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 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 software 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.

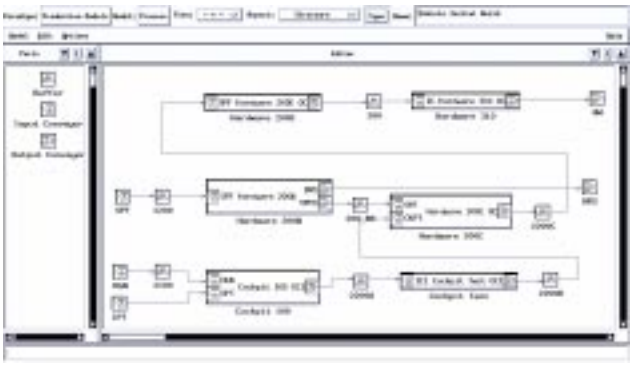

Figure 5: Production Model for Vehicle Initial Build

cesses and business organizations, the SSPF modeling paradigm was developed. The modeling paradigm utilizes many different kinds of models, the salient among which are: (1) Production Flow Models (an example is shown in Figure 5) which capture the processes, buffers and the flow of material, (2) Organization Models, which represent the business organizations at Saturn, (3) Data Acquisition Models which capture the parameters for production data acquisition, (4) Screen Models which allow the user to develop custom screens for visualization.

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. In the first part of 1996, the prototype was moved to a production release version.

SSPF has been in operation in the Saturn plant since August, 1996 and has been an integral part of a highly successful throughput improvement program. The current implementation is primarily focused on analysis of historical data. The toolset is being extended to the planned features of decision support.

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

- Production throughput at Saturn has markedly improved through the use of SSPF in conjunction with a focused business process for improved throughput. The exact amount of improvement cannot be reported due to competitive reasons.
- Prior to the use of Model-Integrated Computing approach, MES approaches toward issues concerning throughput improvement were so resource intensive that solutions were not responsive to business requirements. High costs of MES were exhibited as missed opportunities in improvement of throughput. Now with the MGA approach, these domain specific issues concerning throughput improvement are successfully being addressed.
- 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. 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 re-inpterpret them. 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 due to the use of MIC. Without the use of MIC, trying to keep an application upgrade consistent for hundreds of processes and buffers would be a very difficult and costly task.
- Being able to capture the heterogeneity in data points as part of the models also helped considerably in addressing this issue.

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

and (4) Evolution.

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

- Tight integration with the plant
- Real-time performance
- Distributed system with large number of clients
- Heterogeneous platforms
- Multiple custom and COTS software components
- Robustness

Integration. The issues faced during integration of SSPF with Saturn systems are:
Data collection : There is 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 hardware/operating systems already in use at Saturn and to interface with existing software packages.

Maintenance. Due to the continuous, ongoing improvments in the product, the production processes and the engineering and business practices, the Saturn plant undergoes changes continually (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 current form, SSPF is a data collection, logging, retrieval, distribution and visualization service. In the future, however, SSPF will have many more custom developed and COTS 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 Architecture

The SSPF Architecture, shown in Figure 4 has three levels : (1) the Meta Level, (2) the MIPS Level and (3) the Application Level.

At the Meta Level, the modeling paradigm (discussed below) for SSPF is defined, which consists of a declarative representation of the concepts and relationships used to model the Saturn plant.

From the modeling paradigm, the tools at the MIPS Level are customized. These tools consist of the model editor, the model database and the model interpreter.

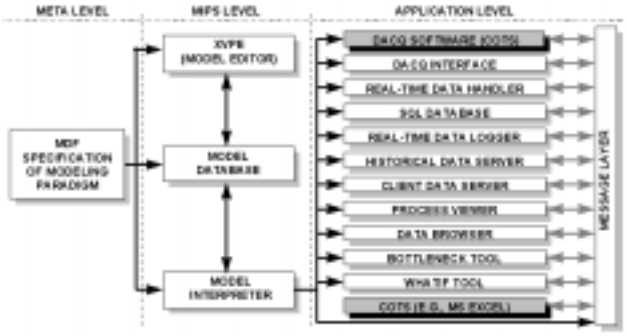

Figure 4: SSPF Architecture

The model interpreter is responsible for transforming the plant models into configuration information, C++ code and SQL scripts for the components in the Application Level which fulfill the SSPF functionalities described above.

The Application Level consists of many components, which include data acquisiton components, real-time and historical data servers, data loggers, viewers and COTS. They will not be described here for sake of brevity. The Message Layer in the Application Level, itself configured from the models, is responsible for data packaging and transfer between the applications components by using a variety of interfaces including sockets, OLE, APIs, etc.

Since all the above custom developed components, and the interfaces to COTS components are configured and/or generated from the models, any changes in the plant are handled by changing the models and re-interpreting them. If additional functionality is desrired, it requires only development of a component or interface which can be configured from the models. This results in easy and cost-effective maintenance and evolution of the system as our experiences (discussed below) have shown.

3.4. SSPF Modeling Paradigm

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. Buffers lie between processes and hold parts that are produced by an upstream process before they are consumed by a downstream process. 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, sub-assemblies, etc.) through the processes and buffers.

To model Saturn Site in terms of its production pro-

vide a suite of decision support tools that are focused on improving throughput. The intent of the system is to capture metrics and decision methods that represent best practices and have these globally used throughout the Saturn organization. Conceptually this is built on visual controls, a method that is important for control of a large organization such as Saturn. To understand the flow of production through the manufacturing site, a view of both current status and of historical information is required. SSPF is a client/server system in which the server components collect, process, transform, archive and provide access to production data to over 300 clients across the site (on plant floor as well as business offices). It is necessary that all participants in the manufacturing process have equal and consistent access to production flow data regardless of role. A plant manager and a car builder must all have the same view. There are approximately 2000 people involved directly in the manufacturing process at any one time. It is estimated that there would be 300 simultaneous users. SSPF has an architecture that easily accommodates a very large number of users without affecting performance of the system.

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). 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.

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 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. SSPF stores the data in a structured manner using the a relational database (Microsoft SQL/Server). 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.

Figure 3: SSPF Viewer

Data Distribution. SSPF is required to provide current (real-time) and historical data to custom developed and COTS visualization, analysis and decision support tools, such as bottleneck analyzers, spreadsheets, statistical analysis tools, web browsers, etc. It is required to provide seamless access to the data and become an integral part of the data warehouse at Saturn, thereby facilitating informed decision making through the use of many different tools.

Data Visualization. Visualization of real-time and historical production flow is an important component of any business decision making. For visualizing the data, SSPF includes a viewer (see Figure 3), which is configured from the plant models. The viewer 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, etc.

Bottleneck Analysis and Other Problem Solving Activities 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. Identification of bottlenecks and other analyses such as 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.

3.2. Design Challenges and Issues

The design challenges and issues 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,

bility 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-dataflow based run-time kernel: the Multigraph Kernel (MGK). MGK facilitates the dynamic creation of

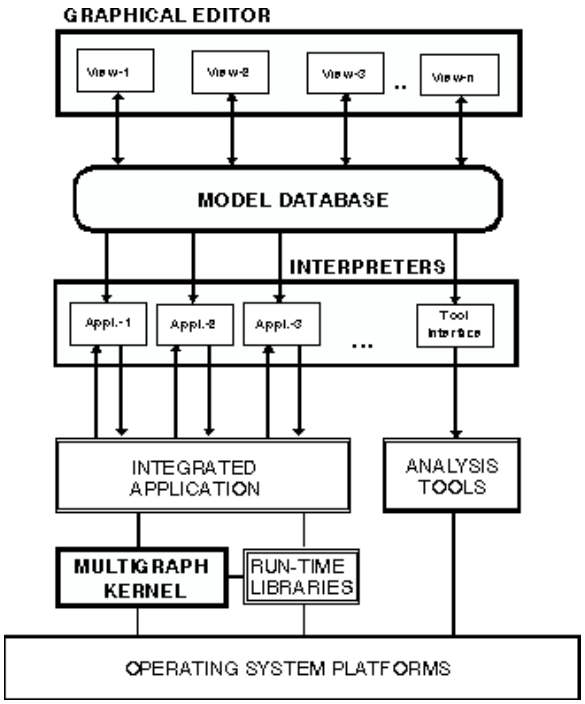

Figure 2: MGA Components

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 system, which was developed using MIC.

3. A Model-Integrated 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 Corporation.

The Measurement and Computing Systems Laboratory at the Department of Electrical and Computer Engineering of Vanderbilt University in cooperation with the Saturn Corporation of General Motors has introduced the MIC technology in a MES architecture developed for Saturn's site-wide discrete manufacturing operations. Saturn is a flexible manufacturing plant of GM, producing over 300,000 small-size cars per year. The Saturn Site Production Flow (SSPF) system provides a problem solving environment and decision making support by facilitating structured and integrated production data collection, archival, visualization and analysis using custom developed and COTS software components. SSPF is intended to pro-

stallation (the cost of installing and maintaining an integrated MES solution, using traditional software technology, is estimated to run between 400,000 and 1 million dollars).

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, either 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.

Proposed new solutions introduce state-of-the-art object- oriented software technology, 'plug-and-play' software architecture which mitigates but does not solve the core problem: keeping the MES architecture and components consistent with the changing manufacturing processes. The primary objective of our research has been to provide an evolutionary MES architecture which supports the automatic reconfiguration of all MES components with changes in the manufacturing processes.

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.

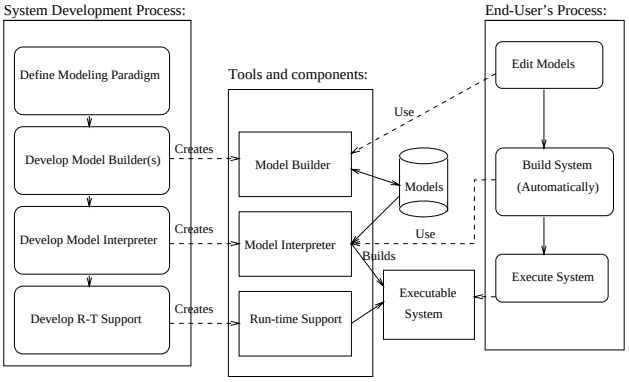

Figure 1: Model-Integrated System Development Process

2. Model-Integrated Computing

Recognizing the need for software systems that evolve with their environment, we advocate 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 programma-

EVOLUTIONARY DESIGN FOR MANUFACTURING EXECUTION SYSTEMS *

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

E. Long
Saturn Corporation
Spring Hill, TN, USA
LNUSSAT.ELONG1@gmeds.com

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 present a detailed experience report and conclude with comparisions with other approaches.

Keywords

manufacturing execution systems, model-integrated computing, end-user programming, component-based software integration

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

1. Introduction

Manufacturing Execution Systems (MES) are middle-level information systems that bridge the gap between factory-floor information systems, which focus on the operation of production equipment and on the control of processes, and front-office information systems dedicated to accounting, forecasting, and other resource planning activities, or with design and engineering systems. MES applications track and manage all aspects of a job on the shop floor, at any point in the production cycle, in near real-time. For example, they identify bottlenecks and material shortages on the shop floor, and they provide up-to-the-minute process performance results along with comparisons to past performance and to projected business results.

MES software is prototypical for computer-based systems applications: the software is tightly integrated with a dynamic, continuously changing manufacturing environment, it implements critical functions in the manufacturing plant under (soft) real-time constraints, it is built from a heterogeneous set of COTS and custom components installed on a large-scale distributed computing platform, and it is a front-line production critical software subject to high reliability requirements.

Today's MES solutions are subject to complexities that make them difficult to implement and integrate and, often, even more difficult to modify and upgrade. The introduction of Model-Integrated Computing in MES has resulted in an evolvable, expandable, highly reliable system which has been designed, implemented and installed for less than the cost of a single MES in-