Model Integrated Computing in the Large

Akos Ledeczi, Gyorgy Balogh, Zoltan Molnar, Peter Volgyesi and Miklos Maroti
Institute for Software Integrated Systems
Vanderbilt University, Nashville, TN 37235
+1 615 343 8307
{akos, bogyom, zolmol, volgy, mmaroti}@isis.vanderbilt.edu

Abstract[1],[2]—As model integrated computing gains wider and wider acceptance, the scalability of the supporting tools becomes a significant issue. Large, complex projects involve many developers who create large and complex models. Supporting large-scale model integrated computing requires two key features currently lacking in available tools: (1) distributed, simultaneous multi-user access to the models and (2) model versioning. Our proposed solution is based on storing the models at a relatively fine granularity in xml files on a server under the control of a traditional source code control system. The model builder tool then only needs to implement the client functionality of the version control tool. However, there is one significant technical challenge that needs to be solved. Models are captured as complex interdependent data structures in a model editor. Containment, binary and n-ary relations are kept as a consistent set of data by the model builder. The model builder tool needs to keep track of these interdependencies and ensure consistency at all times even when multiple users edit the models.

TABLE OF CONTENTS

1. INTRODUCTION

As Model Integrated Computing (MIC) [1] gains wider and wider acceptance the scalability of the supporting tools becomes a significant issue. Large, complex projects involve many developers who create large and complex models. Supporting large-scale model integrated computing requires two key features currently lacking in available tools: (1) distributed, simultaneous multi-user access to the models and (2) model versioning. Traditional, code-driven development environments have had these features available through software version control tools such as CVS or Visual Sourcesafe for a long time now. They are typically based on a client-server model where the server stores the source files that comprise the project and provide check-in/check-out support to clients. Other features include versioning and merging capabilities. The question is how can be similar support provided to model integrated development?

As traditional software version control tools are mature products and most developers already use them, the natural answer is to utilize these tools in the model integrated computing arena as well. Our proposed solution is based on storing the models at a relatively fine granularity in xml files on a server under the control of a versioning system. The model builder tool then only needs to implement the client functionality of the version control tool. This approach has several significant advantages:

- Multiple users can access the same project simultaneously. The server provides locking through the check-in/check-out facilities of the version control software. We mandate strict access control implemented by the model builder client to ensure that the server stores a consistent set of models all the time.
- Having consistent models all the time means that any past version of the models is available from the server.
- Distributed multi-user access and model version control can be achieved "almost free," that is, only relatively minor modifications to the model builder tool are required to gain these two significant features.

However, there is one significant technical challenge that needs to be solved. Models are captured as complex interdependent data structures in a model editor. Containment, binary and n-ary relations, attributes and preferences are kept as a consistent set of data by the model builder. In traditional development, text entered by the user describes interdependencies between code components and a compiler is tasked to figure out whether the code is consistent or not (e.g. does a particular function call refer to an existing function or not) and all errors are corrected manually by the user. Model integrated computing tools keep the models consistent and compliant with the given domain-specific language at all times. Consequently, editing a certain model can impact several other models. The model builder tool needs to keep track of these interdependencies and lock all models that can be affected by editing the current model.

[1] 0-7803-8870-4/05/$20.00© 2005 IEEE
[2] IEEEAC paper #1120, Version 3, Updated November 28, 2004

We have implemented these features in the GME [2], the Generic Modeling Environment (http://www.isis.vanderbilt.edu/Projects/gme/) and integrated Rational Clearcase [7] and Microsoft Visual Sourcesafe [8] support. In the rest of the paper we describe Model Integrated Computing and the Generic Modeling Environment in more detail. Then we detail the technical approach and challenges to providing distributed simultaneous access to models and version control. Then an innovative technique to model migration supporting scalability is presented. Finally, we present our conclusions.

2. MODEL-INTEGRATED COMPUTING

Model-Integrated Computing (MIC) employs domain-specific models to represent the system, its environment, and their relationship. These models are then used to automatically synthesize the embedded applications and generate inputs to COTS analysis tools such as model checkers or simulators. This approach speeds up the design cycle, facilitates the evolution of the application and helps system maintenance, dramatically reducing costs during the entire lifecycle of the system.

Creating domain-specific visual model building, constraint management, and automatic program synthesis components for a MIC environment for each new domain would be cost-prohibitive for most domains. Applying a generic environment with generic modeling concepts and components would eliminate one of the biggest advantages of MIC — the dedicated support for widely different application domains. An alternative solution is to use a configurable environment that makes it possible to customize the MIPS components for a given domain.

The configuration can be done through metamodels specifying the modeling paradigm of the application domain. The modeling paradigm is the modeling language of the domain specifying the objects and their relationships. In addition to syntactic rules, semantic information can also be described as a set of constraints. The Unified Modeling Language (UML) and the Object Constraint Language (OCL), respectively, are used for these purposes in MIC. The metamodels, are used to automatically generate the MIC environment for the domain. An interesting aspect of this approach is that the same environment is used to build the metamodels [2].

The generated domain-specific MIC environment is used to build domain models that are stored in a model database. These models are used to automatically generate the applications or to synthesize input to different analysis tools. This process is called model interpretation or translation.

This approach and the same software toolset have been used to create and deploy large-scale systems that are in everyday use in widely different engineering domains. The Saturn Site Production System (SSPF) is a large distributed production monitoring system used by the Saturn Corporation in car manufacturing [6]. Other systems include a fault detection isolation and recovery system used by Boeing and NASA on the International Space Station [4], an integrated simulation environment for embedded system development [3], and a safety and reliability analysis tool used by Sandia National Labs [5].

The Generic Modeling Environment

The Generic Modeling Environment (GME) is a metaprogrammable graphical editor supporting MIC. The component based architecture of GME is shown below.

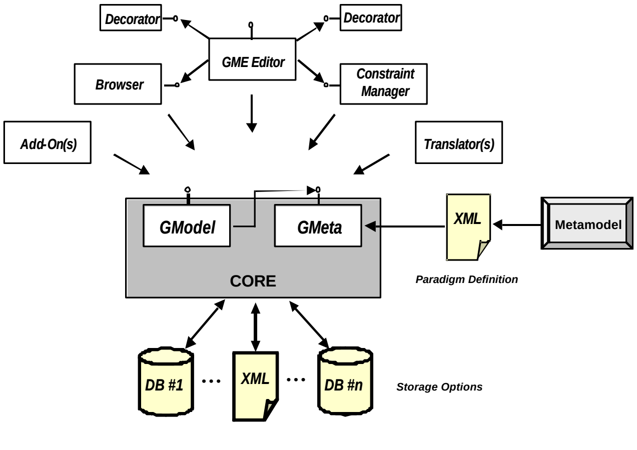


Figure 1 GME Architecture

The thin storage layer includes components for the different storage formats. Currently, an ODBC interface to different relational databases, an XML format and a fast proprietary binary file format are sup-ported. Supporting an additional format requires the implementation of a single, well-defined, small interface component.

The Core component implements the two fundamental building blocks of a modeling environment: objects and relations. Among its services are distributed access (i.e. locking) and undo/redo.

Two components use the services of the Core: the GMeta and the GModel. The GMeta defines the modeling paradigm, while the GModel implements the GME modeling concepts for the given paradigm. The GModel uses the GMeta component extensively through its public COM interfaces. The GModel component exposes its services through a set of COM interfaces as well. The user interacts with the components at the top of the architecture: the GME User Interface, the Model Browser, the Constraint Manager, Translators and Add-ons.

Add-ons are event-driven model interpreters. The GModel component exposes a set of events, such as “Object Deleted,” “Set Member Added,” “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 User Interface. When a particular domain calls for some special operations, these can be supported without modifying the GME itself.

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

The GME User Interface 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. Any operation that can be accomplished through the GUI, can also be done programmatically through the interfaces. This architecture is very flexible and supports extensibility of the whole environment.

Generic Modeling Concepts

The vocabulary of the domain-specific languages implemented by different GME configurations is based on a set of generic concepts built into GME itself. The choice of these generic concepts is the most critical design decision. GME supports various concepts for building large-scale, complex models. These include: hierarchy, multiple aspects, sets, references, and explicit constraints. The UML class diagram in Figure 2 depicts the complex relationships among these and other important concepts.

A Project contains a set of Folders. Folders are containers that help organize models, just like folders on a disk help organize files. Folders contain Models. Models, Atoms, References, Connections and Sets are all first class objects, or FCO-s for short.

Atoms are the elementary objects – they cannot contain parts. Each kind of Atom is associated with an icon and can have a predefined set of attributes. The attribute values are user changeable. A good example for an Atom is an AND or XOR gate in a gate level digital circuit model.

Models are the compound objects in our framework. They can have parts and inner structure. A part in a container Model always has a Role. The modeling paradigm determines what kind of parts are allowed in Models acting in which Roles, but the modeler determines the specific instances and number of parts a given model contains (of course, explicit constraints can always restrict the design space). For example, if we want to model digital circuits below the gate level, then we would have to use Models for gates (instead of Atoms) that would contain, for example, transistor Atoms.

This containment relationship creates the hierarchical decomposition of Models. If a Model can have the same kind of Model as a contained part, then the depth of the hierarchy can be (theoretically) unlimited. Any object must have at most one parent, and that parent must be a Model. At least one Model does not have a parent; it is called a root Model.

Aspects provide primarily visibility control. Every Model has a predefined set of Aspects. Each part can be visible or hidden in an Aspect. Every part has a set of primary aspects where it can be created or deleted. There are no restrictions on the set of Aspects a Model and its parts can have; a mapping can be defined to specify what Aspects of a part is shown in what Aspect of the parent Model.

The simplest way to express a relationship between two objects in GME is with a Connection. Connections can be directed or undirected. Connections can have Attributes themselves. In order to make a Connection between two objects they must have the same parent in the containment hierarchy (and they also must be visible in the same Aspect, i.e. one of the primary Aspects of the Connection). The paradigm specifications can define several kinds of Connections. It is also specified what kind of object can participate in a given kind of Connection. Connections can further be restricted by explicit Constraints specifying their multiplicity, for instance. A Connection can only express a relationship between objects contained by the same Model. Note that a Root Model, for example, cannot participate in a Connection at all. In our experience, it is often necessary to associate different kinds of model objects in different parts of the model hierarchy or even in different model hierarchies altogether. References support these kinds of relationships well.

References are similar to pointers in object oriented programming languages. A reference is not a "real" object, it just refers to (points to) one. In GME, a reference must appear as a part in a Model. This establishes a relationship between the Model that contains the reference and the referred-to object. Any FCO, except for a Connection, can be referred to (even references themselves). References can be connected just like regular model objects. A reference always refers to exactly one object, while a single object can be referred to by multiple References. If a Reference refers to nothing, it is called a Null Reference. This can act as a placeholder for future use, for example.

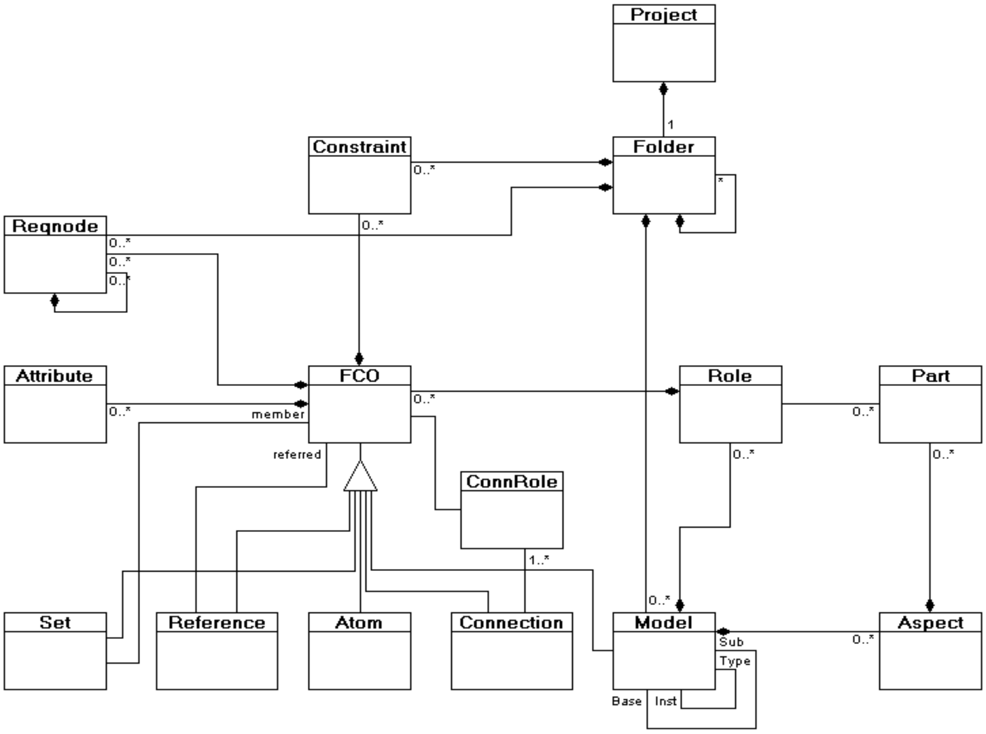


Figure 2 GME modeling concepts

Connections and References are binary relationships. Sets can be used to specify a relationship among a group of objects. The only restriction is that all the members of a Set must have the same container (parent) and be visible in the same Aspect

Some information does not lend itself well to graphical representation. The GME provides the facility to augment the graphical objects with textual attributes. All FCOs can have different sets of Attributes. The kinds of Attributes available are text, integer, double, boolean and enumerated.

Folders, FCOs (Models, Atoms, Sets, References, Connections), Roles, Constraints and Aspects are the main concepts that are used to define a modeling paradigm. In other words, the modeling language is made up of instances of these concepts. In an object-oriented programming language, such as Java, the corresponding concepts are the class, interface, built-in types, etc. Models in GME are similar to classes in Java; they can be instantiated. When a particular model is created in GME, it becomes a type (class). It can be subtyped and instantiated as many times as the user wishes. The general rules that govern the behavior of this inheritance hierarchy are:

- Only attribute values of model instances can be modified. No parts can be added or deleted.
- Parts cannot be deleted but new parts can be added to subtypes.

This concept supports the reuse and maintenance of models because any change in a type automatically propagates down the type hierarchy. Also, this makes it possible to create libraries of type models that can be used in multiple applications in the given domain

3. DISTRIBUTED ACCESS AND VERSION CONTROL

Existing GME backends include a binary file format for fast and easy model access, a rarely used ODBC compliant relational database interface and an XML file format. All these store a project – a set of related models – as a single entity. This does not lend itself to fine grained distributed access by multiple users.

As the structure of a project is inherently hierarchical – it has its own folder structure and models typically contain other models in a recursive manner – storing each folder and model in its own xml file is a natural fit. This fine grained xml backend then can be put under the control of a traditional source code control tool, such as Rational Clearcase or Microsoft Sourcesafe. Locking files for distributed access can then be accomplished utilizing the check-in/check-out facilities of these tools. Furthermore, version control is automatically achieved also.

This client/server architecture is depicted in Figure 3. The xml files are stored on a server under the control of a version control tool. Multiple users on different machines can request access to different parts of the project. If the access is granted they get a local copy of the files and keep a lock on the files until they check in their modifications.

The only significant technical challenge with this approach is how to keep the models consistent as multiple users modify them. The syntax and static semantics of a domain specific language in GME are defined by metamodels in the UML class diagram notation and a set of Object Constraint Language (OCL) constraints. When building models all the rules are enforced by GME at all times. That is, the models are always syntactically correct and consistent. This is in contrast to traditional code-based development methods where the users can modify source files at will and it is the compiler's job to identify inconsistencies and the users' responsibility to fix them.

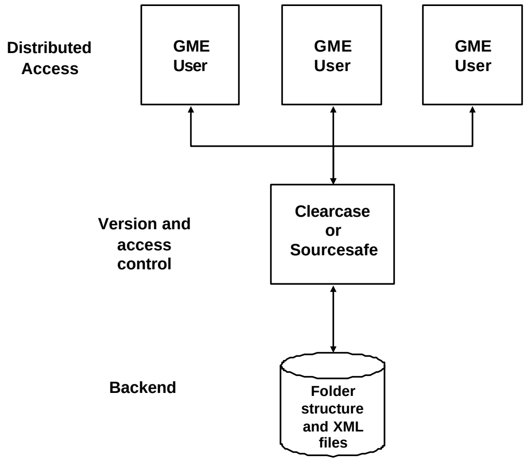


Figure 3 Client server architecture

Having correct and consistent models all the time has been proved to be a valuable feature that needed to be preserved. The consequence of this design decision is that not only do we have to lock a given model when it is edited by a user, but all other models whose concurrent modification might introduce inconsistencies. Another related rule is that no user's work should be wasted, that is, a modification by one user must not invalidate another user's modification (for example, a model must not be deleted when someone is working on one of its descendant in the model hierarchy).

What are the modeling concepts in GME that can affect model consistency in case of concurrent model access? Containment clearly needs to be managed, e.g. when working on a model, none of its descendant should be accessible. Type inheritance forms an orthogonal hierarchy that needs to be guarded, e.g. when a type model is being edited none of its descendants in the inheritance hierarchy can be modified concurrently. References can cut across the containment hierarchy. Furthermore, what model a reference refers to determines what parts it have, specifically what ports it has for making connections to the reference. References can form chains, that is, a reference can refer to a reference that refers to a model etc. Hence, when a model has references pointing to it is being edited, all models along all reference chains need to be locked also. Fortunately, sets and connections always involve objects inside a given model (or their children in case of connections to ports of models). Because of locking due to the containment relation, sets and connections do not necessitate locking additional objects.

Careful design decisions of how the model information is stored in the xml files and the introduction of a model cache kept the number files that need to be locked at a minimum as described below.

Operation

Every project has its own folder containing an .mgx project file, a .bin binary cache file and separate .xml files for each GME model and folder. The project file contains the source control database information; it can be Visual Sourcesafe or Rational Clearcase. The xml files contain all contained objects, relationships, attributes and registry information of the given model. Relationships are stored in one direction only, for example, the basetype of the model is stored, but its subtypes or instances are not. This way the basetype does not need to be locked when one of its subtypes is being edited. The basetype still cannot be edited, as locking all its derived types would fail, but other subtypes or instances of the basetype can be modified. Finally, the cache file contains the structure of the project, i.e. a pointer for each model object in the entire project and all their relationships. This way there is no need to parse a large number of xml files to figure out what to lock when a modification request arrives, that information is available in the cache. The cache is not stored on the server; instead, it is built and maintained by each client.

When a multiuser project is opened, it gets the latest version of xml files then incrementally updates its internal data structure and the cache file. When a modification request occurs GME tries to check out the file being modified and all the dependent files. If any one of the files cannot be checked out, the operation fails and the user gets an error message. Otherwise, all the dependent files are checked out and the modification gets done. Checked out files will be checked in when the project is saved or closed. The user can decide to keep the files checked out after check-in. This way the user can ensure that nobody takes over his part after a save operation. GME keeps in memory only the project structure (i.e. the cache) and saves all other information to disk when it is not necessary. This mechanism makes it possible to work with large projects efficiently.

Example

The Embedded System Modeling Language (ESML) was designed for avionics system development [9]. The Model of Computation ESML targets is based on the Real-Time Event Channel [10] technology defined in the CORBA standard (http://www.omg.org). The result of modeling in ESML is a set of diagrams that visually depict components, interactions, and configurations. Various external tools integrated with the modeling environment perform analyses including end-to-end deadline and rate verification, schedulability checks, event dependency analysis, and others. A translator generates C++ code for system configuration and initialization from the ESML models, and then runs a build process to compile executables.

ESML models for large avionics systems are necessarily large and complex themselves. The first test case of the distributed multi-user access and version control features of GME is ESML. Figures 4 and 5 show different parts of an ESML project opened by two different users at the same time. The models browsers on the right hand side depict (part of) the hierarchical structure of the project. Green checkmarks indicate models currently locked for the current user, while red X marks each model that is checked out by another user.

4. SMART COPY

When working with large projects, the simple copy/paste methods that work well for text become severely inadequate. The issue is again the fact that models capture rich interrelationships between objects. When we select a set of objects and copy them to a different project all relationships that point out of the originally selected set are lost. Furthermore, if we select another set and copy it over, the relationships that existed between objects that were copied first and object that were copied subsequently are not restored either. We have developed two related techniques to address these limitations: a *copy closure* that attacks the first problem and a *smart copy* that solves the second one.

Copy closure

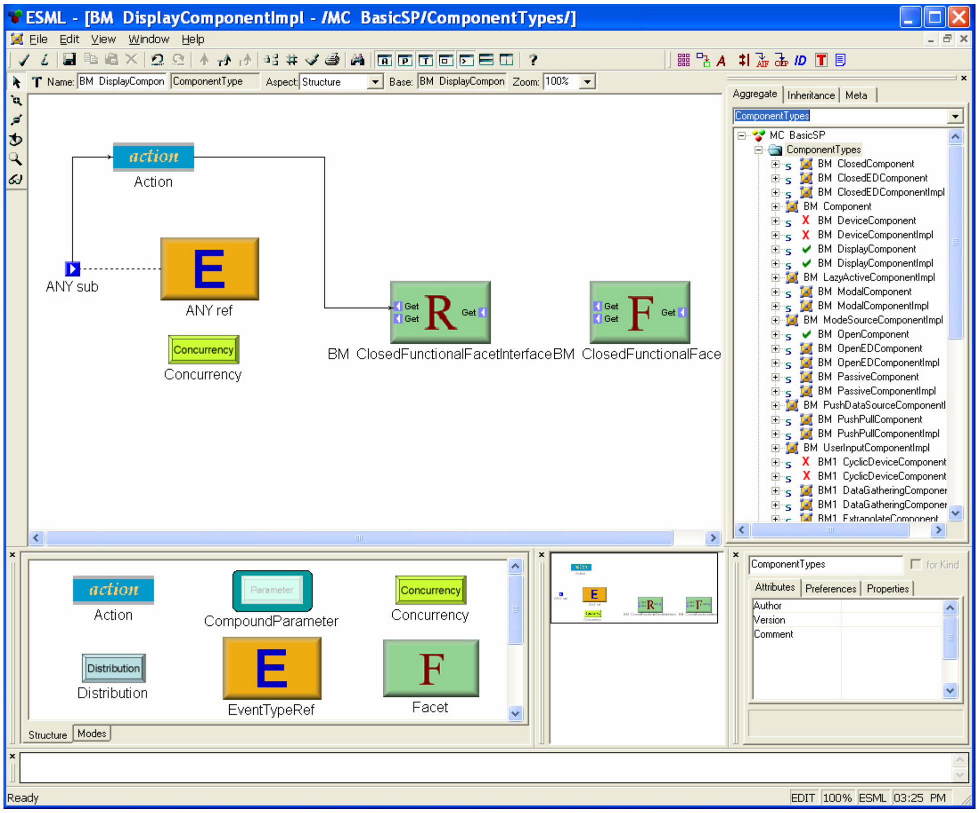
The basic scenario is to allow the user to select a set of objects and create a closure of the set following a user selectable set of relationships, such as containment, inheritance or connection, etc., recursively. This expanded set of object is then copied to the clipboard and can be inserted with the regular Paste command into another project. This way, for example, a model can be selected and a closure can be created automatically following the type inheritance hierarchy upward, that is, all base types of the model will be also copied. The following relationships can be selected:

- Containment: when the algorithm encounters a model (it is part of the originally selected objects or becomes part of the closure), then all of its contained objects become part of the closure. This relationship – just as all others described below – are followed in a recursive manner
- Container: the container model (or folder) becomes part of the closure.
- Refers To: when a reference is encountered the object it refers to is included in the closure.
- Referred By: if an object is part of the closure then all references referring to it will be included also.
- Set Members: when a set is inserted into the closure all of its members are inserted as well.
- Member of Sets: when an object becomes part of the closure, all sets this objects is member of will be inserted.
- Connection: this option has double meaning: (1) if a connection is included then the connection ends will be inserted into the closure and (2) if an object is inserted into the closure and it is connected to an object (it may be itself) then the connection is inserted, and consequently the destination object of the connection will be inserted also.
- Base Type: if a derived object is met, then the basetype of that is included also.
- Derived Types: if a type is met which has one or more derived types and/or instances, then all these models will be included.

Smart copy

The smart copy tries to make a merging copy, that is, it attempts to restore relationships between objects that exist in the source project, but because of multiple partial copy/paste operations would otherwise be lost in the target project. The basic use case is to select and copy a reference and paste it into another project. The feature tries to reestablish the referred to relationship in the target project based on relative name paths. When naming is ambiguous, the parsing logic tries to find the right object based on the original order of creation of the objects with the same name and kind.

References, sets, connections and type inheritance can be restored using the smart copy. Note that references and sets are always inserted regardless whether their referred objects or set members are found or not. A connection is only inserted in case both its source and destination are found.



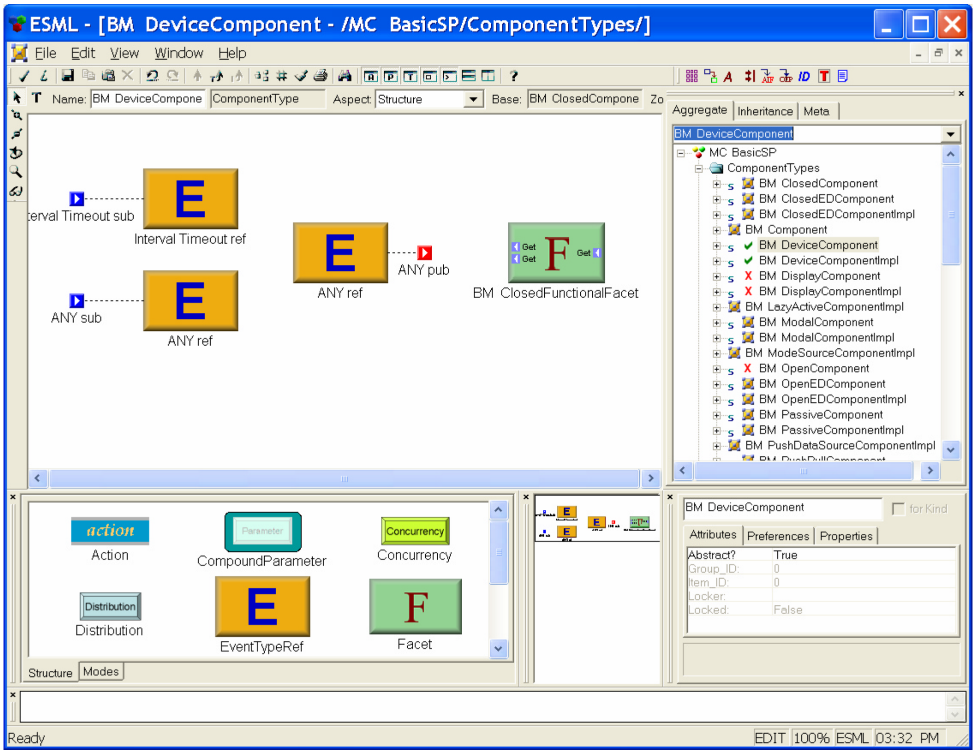
Name: BM DisplayCompon ComponentType Aspect: Structure Base: BM DisplayCompon Zoom: 100%

ComponentTypes

Author	
Version	
Comment	

Ready EDIT 100% ESML 03:25 PM

Figure 4 ESML project edited by a user



Name: BM DeviceCompone ComponentType Aspect: Structure Base: BM ClosedCompone Zo

BM DeviceComponent

Abstract?	True
Group_ID:	0
Item_ID:	0
Locker:	
Locked:	False

Ready EDIT 100% ESML 03:32 PM

Figure 5 Same ESML project edited by another user

5. Conclusion

Models play more and more important roles in software development. Model-driven development methods, such as model integrated computing, need to scale up to being able to support large, complex projects. Distributed, simultaneous access to the models by multiple users and version control are all necessary conditions for this to happen.

The technical approach we developed utilizing xml and traditional source code control tools is a relatively simple, yet powerful technique to provide these capabilities. A significant advantage is that it utilizes existing tools already begin used by developers, hence, the learning curve is relatively flat. We also presented a new powerful copy/paste method that proved to be an invaluable tool in managing complex models. An implementation of these techniques is available in the latest version of the Generic Modeling Environment downloadable at http://www.isis.vanderbilt .edu/Projects/gme/.

Acknowledgements

This work was supported in part by the Escher Research Institute, the Boeing Company and the Defense Advanced Research Projects Agency (DARPA, contract number: F33615-C-00-1633).

References

[1] Karsai G., Sztipanovits J., et al.: “Model-Integrated Development of Embedded Software,” *Proceedings of the IEEE*, Vol. 91, Issue: 1, pp. 145- 164, January, 2003

[2] Ledeczi A., et al.: “Composing Domain-Specific Design Environments,” IEEE *Computer*, pp. 44-51, Nov, 2001

[3] Ledeczi A., Davis J., Neema S., Agrawal A.: “Modeling Methodology for Integrated Simulation of Embedded Systems,” *ACM Transactions on Modeling and Computer Simulation*, Vol. 13, Issue 1, pp. 82-103, Jan 2003

[4] Carnes J. R., Misra A.: "Model-Integrated Toolset for Fault Detection, Isolation and Recovery (FDIR)", *Proceedings of the International Conference on Engineering of Computer Based Systems*, March 11-15, 1996

[5] Davis J. R., Scott J., et al.: "Multi-Domain Surety Modeling and Analysis for High Assurance Systems", *Proceedings of the Engineering of Computer Based Systems Conference*, March, 1999

[6] Long E., Misra A. et al.: "Increasing Productivity at Saturn", *IEEE Computer*, August, 1998

[7] http://www-306.ibm.com/software/awdtools/clearcase/

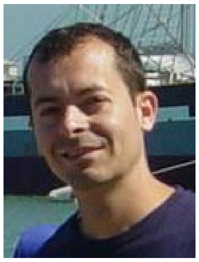
[8] http://msdn.microsoft.com/vstudio/previous/ssafe/

[9] Karsai G, Neema S., Abbott B., Sharp D.: “A Modeling Language and its Supporting Tools for Avionics Systems," *Proceedings of the 21st Digital Avionics Systems Conference*, August 2002

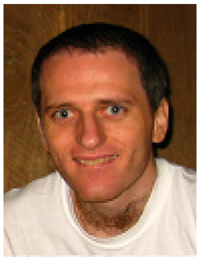
[10] Harrison T., Levine D. and Schmidt D. C.: “The Design and Performance of a Real-time CORBA Event Service,” *Proceedings of OOPSLA*, ACM, Atlanta, GA, Oct, 1997.

Biography

Ákos Lédeczi *(Member, IEEE) received the M.Sc. degree in electrical engineering from the Technical University of Budapest, Budapest, Hungary, in 1989, and the Ph.D. degree in electrical engineering from Vanderbilt University, Nashville, TN, in 1995. He is currently a Senior Research Scientist at the Institute for Software Integrated Systems, Vanderbilt University. His research interests include tools for visual modeling of complex systems, model-based software synthesis, and sensor networks.*



György Balogh *received B.Sc. degree in computer science from Jozsef Attila Science University, Szeged, Hungary, in 1996. He is currently a Software Engineer at the Institute for Software Integrated Systems, Vanderbilt University. His research interests include sensor networks, genetic algorithms and neural networks.*



Zoltán Molnár *received the M.Sc. degree in computer science from the Jozsef Attila Science University, Szeged, Hungary in 1999. Previously he worked in telecommunication field for Nokia Corp. as softare development engineer. He is currently Staff Engineer at the Institute for Software Integrated Systems, Vanderbilt University.*

Peter Volgyesi *is a Research Scientist at the Institute for Software Integrated Systems at Vanderbilt University. His current research interests include model integrated computing, visual programming environments, software engineering of embedded systems and wireless sensor networks. Previously he conducted research on composition and verification technologies for real-time systems at the Embedded Information Technologies Research Group of the Hungarian Academy of Sciences. He received his M.Sc. in Informatics from the Budapest University of Technology and Economics in 2000.*

Miklos Maroti *is an Assistant Professor at Vanderbilt University. His current research interests include algebraic and ordered systems, varieties, decidability, formal specification and analysis of embedded systems, and active libraries of middleware components. He received a PhD in mathematics from Vanderbilt University.*