GME: The Generic Modeling Environment

James Davis
Institute for Software Integrated Systems,
Vanderbilt University
Box 1829, Station B, Nashville, TN 37235
(615) 343-7530
james.davis@vanderbilt.edu

Categories and Subject Descriptors
D.2.6 [**Programming Environments**]: Graphical Environments, Integrated Environments

General Terms
Design, Languages.

Keywords
Model Integrated Computing, domain specific visual languages, system synthesis, abstract system models.

1. INTRODUCTION
The Generic Modeling Environment (GME) is an architecture developed for producing domain-specific design environments. These domain specific environments are used to capture specifications in a natural language for the end user and to automatically generate or configure target applications in a given domain (i.e. in a particular engineering field). Well known examples include Matlab/Simulink for signal processing and LabView for instrumentation. One of the common characteristics of many domain specific tools include a visual specification interface. The advantages of visual environments have been demonstrated in many different domains. However, the high cost of development restricts their use. Fields with small markets do not typically justify the high cost of developing a customized visual interface. GME is presented as a solution to this problem, as it is a configurable, graphical modeling environment. GME supports a variety of general modeling principles in the generic tool. These general modeling principles are then utilized in creating the domain specific language. By making GME easily configurable for a wide variety of domains, cost issues are primarily related to the development of the generic environment.

GME is based on over fifteen years of research in Model Integrated Computing (MIC) at the Institute for Software Integrated Systems at Vanderbilt University. MIC focuses on the development of domain specific languages and domain specific environments. GME is the architecture used to realize domain specific languages and domain specific modeling environments. GME is configured using metamodels to specify the modeling language of the target domain. Metamodels capture the syntax, semantics, and presentation of the domain specific language. What objects exist in the language, how they may be related, and how they are presented are all captured in the metamodels. Visualization of the different objects and relationships in the language are limited to the set of presentation idioms that GME supports. However, many of these limitations are of the visual editor – the core component of GME could handle other presentation methods. In practice, the set of visual idioms GME supports are sufficient for realizing a large set of domain specific languages.

GME metamodels are based on the Unified Modeling Language (UML). UML class diagrams are used to capture the syntax of the domain specific language. Presentation/visualization information is captured using a combination of stereotypes and predefined attributes of the UML classes and associations. The Object Constraint Language (OCL) is used to compose constraints to impose the static semantics of the target language. The metamodels are used to automatically configure GME for the specified target domain. It is interesting to note that GME metamodels are constructed using GME. In effect, metamodels define the set of all possible models that can be constructed in the target modeling paradigm or language.

Once models are constructed in GME, a process known as *model interpretation* is used to process the models and to generate applications, data for COTS tools, or configuration for third party tools. The model interpreter is a small application component that is written to work on the domain models. It must be generic in nature to ensure that all models that are legal in the domain specific language can be handled. While developing a model interpreter may be time consuming, it is developed once, usually by an engineer versed in MIC, and is then employed many times by domain experts using the domain specific environment.

GME utilizes many object oriented features. In addition to the heavy reliance on UML in the metamodeling framework, features such as inheritance are available in GME as a domain modeling tool. Once an object is created in GME, it effectively becomes a type. It can be subtyped and instantiated at will. Whenever modifications occur to the base type, the modifications are automatically enforced on any subtypes or instances.

2. GME ARCHITECTURE
GME has a modular, component-based architecture depicted in Figure 1. Currently SQL, XML and a fast, proprietary binary file format are supported as a thin storage layer for model persistence. The Core component implements the two fundamental building blocks of a modeling environment: objects and relations. Among its services are distributed, multi-user access (i.e. locking) and undo/redo.

OOPSLA'03, October 26–30, 2003, Anaheim, California, USA.
ACM 1-58113-751-6/03/0010.

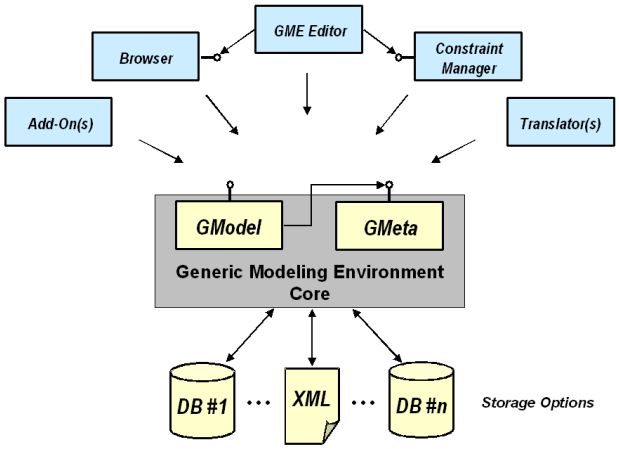

Figure 1: GME architecture

Two components use the services of the Core: the GMeta and the GModel. The GMeta exposes the modeling paradigm, while the GModel implements the GME modeling concepts for the given paradigm. The architecture is reflective: the GModel uses the GMeta component extensively through its public COM interfaces. The GModel component publishes its services through a set of COM interfaces as well.

The user interacts with the components at the top of the architecture: the GME Editor, the Model Browser, the Constraint Manager, Interpreters and Add-ons. Add-ons are event-driven model interpreters. The GModel component exposes a set of events, such as "Object Deleted," "Connection Created," "Attribute Changed," etc. External components can register to receive some or all of these events. They are automatically invoked by the GModel when the events occur. Add-ons are extremely useful for extending the capabilities of the GME Editor. When a particular domain calls for some special operations, these can be supported without modifying the GME itself. A good example for an add-on is the OCL syntax checker integrated into the metamodeling environment.

The Constraint Manager can be considered as an interpreter and an add-on at the same time. The user can start it explicitly and it is also invoked automatically when event-driven constraints are present in the given paradigm. Depending on the priority of a constraint, the operation that caused a constraint violation will be aborted. For less serious violations, the Constraint Manager only issues a warning.

The GME Editor component has no special privileges in this architecture. Any other component (translator, add-on) has the same access rights and uses the same set of COM interfaces to the GME Core. Any operation that can be accomplished through the Editor, can also be done programmatically through the interfaces. This architecture is very flexible and makes the whole environment easily extensible and customizable. A GME Editor component is being developed in Eclipse. One advantage of the GME architecture is the ability to extend the architecture with different components to enhance the versatility and usability of GME.

Model Interpreters can be developed using the GME COM interfaces or using a high level C++ interface. The high level interpreter interface is referred to as the Builder Object Network (BON). The BON is a set of C++ classes that hide the complexity of the COM interface from the user. Additionally, the BON can be extended to domain specific classes for use in developing the modeling interpreter. The metamodels can be used to automatically generate the extension to the BON. This not only enables the model interpreter developer to utilize the same classes as specified in the metamodel, but also reduces the time and effort necessary to produce an interpreter. Many of the generated BON extensions were required to be manually created in the past.

3. Example DSDE

Figure 2 shows an example GME domain: a system on a chip/platform design environment. This domain is an integrated, extensible, simulation environment. Data flow models, such as those shown, are used to specify the processing of the application. Other models represent the hardware available. Through the model interpreters, simulations can be generated to evaluate possible system configurations for performance and power characteristics. The same set of models are used to configure many different types of simulators, thus reducing the designer's effort of producing the simulations.

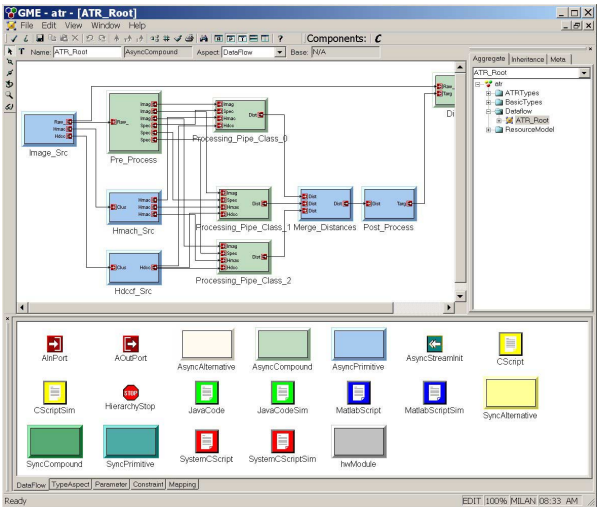

Figure 2: Example modeling environment

4. REFERENCES

[1] Sztipanovits J., Karsai G.: "Model-Integrated Computing", Computer, pp. 110-112, April, 1997.

[2] Ledeczi A., et.al.: "Composing Domain-Specific Design Environments", Computer, pp. 44-51, November, 2001.

[3] The GME User's Manual, available from http://www.isis.vanderbilt.edu/projects/gme.

[4] Ledeczi A., Davis J., Neema S., Agrawal A.: "Modeling Methodology for Integrated Simulation of Embedded Systems", ACM Transactions on Modeling and Computer Simulation, January 2003