MULTIGRAPH: An Architecture for Model-Integrated Computing

Janos Sztipanovits, Gabor Karsai, Csaba Biegl, Ted Bapty, Akos Ledeczi and Amit Misra

Department of Electrical and Computer Engineering
Measurement and Computing Systems Laboratory
Vanderbilt University
Nashville, TN 37235

Abstract

The design, implementation and deployment of computer applications tightly integrated with complex, changing environments is a difficult task. This paper presents the Multigraph Architecture (MGA) developed for building complex embedded systems. The MGA is a meta-level architecture which includes tools and methods to create domain specific model integrated program synthesis environments. These environments support the integrated modeling of systems independently from their implementation, include tools for model analysis and application specific model interpreters for the synthesis of executable programs.

1. Introduction

In the rapidly expanding category of computer integrated systems (CIS), functional, performance, and reliability requirements mandate a tight integration of "information processing" and "physical environment". In CIS's, embedded computer applications are essential to the overall system performance. Several characteristics of the CISs represent significant challenge for the underlying technology:

- *Functionalities implemented by physical components and software are inextricably combined*. Consequently, the design process needs to employ tools and methods that allow the tight integration of the hardware and software development cycle. The problems of HW/SW codesign have been increasingly recognized and investigated in the last five years [1].
- *Use of design-time models in system operation.* The ever increasing computing power enables the application of sophisticated model-based methods in monitoring, control, diagnostics and fault recovery. The use of models in system operation require maintaining the consistency of the software and its environment during the full lifetime of CISs.
- *Evolutionary systems.* Many of the CIS's, such as space systems or computer-integrated manufacturing systems have several decade long lifecycle. During this period both the physical hardware and the software components evolve - in many cases parallel with the regular system operation. For example, the International Space Station Alpha (ISSA) is an evolutionary system by design; the tools and methods have to be prepared to follow changes in the hardware configuration and software capabilities.

The tight integration of the software and its physical environment has profound impact on the nature of the software technology to be applied. The reason can be best explained by Brook's argument, which remains valid eight years after its publication: the essential complexity of large-scale software systems is in their "conceptual construct" [2]. In CIS's, the conceptual construct of the software is combined with the conceptual construct of its environment, therefore the methods and tools developed for managing complexity must cover both sides.

Recent developments in software and systems engineering indicate that the use of models have great potential in answering these challenges. Multiple-view models with rigorous formal semantics can support and integrate design, analysis, implementation and operation activities, such as: (a) capture the conceptual structure of the design [3], (b) trace process and product information [4], (c) analyze properties of the system [5], and (d) synthesize software from the models [6].

The Multigraph Architecture (MGA) has been developed as a model-integrated program synthesis (MIPS) environment for large-scale embedded applications in manufacturing, instrumentation, monitoring, and fault detection, isolation and recovery (FDIR) systems. In these and similar applications, system designers and operators view the software as an implementation method of functionalities (controllers, monitoring and diagnostic systems, simulators, etc.), which are integral parts of a complex, dynamic environment.

This paper provides an overview of the MGA and describes the role of a meta-level architecture in the technology of complex computer systems. Companion papers [7][8] present applications of MGA in several major projects in manufacturing and large-scale signal processing.

2. Background

The MGA research shares ideas and builds on results of several important directions in software engineering. The *model-based approach* has become a particularly rich, productive research area in object-oriented software engineering, and has already made significant impact on today's software engineering practice. One of the fundamental goals of the research in object modeling is to identify general modeling paradigms to be used in the design and analysis of complex systems. Different approaches are proposed for modeling functional structure, static and dynamic behavior, physical structure, inter-object communication using multiple views, finite state machines, dataflow modeling and other abstractions [9].

Research on *software reuse* focused much attention on domain specific modeling paradigms. The underlying assumption has been that large-scale software reuse is based on common domain characteristics expressed by domain specific models and domain specific software architectures (DSSA). Projects related to ARPA's DSSA Program and the Software Technology for Adaptable Reliable Systems (STARS) program generated many results in this area. One of the key conclusions of the research in reuse is to emphasize composition instead of decomposition [10][11].

The MGA has evolved as a framework for the model-integrated construction of large-scale, embedded computer applications. The following properties of these applications have had significant impact on the evolution of the architecture:

- The *application domains* are typically dominated by some mature engineering discipline. The modeling paradigms are not "negotiable": instead of simplified, "easy-to-use" programming models, the users need to be supported by rich, domain specific concepts, relations, and composition principles routinely used in the field.
- The *scope of modeling* is determined by the problem and related functionalities and not by their implementation. The modeling paradigms are typically not narrowed to "software implemented" functionalities, but cover the relevant parts of the "environment" as well.
- The conceptual structure of domain models does not necessarily reflect the structure (architecture) of the related application. The relationship between the two (often called problem space and solution space) can be quite complex. The effect of small changes in domain model specifications may ripple through several software modules, and may even change the composition of the application. Additionally, domain model changes may occur during system operation evoked by end-user requests or changes in the environment. This necessitates system reconfiguration while the application is running [12].

3. Functional Components of the Multigraph Architecture

The MGA provides a unified software architecture and framework for building model-integrated program synthesis (MIPS) environments. The MIPS environments are domain specific, and include tools for: (1) building, testing, and storing domain models, (2) transforming the models into executable programs and/or extracting information for system engineering tools, and (3) integrating applications on heterogeneous parallel/distributed computing platforms. The MGA-based MIPS environments have the following functional components [6,12] (Figure 1):

Graphical Model Builder (GMB): The modeling paradigms include concepts, relationships, model composition principles, integrity constraints, and representation formalisms that are accepted and used in the application domain. Domain models are constructed by the GMB tool which provides a customizable model building environment for domain experts. It enforces domain specific constraints during model building, uses domain specific graphical formalism, and supports checking the models using consistency and completeness criteria. The latest version of GMB is the Visual Programming Environment (XVPE) [13].

Model Database: The model database stores the complex, multiple-view domain models. In the last MGA implementations the model database function is implemented using object-oriented databases (OODB).

Model Interpreters: Model Interpreters synthesize executable programs from domain models, and generate data structures for systems engineering tools that perform various analyses of the system to be built [6,12]. Since the model interpreters capture the relationship between the problem space and solution space, they are specific to the domain modeling paradigm and to the type of applications to be generated. In the FDIR domain [5], for example, different model interpreters generate the diagnostic system, fault detection system, and operator interface components of the application from the same integrated domain models. Similarly, a different model interpreter generates input for system engineering tools that analyze the diagnosability of the design.

Multigraph Kernel (MGK): The executable programs are specified in terms of the Multigraph Computational Model (MCM). The MCM is a macro-dataflow model which represents the synthesized programs as an attributed, directed, bipartite graph. The MGK is a runtime system for

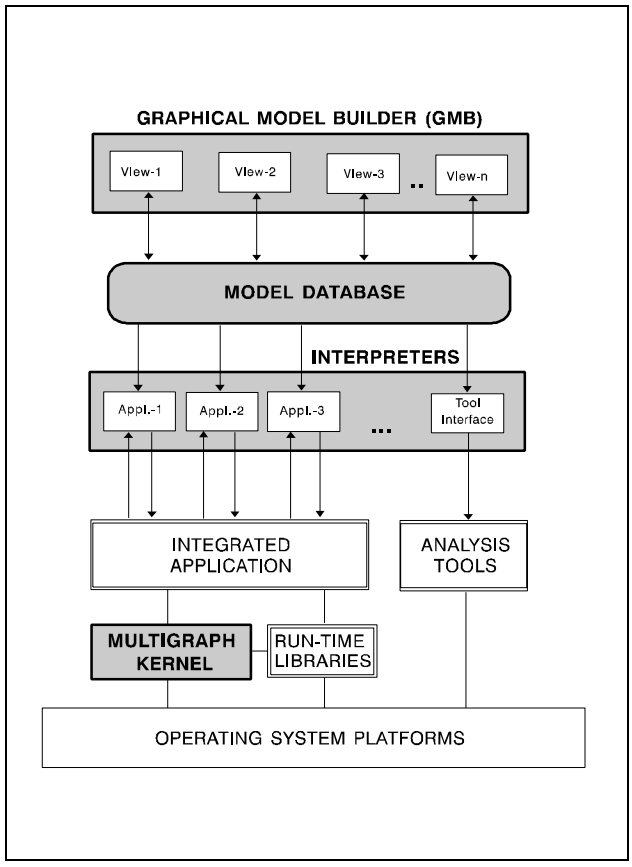

Figure 1: Multigraph Architecture

the model, and provides a *unified system integration layer* above heterogeneous computing environments including open system platforms, high performance, parallel, distributed computers and signal processors [6,12]. The elementary computations scheduled by the MGK, are carefully defined reusable code components that are part of application specific run-time libraries. The model interpreters have two options for synthesizing applications: 1) building/changing the structure of the dataflow graph, or 2) setting/changing parameters of the elementary computation blocks. The MGK is implemented as an overlay above operating and communication systems. A unique capability of the MGK is its *support of the dynamic reconfiguration of the executing system* [6,12].

Currently, the construction of an MGA-based MIPS environment for a new domain includes the following steps:

Step 1. *Definition of the modeling paradigm.* This step requires understanding of the concepts, relationships, model composition principles, and integrity constraints of the domain.

Step 2. *Customization of the model building tool and the model database*. Customization of the GMB tool requires the following information: (a) assignments between modeling concepts and graphic symbols, (b) assignments between graphic operations and model composition principles, (c) assignments between domain specific modeling constraints and admissible graphic operations. The assignments are represented in the Editor Definition Language (EDL) of XVPE [13]. Customization of the model database requires the specification of database schema using the Object Description Language (ODL) of the OODB.

Step 3. *Specification and implementation of model interpreters.* Model interpreters perform a mapping between "domain model objects" and "run-time objects". Domain model objects are created during model building and they are accessible in the model database. Run-time objects are described in terms of the Multigraph Computational Model. Specification of this mapping depends on the type of the application to be synthesized. Different functional components of the executable system (e.g. "simulator", "diagnostic system", "signal processing system", etc.) are generated by unique model interpreters.

Step 4. *Implementation of core run-time libraries.* Run-time libraries typically consist of software modules of subroutine-size, with standard interfaces to the Multigraph Kernel. Due to their small size, individual modules can be easily implemented and tested. The resulting code is re-usable in different application domains.

In summary, MGA is an infrastructure for building MIPS environments. It provides a framework for modeling, model representation, model interpretation and execution, and includes tools and run-time system components customized to specific domains. Currently, MGA-based tools are supported on standalone and networked Unix workstations (Sun, HP9000, IBM6000, SG), PC's, and distributed memory multiprocessors (networks of Transputers and TI-TMSC40-s).

4. Levels in the MGA

In its current configuration, the MGA has three levels. Since the higher levels include tools and methods to create systems on the lower levels, MGA is a meta-level architecture. An overview of the levels in MGA are shown in Figure 2.

Going from the bottom up, the MGA levels include the following components:

4.1 Application Level

The application level refers to synthesized applications, i.e. monitoring, control, diagnostics, simulation and other

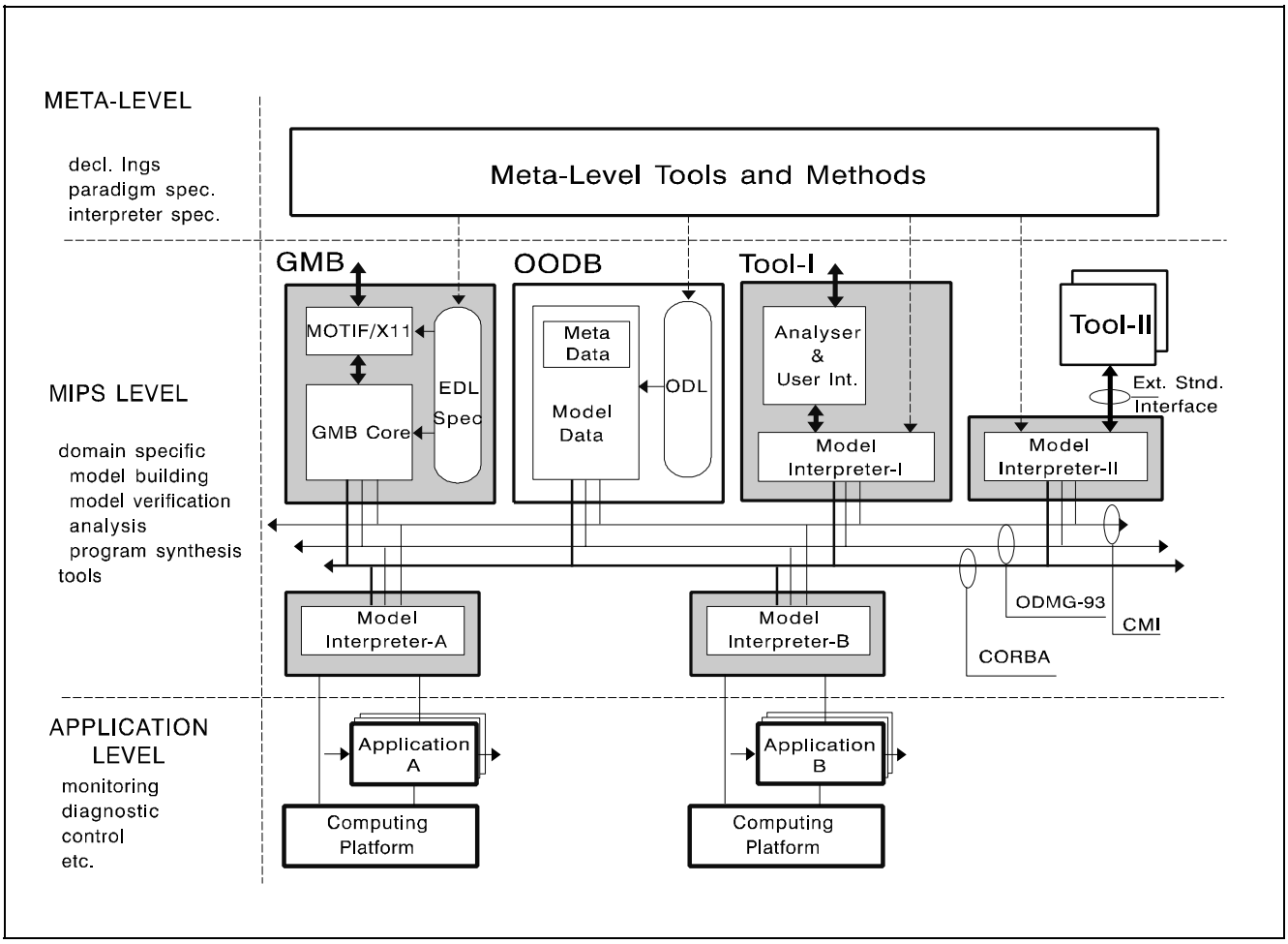

Figure 2: Levels of MGA

systems. These systems are generated by model interpreters which are common for a particular application system *category* (e.g. dynamic simulators using a particular solver in the run-time system, etc.) and which are part of a domain specific MIPS environment. The properties of the generated applications (parameters and/or structure) are determined by the relevant characteristics of models that are built by the MIPS tools.

4.2 MIPS Level

The MIPS level comprises the customized versions of the generic MGA tools. It has a modular internal structure, which includes the *Graphical Model Builder (GMB), Model Database (OODB), tightly coupled analysis tools (TOOL-I), loosely coupled analysis tools (TOOL-II)* and *Model Interpreters.* The function of the GMB, Model Database, and Model Interpreters have been described before. The analysis tools facilitate various systems engineering analyses during design time. *Tightly coupled tools* embed an MGA Model Interpreter which accesses models in the Model Database and synthesizes data structures for the analyzer (e.g. DTOOL [5]). *Loosely coupled tools* are external systems (from the point of view of MGA), with some given external interface. Several simulation and analysis tools have unique model definition languages (e.g. ASPEN+ used in chemical engineering, or SPN Stochastic Petri Net packages used in performance analysis of parallel configurations). These tools are interfaced to MGA through special model interpreters providing a two-way translation between the unique model representation language of the tools and the internal representation forms of MGA. We have successfully used this technology for the following packages: ASPEN+ (steady-state simulation), SPEEDUP (dynamic simulation) [7], SPNP, and Vantage (DuPont's real-time database system) [7].

The MGA tools on the MIPS level communicate via a multi-layer interface:

1. *Database Access Layer (ODMG-93):* All of the MGA components access the Model Database. We have chosen

object-oriented databases as the primary model database for MGA, because their services satisfy the needs of building and managing large-scale model databases. The Database Access Layer supports the definition, creation, and manipulation of objects stored in the Model Database. We have adopted the ODMG-93 standard for this layer which is widely supported by the OODB vendors [14]. The ODMG-93 compliant OODB-s provide services of persistence, transactions, recovery, and concurrent sharing for application objects on all levels of granularity. The ODMG-93 compliance assures that commercial off-the-shelf database systems can be used as Model Database, and that the MGA components are relatively isolated form the rapid changes in the database technology.

2. *Common Model Interface Layer (CMI):* Services of the Object Database Management Layer do not define the model semantics, although it needs to be "known" for the GMB and the Model Interpreters. The CMI is the representation of the model semantics for inter-tool communication. The "physical" manifestation of the CMI are the C++ header files generated by the ODL translator of the OODB [14].
3. *Distributed Object Communication Layer (CORBA):* The MGA allows concurrent access to the Model Database by the GMB, and by various systems engineering analysis tools (and program synthesis tools). This is a must in large-scale engineering applications where several engineering groups work concurrently on various aspects of the same system. From the operational point of view, the architecture is designed as a distributed object system, where the communicating "macro objects" are: GMB, OODB, and the Model Interpreters. For intertool communication, we selected the emerging standard of the Object Management Group (OMG), the Common Object Request Broker Architecture (CORBA) [14]. CORBA supports the management of interaction between client and server objects.

4.3 Meta-Level

Customization of the tools on the MIPS level is accomplished using methods and tools on the meta-level. The first step in building a domain specific MIPS environment is the specification of the modeling paradigm. The modeling paradigm defines concepts, relationships, model composition principles and model integrity constraints specific to the domain, therefore the following information must be expressed:

1. Concepts and relationships captured in the models.
2. Model composition principles and integrity constraints that define model organization and capture the semantics of models.
3. Graphical formalism used for model representation.

The first two components define the semantic content of models unique to a given modeling paradigm. The third component defines the representation formalism, which may exist in different alternatives. Currently, the paradigm specification is supported by the EDL and ODL declarative languages and the related translators that are part of the XVPE and the OODB.

A critical aspect of managing the complexity of large-scale systems is the flexibility of the meta-level toward defining complex model composition principles. In our experience, each engineering domain tends to develop a unique combination of composition principles which are ultimately used to conceptualize systems. Domain engineers require direct support of these "customary" composition principles in their modeling environment, and consider other principles idiosyncratic. Currently, the MGA meta-level allows the combination of the following composition principles in a domain:

Aspects: A frequently used engineering technique is to focus on selected features of complex systems. MGA tools allow the definition of elaborate sets of aspects capturing particular kind of system characteristics. Since the modeling aspects describe the same underlying system and/or include relationships that are not independent from each other, the resulting models may be subject to complex integrity constraints and/or dependency relationships that must be satisfied or explicitly represented. In MGA, modeling aspects are not predefined and forced onto different application domains. Aspects can be selected and modified according to the nature of the domain. The following two aspect categories are supported:

Independent aspects: They do not have generic integrity constraints, therefore models can be built independently from each other. However, the model components in independent aspects may include associations/references to other aspects expressing a particular relationship.

Dependent aspects: Dependent aspects have strong logical or conceptual relationship. The current version of MGA modeling tools support a specific, frequently used dependency, called *structural dependency*. In structurally dependent multiple aspect modeling the basic model structure is defined in a *dominant aspect* (e.g. physical structure or functional structure). Having the model components defined, each component (e.g. a functionality or a physical component) can be modelled from a set of different, usually complementary views (e.g. dynamics, fault characteristics, etc.). In larger systems the dependent

aspects can be arranged into independent sets (e.g. the set of behaviors (dynamics, fault, etc), the set of interaction types (electrical, mechanical, etc.). If these sets are *orthogonal*, i.e. they describe mutually independent characteristics, the Cartesian product of the dependent aspect sets may be meaningful (e.g. dynamics of the electrical interactions of a functionality, etc.).

Hierarchical composition: Models can be built hierarchically, i.e. models can contain sub-models of the same and/or different types. The depth of the model hierarchies is not limited. Hierarchical composition can be used for many different purposes in modeling paradigms, such as expressing part-whole relationships or different levels abstractions.

Module interconnections: A common model composition concept in engineering is module interconnection. Model objects with interconnectivity have well-defined interfaces, which controls the visibility across the boundaries. The module interfaces are specified in terms of input/output objects of a given type.

Associations: Model objects may have associations in the context of other model objects. The associations can be defined through creating "references" to model objects from the context of other model objects and connecting the references to the associated model object. For example, a signal can be associated with a physical variable through creating a physical variable reference in the signal-flow model, and connecting the reference to a particular signal. Associations are used to express conceptual relationships among model components.

Object Conditionalization: Certain objects and connections can either be "present" or "absent" in a model under different conditions. For example, a fault propagation link can be absent when the system is in start-up phase, but it can be present in full operation. Object conditionalization means that objects can be conditionalized by some other objects.

Model types: A model can be declared to be a type. The instances of a model types can be components of more than one model. If a model type is changed, the change propagates to all of its intstances.

Using the appropriate combination of these composition principles and defining the specific concepts and relationships of the domain, a wide variety of domain specific modeling environments can be created.

The second step in building domain specific MIPS environments is to define and implement model interpreters. Currently, writing model interpreters is relatively complicated and requires in-depth knowledge of the internal interfaces of MGA. One of our ongoing research effort in MGA is the development of a declarative representation for the interpretation process, and the development of tools that use these representations to instantiate predefined interpreter templates.

5. Applications of MGA

MGA first evolved in instrumentation systems which was followed by several successful applications in the aerospace and chemical manufacturing industries. A summary of selected MGA applications developed recently are shown in Table 1.

MGA-DTOOL/MGA-RDS: A Model-Based Engineering Environment for FDIR in Aerospace

MGA is the software framework of a model-based robust diagnostic system (MGA-RDS) and diagnosability/testability analysis tool (MGA-DTOOL). It has been developed in cooperation with the Boeing Company, and is used in the *International Space Station Alpha (ISSA) Program*. In this application, the multiple-view modeling environment supports the *functional, physical, and behavioral* modeling of ISSA system components [5]. A unique feature of this application is the complexity of the modeling paradigm and the large size of the models. Tools of the modeling environment allow graphical model building, and support extensive model consistency checking. The system has several model interpreters. The model interpreter for the Diagnosability/Testability Analysis Tool (DTOOL) [5] extracts relevant information from the multiple-view models and synthesizes data structures required by DTOOL. DTOOL evaluates detectability, distinguishability, and predictability of faults given on-line sensor allocation and built-in-test (BIT) coverage, generates optimum test sequences, and provides advice for additional sensors/BIT coverage to meet defined criteria. A different family of model interpreters automatically generates executable code for the real-time diagnostic system from the same integrated model-set allowing significant savings in system/software engineering time.

Problem Solving Environment for Chemical Plants

The Intelligent Process Control System (IPCS) is an on-line problem solving environment and decision support tool for process and production management. The IPCS application is described in a companion paper in this proceedings [7].

CADDMAS: Computer Assisted Dynamic Data Monitoring and Analysis System

MGA is the underlying software technology for the Computer Aided Dynamic Data Monitoring System

System	Domain	Function	Platform	Difficulty
MGA-DTOOL	aerospace systems	modeling and diagnosability analysis	SUN and HP700 workstations	Complexity of modeling paradigm, size of models
MGA-RDS	aerospace systems	robust, real-time diagnostics	SUN and HP700 workstations	Complexity of modeling paradigm and model interpreters
IPCS	chemical plants	problem solving environment for process management	HP700 workstations	Complexity of modeling paradigms, heterogeneity of the synthesized applications
CADDMAS	dynamic data monitoring	real-time vibration analysis	heterogeneous network of PC-s and over 100 TI-TMSC40 and C31 signal processors	Complexity of the synthesized parallel applications, complexity of the computing platform
DATVAL	turbine engine testing	model-based sensor data validation	SGI workstations	Relationship to legacy systems

Table 1: Selected MGA applications

(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 [6]. Different versions of the CADDMAS are now being applied as primary on-line test systems in the turbine engine testing facilities of AEDC. A derivative of the CADDMAS system has been requested by NASA and will be installed in the Structures and Dynamics Laboratory of *NASA-MSFC for Space Shuttle Main Engine testing*.

In the CADDMAS application, the MGA modeling environment supports the hierarchical modeling of signal flow graphs, hardware resources, and resource limitations. The model interpreters synthesize the complex executable program and configure the parallel computing platform. The main difficulty of the CADDMAS application is the complexity of the executable system synthesized from the models. The model-based programming environment allows AEDC and NASA personnel to reconfigure the complex parallel CADDMAS software according to the needs of a new test. The MGA-based parallel program synthesis system has been recently extended to image processing [9] and parallel turbine engine simulation applications.

DATVAL: Data Validation System for Turbine Engine Testing

A primary concern of testing systems is of the quality of data produced. In cooperation with USAF-AEDC, a model-based, real-time data validation system is under development using MGA. DATVAL detects anomalies in sensor data and isolates the most plausible source of faults. The data validation system utilizes massive amount of information about the test article and the testing facility. The modeling paradigm of the full system will include multiple view models of the facility, the test article, and will configure the real-time validation program on a distributed workstation network. The primary technical challenge of the DATVAL application is the elaboration of methods to integrate legacy systems (large-scale engine simulator code written in FORTRAN) with MIPS environments.

6. Summary

According to our experience, the model-integrated approach in program synthesis is a powerful method. It is particularly useful in constructing and maintaining large-scale embedded applications operating in dynamically changing, complex environments. It has been our experience that the tremendous productivity advantage of domain specific MIPS environments and related systems

engineering tools can be exploited only if the following criteria are met:

- The modeling environments are closely tailored to the domain.
- The modeling environments are able to evolve with the increased understanding of the domain problems and solution strategies.
- The modeling environment supports interoperability with engineering databases and systems engineering tools.

Some of the important conclusions of the practical experiences with the MGA are the following:

- *Scalability of models*. In several MGA applications both the complexity of the modeling paradigm and the size of the models have become a major issue. Our experience is that the introduction of domain specific model composition principles has been the key element of managing this complexity. This relieved the domain engineers from the responsibility of finding an efficient implementation for the domain specific modeling paradigm in terms of the object model of the underlying OODB.
- *Scalability of executable programs.* The IPCS and CADDMAS applications synthesize highly complex executable programs, whose structure and operation strongly depends on their "environment", i.e. on the plant, the system to be tested, the available computing platform, etc. In these applications, model-integrated program synthesis has dramatically decreased the cost of application development, and virtually eliminated the need for costly system integration.
- *Different application domains.* We have built customized MIPS environments for a wide range of applications, including instrumentation, signal processing, process management systems for chemical plants, fault diagnostics, simulation and others. Besides the generic tools of the MGA, modeling paradigm definitions, model interpreters, run-time libraries, even models have proved to be highly reusable in different domains.

Our current research focuses on the development of meta-level tools which will significantly decrease the cost of creating customized MIPS environments even for complex, critical applications.

References

[1] Rozenblit, J., Buchenrieder, K.: *Codesign: Computer-Aided Software/Hardware Engineering*, IEEE Press, 1995.

[2] Brooks, F.P.,Jr.: "No Silver Bullet: Essence and Accidents of Software Engineering", *Computer*, Vol. 20., No. 4., pp. 10-19, Apr. 1987.

[3] Harel, D.: Biting the Silver Bullet", *Computer*, Vol. 25., No. 1., pp.8-19, January, 1992.

[4] White, S.: "Tracing Product and Process Information when Developing Complex Systems", *Proc. of the 1994 CSESAW Workshop*, pp. 45-50, July 1994.

[5] 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, 1992 WA.

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

[7] Karsai, G., Sztipanovits, J., Franke, H., Padalkar, S., DeCaria, F.: "Model-Embedded On-Line Problem Solving Environment for Chemical Engineering", in this proceedings.

[8] Moore, M., Nichols, J.: "Model-Based Synthesis of Real-Time Image Processing Systems", in this proceedings.

[9] Booch, G.: *Object-Oriented Analysis and Design with Applications*, Prentice Hall, 1991.

[10] D'Ippolito, R., Lee, K.: "Modeling Software Systems by Domain," *AAAI-92 Workshop on Automating Software Design*, 1992.

[11] G.Abowd, R.Allen, D.Garlan, "Using Style to Give Meaning to Software Architecture", in *Proc. SIGSOFT'93: Foundations of Software Eng.*, 12/93.

[12] 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.

[13] Karsai, G.: A Visual Programming Environment for Domain Specific Model-Based Programming," *IEEE Computer*, pp. 36-44 March 1995.

[14] Cattell R.G.G. (Ed.): *Object Database Standard: ODMG-93*, Morgan Kaufmann Publishers, 1994.