

Model-Integrated Computing

Janos Sztipanovits and Gabor Karsai, Vanderbilt University

Computers now control many critical systems in our lives, from the brakes on our cars to the avionics control systems on planes. Such computers wed physical systems to software, tightly integrating the two and generating complex component interactions unknown in earlier systems. Thus, it is imperative that we construct software and its associated physical system so they can evolve together.

One approach that accomplishes this, *model-integrated computing*, works by extending the scope and use of models. It starts by defining the computational processes that a system must perform and develops models that become the backbone for the development of computer-based systems. In this approach, integrated, multiple-view models capture information relevant to the system under design. Instead of developing a specialized application or system, we want to provide solutions for an entire class of problems. Models can represent the designer's understanding of an entire computer-based system, including the information-processing architecture, physical architecture, and operating environment. Integrated modeling explicitly represents dependencies and constraints among various modeling views.

In model-integrated computing, tool-specific model interpreters translate model information into languages used by the tools. These tools analyze the system's interdependent characteristics (such as performance, safety, and reliability).

Other interpreters translate models into executable specifications used to automatically synthesize software.

Model-integrated computing is similar to the domain-specific software architecture approach, yet its models capture not only the software's architecture, but also the environment's. And it uses models in a more general sense than model-based software engineering: Models capture the information that defines the application's changing part, information that is frequently more comprehensive than models of the software itself.

The Multigraph Architecture is the framework for model-integrated computing developed at Vanderbilt's Measurement and Computing Systems Laboratory. Major applications include

- Boeing's modeling and analysis environment for fault detection, isolation, and recovery (used on the International Space Station),
- DuPont's online problem-solving environment for chemical manufacturing,
- Saturn Corporation's manufacturing-execution system, and
- high-performance, real-time instrumentation systems installed at the Air Force Arnold Engineering and Development Center and NASA-MSFC.

Our Web site, <http://mcs.l.vuse.vanderbilt.edu>, gives detailed information on these projects.

MULTIGRAPH ARCHITECTURE

Using model-integrated methods poses several challenges. First, it's difficult to define models of inherently heterogeneous systems when their development cuts through several disciplines that neither share terminology nor even think about problems in the same way. Thus, modeling tools must support domain-specific modeling paradigms. Analysis methods and tools frequently preserve their *discipline-specific* modeling perspectives and techniques. This creates the need for a translation mechanism to bridge domain-specific models and the formalisms required by analysis tools. So the primary challenge in building a tool infrastructure is to find an architecture that separates generic and domain-specific components. It must also facilitate the economical use of model-integrated engineering in widely varying domains.

MGA evolved during the last decade into a system-modeling framework, tool integration architecture, and infrastructure for model-integrated computing.

As Figure 1 shows, MGA employs a two-level development process. Software or system engineers follow a metalevel process to specify and configure domain-specific environments. Domain engineers then use the environment to build and analyze domain models and generate applications. An expandable tool set supports both processes.

Metalevel process

The metalevel is a domain-independent abstraction layer in which domain-specific environments are formally specified, validated, and synthesized. Metamodels are abstract specifications that provide formal semantics for domain-specific modeling languages. They define the properties of domain models in terms of concepts, relations, model-composition principles, and integrity constraints. Model interpreters (program generators) capture the mapping between domain models and applications, determining the operational semantics. In this context, applications are executable instances of domain models, and domain models are instances of metamodels.

In a project, supported by DARPA's

Evolutionary Design of Complex Software program, we implement the meta-level as a metaprogramming interface to the tools. In addition to supporting formal specification, it

- generates configuration files for tools,
- generates model interpreters, and
- validates the specification of modeling paradigms and model interpreters.

The metalevel process is evolutionary: Domain modeling typically results in a better understanding of the problem, which in turn leads to upgrading of the modeling paradigm and resynthesis of the environment using metalevel tools.

System development process

A typical environment used for model-integrated systems development has four components. A *graphical model builder* constructs domain models, providing a customizable, graphical, model-building environment. The interface between this level and the metalevel enforces domain-specific constraints during model building. By explicitly representing constraints among modeling views, models can be tested using systemwide consistency and completeness criteria (see "A Configurable Visual Programming Environment: A Tool for Domain-Specific Programming," *Computer*, Mar. 1995, p. 36).

A *model database* stores the complex, multiple-view domain models. The latest MGA implementations include object-oriented databases.

Model interpreters synthesize executable programs from domain models and generate data structures for tools. Since model interpreters capture the relationship between the problem and solution, they are specific to the domain modeling paradigm and to the application type.

Typically, we specify executable programs in terms of the *Multigraph computational model*. A macrodataflow model, it represents the synthesized programs as an attributed, directed, bipartite graph that uses buffers or computational nodes. Elementary computations, which the Multigraph kernel

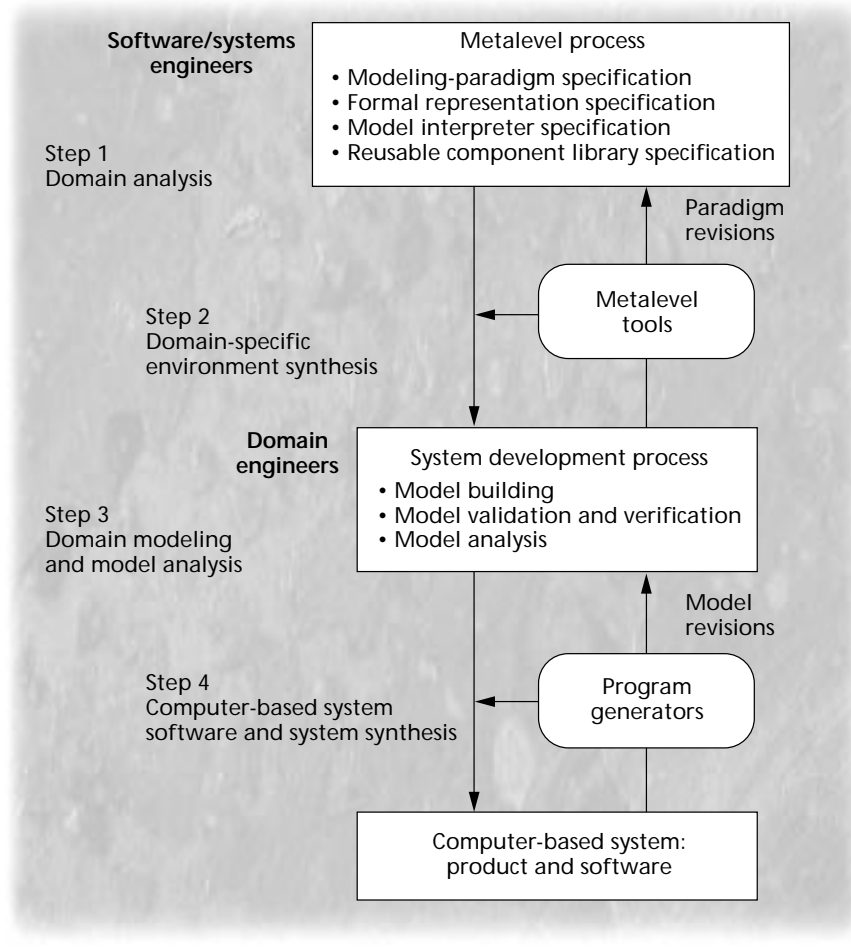


Figure 1. System development using the Multigraph architecture.

schedules, are carefully defined reusable code components in application-specific runtime libraries. We implemented the Multigraph kernel as an overlay above the operating and communication systems. The kernel also supports dynamic reconfiguration of the executing system, a unique capability.

Using these components, domain engineers build and analyze domain models and synthesize executable applications. As requirements enforced by the external environment change, domain engineers can modify models and resynthesize the software. This capability is extremely important in applications in which the physical environment changes continuously.

A characteristic that distinguishes model-integrated computing is that we match modeling paradigms to the needs of the domain engineers rather than those of the software architects (except when the end users are software architects). This approach takes us closer to the technology of end-user program-

mable complex applications. Our approach also differs from object-oriented analysis and design: Models may have multiple (even changing) execution semantics as defined by the model interpreters.

Our current research focuses on introducing formal techniques to the architecture's metalevel components. Another ongoing research project conducted in cooperation with Sandia National Laboratories examines model-integrated engineering of high-consequence systems. ♦

Janos Sztipanovits is a professor of electrical and computer engineering at Vanderbilt University and the founder and director of its Measurement and Computing Systems Laboratory. Contact him at sztipaj@vuse.vanderbilt.edu.

Gabor Karsai is an assistant professor of electrical and computer engineering at Vanderbilt and associate director of MCSL. Contact him at gabor@vuse.vanderbilt.edu.