Towards a Paradigm for Activity Modeling

J. T. Garrett and A. Ledeczi
Institute for Software Integrated Systems, Vanderbilt University
Nashville, TN 37203

F. DeCaria
Old Hickory Plant, E.I. du Pont Nemours and Company
Old Hickory, TN 37216

Abstract

Model Integrated Computing involves defining a domain-specific language that allows for someone to effectively program an environment at whatever level the modeling-environment-creator deems appropriate. We've taken several complex, heterogeneous systems that posed problems of needing integration and requiring frequent reconfiguration at very high levels. This paper discusses gathering the specifications for these systems, how the systems can be represented at these high levels in a paradigm also capturing the specifications, and then how reconfiguring this group can proceed. All of the activities constituting monitoring functionality and the resulting decision making exposed by the information system will be shown to have been solved through utilization of Model Integrated Computing techniques.

1 Introduction

Numerous environments incorporate sensors that log real-time process information. Any combination of these readings over available history may be needed to provide information to an operator or an automated application that can make well-informed modifications to the current environment state. Dispatching this information to sources capable of manipulating it in an intelligent manner can require numerous measures of system integration. Additionally, there is a need to reconfigure this information system to meet demands of new states. An operator monitoring the results from this system who is distanced from the details of the underlying information system would be the desired individual to modify the configuration state. [1]

Configurations for the information system can vary drastically. Modifications may be as simple as moving focus from one group of sensor information to another. They can be as complex as shifting the entire architecture around so that a user interface can now dispatch control information to some other component in the system. Modifying signal processing algorithms that might be applied to the sensor information would be another major change. Manually trying to apply drastic changes to the information system is both time consuming and error prone. The nature of gathering information via numerous data paths is a process whose implementation should be automated.

The Activity Modeling Tool introduced in this paper allows for automatic generation of the target information system following very high level designer specifications. This saves a tremendous amount of time, reduces error introduction, and completely abstracts the modeler from details scattered within the run-time information system.

In this paper we will first summarize Model-Integrated Computing. Then we will talk about the manner in which the Activity Modeling Tool fits within the Model-Integrated Computing framework. After discussing the details of the tool requirements and modeling environment, we will touch on the run-time implementation and the technologies linked together. Finally, we will speak about the system's installation, and realized characteristics, and benefits.

2 Model Integrated Computing (MIC)

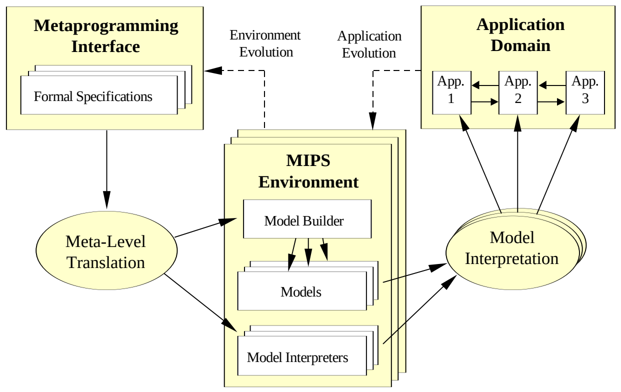

Figure 1: The Multigraph Architecture

Model-Integrated Computing [2] is well suited for the rapid design and implementation of complex computer-based systems. MIC employs domain-specific models to represent the software, its environment, and their relationships. With

Model-Integrated Program Synthesis (MIPS), these models are then used to automatically synthesize the embedded applications and/or generate inputs to COTS analysis tools. 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 MIPS environment for each new domain would be cost-prohibitive for most domains [3]. 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.

Our Multigraph Architecture (MGA) [4] is a toolkit for creating domain-specific MIPS environments. [5] The MGA is illustrated in Figure 1. The metaprogramming interface is used to specify the modeling paradigm of the application domain. The modeling paradigm is the modeling language of the domain specifying the modeling 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) [6] and the Object Constraint Language (OCL) [7], respectively, are used for these purposes in the MGA. These specifications, called metamodels, are used to automatically generate the MIPS environment for the domain. An interesting aspect of this approach is that a MIPS environment itself is used to build the metamodels [8]. The automatically generated domain-specific MIPS 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 COTS analysis tools. This process is called model interpretation.

3 Activity Modeling Tool

The Activity Modeling Tool (AMT) is a MIPS environment for building custom process monitoring and simulation applications for chemical plants. AMT provides a means to model all the components necessary (1) to interface to the real-time database gathering plant sensory data, (2) to define custom processing steps, (3) to create an operator interface, and (4) to interface to a COTS process simulator. From this set of integrated models, the interpreters synthesize the components and combine them together to form a custom process monitoring and simulation application.

3.1 Basic Modeling Overview

The core building blocks within this domain revolve around `Processing` elements. Because the entire AMT paradigm is quite complex, only the salient features for the similar features between these building blocks will be shown. Figure 2 depicts this block.

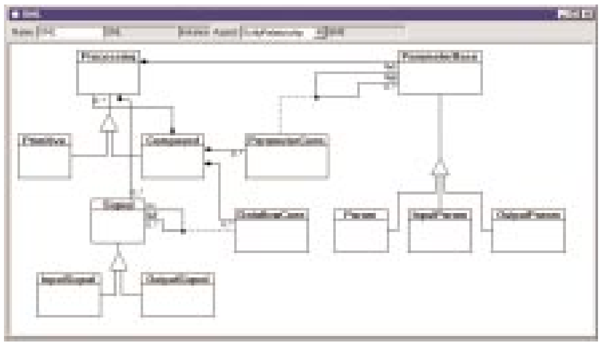

Figure 2: Partial Metamodel of the AMT Environment

In the GME metamodeling environment, the syntax and part of the static semantics of the domain are captured using UML class diagrams. In the AMT paradigm, the `Compound`, `Primitive`, `InputSignal` and `OutputSignal` classes are the basic building blocks of the hierarchical Signal Flow models. The `Processing` and `Signal` classes are abstract base classes that are used to simplify the metamodel itself. `Primitives` are the elementary objects that correspond to computations at the subroutine level in the target domain. The subroutine is specified using the script attribute. In the context of the signal flow models, `Primitives` can contain `Input` and `OutputSignals` (through the aggregation inherited from the Processing class), but not `Processings`. On the other hand, `Compounds` can contain `Processings`, i.e. `Primitives` and `Compounds`, creating a hierarchy of possibly unlimited depth. `DataflowConn` is an association among `Signals`. Its name indicates that this association is implemented using connections as specified in the presentation aspect of the metamodel (not shown in Figure 2).

The core elements modeled within this domain represent the different systems being integrated. Such an integration may be seen in an AMT instance, shown in Figure 3. This high level view demonstrates the manner in which system integration proceeds. The `Net_Data_Interface` represents the sensor or variable database. The `Aspen_Interface` represents the simulation COTS tool. The `Graphical_User_Interface` model represents the operators interaction panel with the environment. Finally, the `Sim_Data_Holder` represents a processing block that works with data moving through the signal flow pipeline.

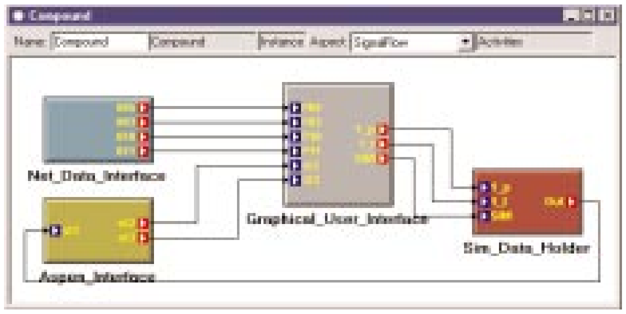

Figure 3: AMT Instance, Highest Level

Figure 3 enumerates the model representations of the components desiring integration. All of these components build from the structure presented in Figure 2 with some extensions visually omitted for brevity. Core to these components is the means for receiving and transmitting information (this is modeled in Figure 2 through the Processing class's aggregation of the Signal basetype). Although `Signal` instances have different meanings within each of the components, they share the notion of being a communication point. This commonality is also shared through the configuration of the run-time kernel used in the target and the pathways used for data propagation.

Each `Signal` is associated with a data type. This can be a simple built-in type such as a double, float, integer or character or an array of any one of these, or a user-defined aggregate type. Aggregate types are explicitly modeled by combining single (and arrays of) built-in types. This `Signal` typing is easily specified within each component instance. Included for the aggregate signal types are elements (`Converters`, not shown in Figure 2) that allow for breaking signals into constituent parts or combining simple types into complex types. Merging several simple streams into a larger structure helps reduce higher level interconnection visual complexity if a logical grouping can be developed.

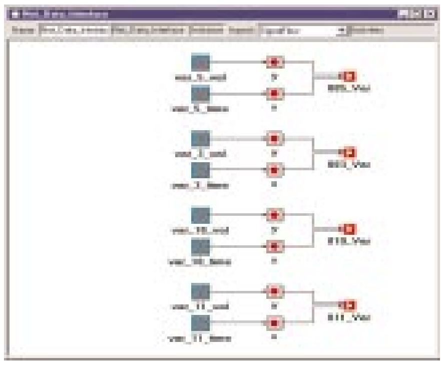

Figure 4: Signal Flow Aspect of Model "Net_Data"

After the syntactic and semantic information have been captured by the metamodels, the presentation specifications need to be added. This step is very specific to the MGA and is beyond the scope of this paper. [8] Once the metamodels are finished, the meta-level translator generates the necessary configuration files to configure a new MGA environment specific to the new paradigm.

In this generated environment, the Signal Flow and the Data Type specifications are shown in different aspects (as specified by the presentation specifications in the metamodels). Figure 4 depicts the Signal Flow aspect of the Net_Data model depicted in Figure 2. This simple model shows four `OutputSignals` that are fed information from elements representing variables from the variable database. An intermediate component is also used ("x" and "y" instances) to combine the variable information into a logical structure used for propagating (x,y) data pairs.

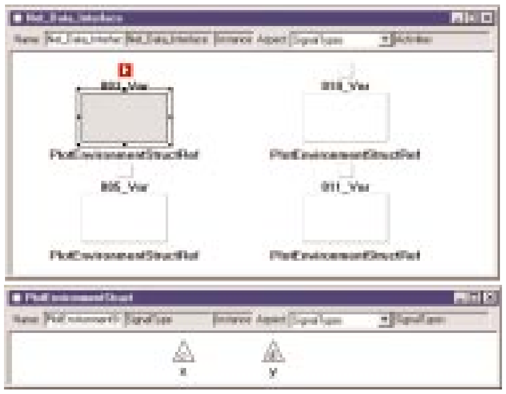

Figure 5: Signal Type Aspect of Model "Net_Data" (Top), and Contents of Referred-To Aggregate Type (Bottom)

Type specification for all the `OutputSignals` can be found in the Signal Types aspect of the same model depicted in Figure 5. There, associations, referred to as a Conditionalization feature within the model editor, are made between type instances and the desired `Signal` to be typed. For example, the conditionalization of the `003_Var` element by the `PlotEnvironmentStructRef` is shown via screendump after performing an operation (right mouse click while in the proper editing mode) to indicate what is being conditionalized by this element. For completeness sake, the contents of the model being referred by the `PlotEnvironmentStructRef` is shown below the top view.

The correctness of signal type specification is enforced by a set of constraints. By applying them, the constraint manager component makes sure that every `InputSignal` and `OutputSignal` has exactly one type associated with it. It also guarantees that signals at each end of a connection are of the same type and have the same size attribute. If signals are conditionalized by aggregate types, then it is ensured that the types are identical in composition.

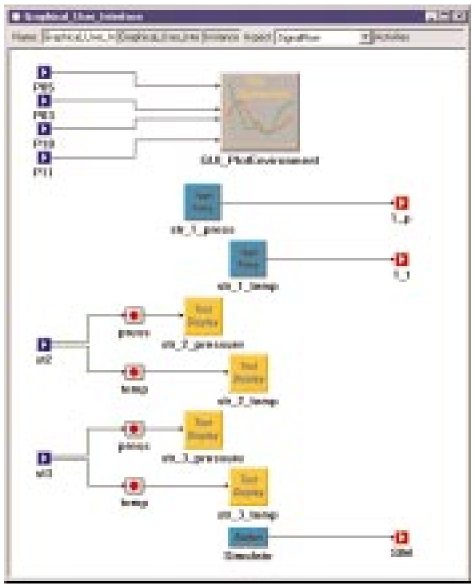

Figure 6: GUI Model
in Signal Flow Aspect

3.2 Additional Component Characteristics

Some additional characteristics of the other system components will now be discussed. The essential features of the variable database have been discussed; variable elements shown in Figure 4 have attributes associated with them identifying their counterpart in the actual database. Similarly, the constituent elements for the process simulation engine work through specifying these peers on the simulation side. Elements representing different variable quantities can have `Signals` entering them, and variable quantities for which inspection is desired can have emerging `Signals`. Feedback scenarios can easily be achieved within the AMT architecture.

The user interface portion of the AMT has no COTS component associated with it, thus more specification can be made through the modeling environment. Development of the user interface proceeded alongside that of the AMT. It deviates the most from the core Signal Flow concepts introduced in Figure 2. Description will proceed through inspection of the abstracted high-level layer depicted in Figure 3.

The GUI can accept and supply `Signals`. Its presentation is based on modeling its behavior through selection and interconnection of various components. Figure 6 shows the most descriptive view of the GUI: the Signal Flow aspect.

Within the Signal Flow aspect of the GUI, one chooses the visual components they wish to appear. Each component has associated with it an inherent distinction made regarding its ability to send, receive, or send/receive information. The middle elements shown in Figure 6 are (from top to bottom) a PlotEnvironment, two text entry fields, four text display fields, and a button. The data directions are obvious for these components. One can see that the same Signal Flow characteristics developed earlier are present within the GUI: that of Signals and Converters.

Additional aspects are also used for modeling the GUI. Similar specification of the Signals is necessary. Additionally, an intuitive manner for graphically placing the interface objects depicted in Figure 6 is accomplished again through the conditionalization feature.

4 Run-Time Environment

Specification of the information system discussed in the Introduction has been reviewed through the use of the GME's multi-aspect, containment based modeling features. Constraint checking has been built into the AMT to ensure that all specifications have been supplied for the target environment. Pending this complete specification, synthesis of the run-time environment can proceed.

The MultiGraph Kernel (MGK) component of the MGA [4] tightly couples the visual interconnections used with a programming interface that can accept configuration easily through modeling interpretation. It is designed to handle the propagation of simple and complex data-typed `Signals`, the execution of scripts associated with our `Processing` elements, and any queuing needs that are either directly specified through `Processing` nodes or implicitly used to implement other system component needs.

The MGK requires an envelope that can dispatch it control to perform data propagation whenever system idle time allows. This envelope uniting all of the components is a standalone executable written with Microsoft Visual C++, running under Win32 platforms. It performs the following tasks: (1) communicating with the process simulation engine, (2) communicating with the variable database, (3) communicating with the remote user interface, and (4) relinquishing control to the MGK for data propagation among components pending the data collection and marshalling steps found in 1, 2, and 3.

Synthesizing this run-time environment takes place through the model-interpreters creation of both a specifications file and C++ code generation. Code generation commences to replace files in the "envelope" server project workspace. Recompiling this workspace creates a new server highly suited to perform the tasks specified. A configuration file is generated that will be dispatched to the remote GUI pending its instantiation. This will allow the GUI to reconfigure itself visually based on the component selection made during modeling.

The process simulation engine interfaced with is Aspen Plus Steady State simulation software [8]. Aspen Plus v10.1-0 exposes a COM interface that has been hooked into by our server. Additionally, it has a canonical way of referencing desired quantities that allows for easily capturing model specifications. Specifying attributes for Aspen at different abstraction layers allow for separating configuration items subject to change through model reconfiguration. Typically all that requires specification are a project file that Aspen loads and then the appropriate information for supplying and receiving information pertinent to this project. Aspen's own GUI allows for further configuration of where its simulation engine resides. The location of Aspen's simulation engine may range from the local PC to a remote cluster of computers or supercomputers.

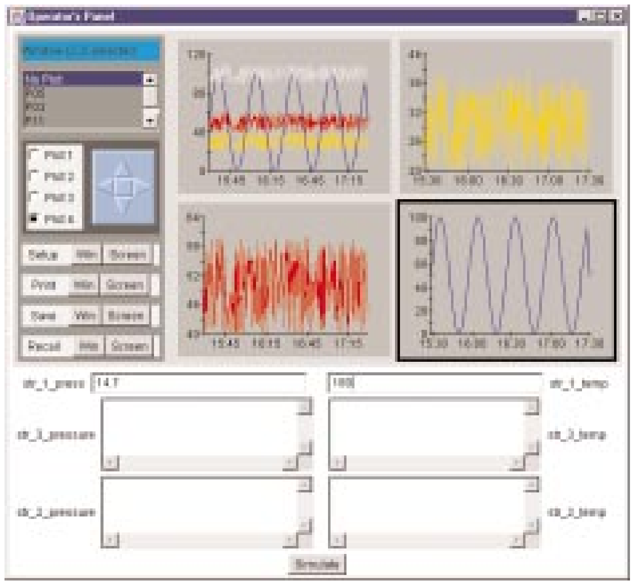

Figure 7: Client's GUI

The next system is the Vantage Process Monitoring and Control (PM&C) variable database running remotely on a VAX machine. Interfacing with this is simplified through a Windows wrapper called Net_Data [9]. Uniquely identifying environment specific settings is very easy. Variables are referred to with a simple number. Connection information is unlikely to change, but modifiable through global attributes for the Net_Data model in the modeling environment. The location of the Vantage PM&C variable database can be anywhere. The Net_Data wrapper for Vantage can communicate over TCP/IP and DECnet.

The final system allowed, at this point, for integration within the AMT is the GUI. The GUI was developed in tandem with the AMT Paradigm and the model interpreter used in conjunction with this paradigm. It is implemented in the form of a Java Applet (or application) that can be located anywhere a network connection permits a TCP/IP connection over a model-time specifiable port to our synthesized server. This might be a web page set as the default for a remote monitoring "terminal" located across the world, or a stand alone application on the same machine that the server is running. Communication between client and server uses a proprietary message based protocol built atop TCP/IP. As mentioned beforehand, when the client is instantiated, the server will transmit the configuration settings to it that were generated from the modeling environment. The client will then create the appropriate visual components and allow user interaction to commence.

Figure 7 depicts the client's rendition of the target modeling environment specified in Figure 6. Visual placement and size of the components adheres to the modeler's choice as specified in another GUI model aspect. The PlotEnvironment occupies the majority of the view at the top of Figure 7. Then the next row is occupied by text entry fields named "str_1_press" and "str_1_temp". They are designed to allow for a user to specify information to be supplied to the simulation engine. One can observe this through Signal Flow aspect inspection of Figures 3 and 6. On the two subsequent rows reside text output fields bearing names "str_3_pressure" through "str_2_temp". Specifically, these entry and display elements correspond to modifying stream pressures and temperatures entering a chemical flash project, and the resulting streams from this project. The final row, with the Simulate button, allows the user to asynchronously initiate this simulation.

5 Conclusions

A prototype has been installed and is being evaluated at the DuPont Chemical Corporation's Old Hickory DMT plant. This prototype allows for a modeler to reconfigure the system rapidly. Instances of the server configuration can be installed on one or several machines allowing for a client to simultaneously monitor different environmental settings by instantiating multiple clients on the same remote machine.

Ideas have already been brought forth about how to expand the existing framework to work with Dynamic Simulations and also in expanding the assortment of components available for placement within the GUI. This type of extension will prove both easy to implement at a modeling level and routine to implement at the programming level beneath. Once the changes have been made to the modeling environment and run-time infrastructure, previous models can be recreated or migrated into the new paradigm. The new functionality will then be present for making additions to existing models.

Both a solution to the application and domain specific problems and the domain-relevant need of requiring frequent reconfiguration have been found through utilizing Model-Integrating Computing tactics. The results are far superior to "manual" integration strategies. Aside from solving the system integration problems at hand, the individual designing the environments can focus on the semantics of the constituent systems rather than their interconnection syntax. This is due to the efforts placed in moving the difficulties associated with system integration into capturing environment characteristics, creating a framework that is capable of having modular modifications made to its structure, and using the modeled information to synthesize a target environment.

Acknowledgements

Research efforts for this project were funded in part by the DuPont Chemical Corporation, Old Hickory Plant, Tennessee.

References

[1] G. Karsai and F. DeCaria, *Model-Integrated On-line Problem-Solving Environment for Chemical Engineering*, IFAC Control Engineering Practice, Vol. 5, No. 5, pp. 1-9, 1997.

[2] J. Sztipanovits and G. Karsai, *Model-Integrated Computing*, IEEE Computer, pp. 110-112, April, 1997.

[3] A. Ledeczi, M. Maroti, G. Karsai, G. Nordstrom, *Metaprogrammable Toolkit for Model-Integrated Computing*, Proceedings of the Engineering of Computer Based Systems (ECBS) Conference, pp. 311-317, Nashville, TN, March, 1999.

[4] J. Sztipanovits, G. Karsai, C. Biegl, T. Bapty, A. Ledeczi, A. Misra, *MULTIGRAPH: An Architecture for Model-Integrated Computing*, Proceedings of the International Conference on Engineering of Complex Computer Systems, pp. 361-368, Ft. Lauderdale, Florida, Nov. 6-10, 1995.

[5] A. Ledeczi, M. Maroti, G. Karsai, G. Nordstrom, *Metaprogrammable Toolkit for Model-Integrated Computing*, Proceedings of the Engineering of Computer Based Systems (ECBS) Conference, pp. 311-317, Nashville, TN, March, 1999.

[6] James Rumbaugh, Ivar Jacobson, and Grady Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

[7] Jos Warmer and Anneke Kleppe, *The Object Constraint Language: Precide Modeling with UML*, Addison-Wesley, 1999.

[8] G. Nordstrom, *Formalizing the Specification of Graphical Modeling Languages*, Proceedings of the IEEE Aerospace 2000 Conference, CD-ROM Reference 10.0402, Big Sky, MT, March, 2000.

[9] Aspen Technologies. http://www.aspentech.com/

[10] Steven L. Lightbody, *Net_Data V2.2 Reference Manual*, 1998.