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 an executable system, 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 it. In the most general terms: the model interpreters are responsible for mapping domain-specific models into run-time components. Typically, 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. Note that model interpreters perform an automatic system generation by instantiating and customizing the generic components. The key in the process is that the models are domain-specific, and do not (necessarily) include software concepts.

At the process level, in MID, we have two interrelated processes: (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, and 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 one is done initially by them, but later by the end-users.

To summarize, with MID the system is created through the following steps: (1) definition of a modeling paradigm, (2) development of the model builder (editor) environment, (3) development of the model interpreters, (4) development of the run-time support system. The product of this process is a set of tools: the model builder, model interpreter(s) 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 automatically generate the system, and (3) execute the system. The key aspect of the development process is that *domain-specific models are used in building the application*, thus it can be re-generated by the end-users.

The MID approach can be contrasted with current development practices as follows. As opposed to developing a highly specialized product, in MID we want to understand an entire class of problems related to the domain. As opposed to developing a specific application, we try to develop first the domain-specific tools to build that application, then use them (or have them used by the end-users) to generate the application. Many of these ideas can already be found in other large-scale packages [2]. What is different here is that, in addition to making the models themselves available for

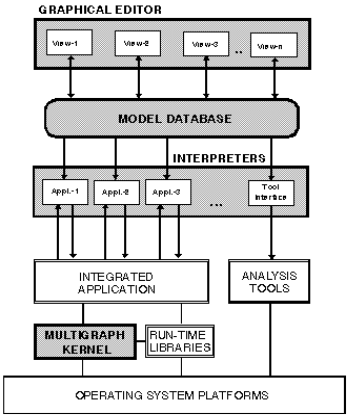

Figure 2. MGA Components

the end-users, we want to make explicit use of the models in generating applications.

It seems that MID 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 Architecture*(MGA)[10] that provides a highly reusable set of generic tools to do MID. 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 the MGA we use a generic Visual Programming Environment (VPE)[5] 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, and how the database schema is organized. 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 adopted to various application domains has enabled us to use it in widely different projects during the last 10 years. A selected list of systems developed using the MGA is as follows:

DTool[7] and **RDS**: The MGA was used as the software framework for a model-integrated robust real-time diagnostic system (RDS) and a diagnosability and testability analysis tool (DTool). The 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[8], 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, such as, for instance, analysis of process operations. The activity models are automatically translated into executable software. The IPCS system is actively used at Du Pont Old Hickory,TN plant for the development of commercial applications, including monitoring, sensor data validation, on-line process simulation, and process diagnosis[4].

CADDMAS: Computer Assisted Dynamic Data Monitoring and Analysis System. 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][6]. 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.

3. A Practical System: SSPF

In this section, we describe the functionalities, requirements, design and development of the application of MID towards providing a problem-solving environment and decision support tools in the context of discrete manufacturing operations at Saturn Corporation. 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 which presents consistent and pertinent

information, and analysis and decision support services that are 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). 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 facility 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 3 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.

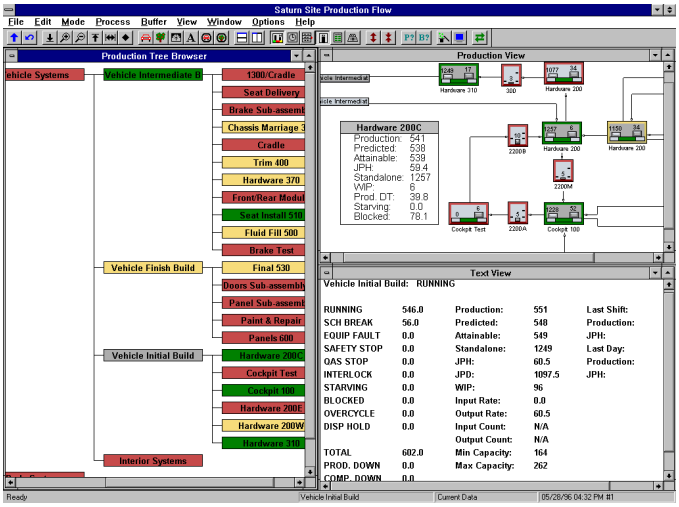

Figure 3. SSPF GUI

3.2. 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. 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.

Production Models Production models hierarchically de-

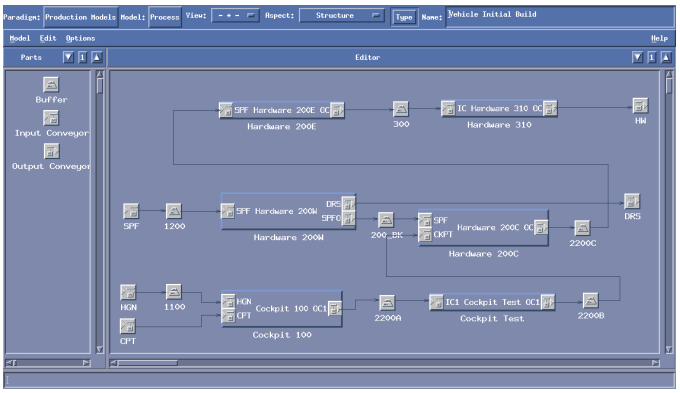

Figure 4. Structural Aspect for Vehicle Initial Build

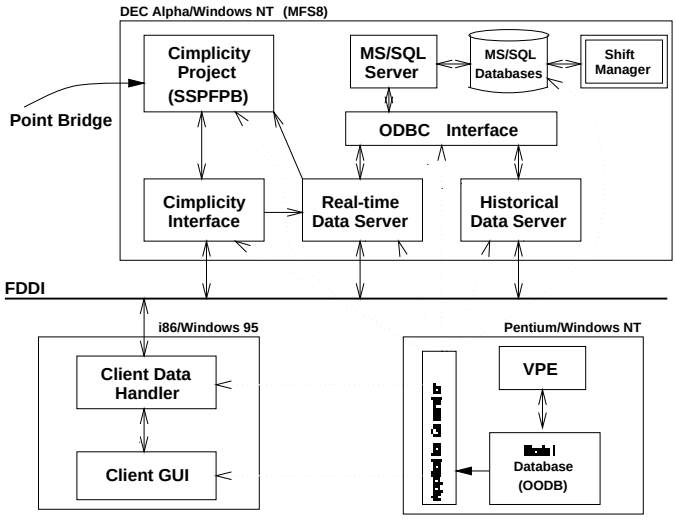

Figure 5. SSPF Architecture

scribe 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 4). A process may have input and output conveyors. These act as interface to buffers and other processes and are used to connect banks and processes together.

3.3. SSPF Architecture and Component Generation

There are three main parts to the SSPF system as shown in Figure 5:
Model-Integrated Programs Synthesis (MIPS) Environment: This consists of the Visual Programming Environment (VPE) also referred to as the Graphical Model Builder (GMB) , Model Database and the Application Generator (AG).
SSPF Server: This 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.
SSPF Client: This consists of the Client Data Handler and the Client GUI.

The Application Generator (AG) translates the SSPF models into the executable. This was 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.

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, with about one-third of the plant modeled. During 1996, the prototype was moved towards a production release. Some of the changes 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 the plant. The modeling phase involved consulting with Saturn personnel, and then 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 MID 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 it required was a change in one configurable component followed by re-generation of the application. Without the use of MID, trying to keep an application upgrade consistent for all the processes (and the 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 idiosyncrasies of these implementations. Being able to capture this information in models also helped considerably.

5. Other Approaches

The benefits of software modeling and generating software from models (to facilitate rapid development) has been known for some time. The importance of domain modeling in the software development process has been recently recognized[3][2].

Many of the concepts developed in [3] are present in our toolset, the MGA. Just like in [2], 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[9]: they instantiate and configure generic components and templates, although our process may be executed at run-time.

6. Conclusions

Model-Integrated Development 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.

7. Acknowledgements

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

References

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

[2] Ganti, M. and et al. An object-oriented software application architecture. *Proc. of the ICSE-12*, pages 212–200, March 1990.

[3] Iscoe, N. and et al. Domain modeling for software engineering. *Proc. of the ICSE-13*, pages 340–343, May 1991.

[4] Karsai, G. and et al. Model-embedded problem solving environment for chemical engineering. *Proc. of IEEE ICECCS'95*, pages 227–234, 1995.

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

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

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

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

[9] S. B. Ornburn and R. J. LeBlanc. Building, modifying, and using component generators. *Proc. of the ICSE-15*, pages 391–404, Apr 1993.

[10] Sztipanovits, J. and et al. Multigraph: An architecture for model-integrated computing. *Proc. of IEEE ICECCS'95*, pages 361–368, 1995.