Proceedings of the 1997 Winter Simulation Conference
ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson

AUTOMATING THE METAMODELING PROCESS

Don Caughlin

Space and Flight Systems Laboratory
University of Colorado at Colorado Springs
Colorado Springs, Colorado 80933-7150, U.S.A.

ABSTRACT

Model abstraction using metamodeling has demonstrated the capability to facilitate software reuse, large scale model integration, verification, and validation. Once restricted to static representations that represented the input-output behavior of models, research has developed the capability to build dynamic metamodels. This capability results from a new approach supported by a taxonomy of metamodeling problems, solution structures, and metamodeling methods. The development of the metamodel, however, still requires a thorough understanding of model abstraction, reduced order modeling, and system identification. In addition, even with the most robust procedures it is possible that the desired data generated by a simulation model will not meet the assumptions or numerical requirements of the procedure. Consequently, there is a requirement for a robust metamodeling support system that will support the subject matter expert. Automation of the metamodeling process will assist the analyst who is not familiar with model abstraction techniques but needs to reuse a piece of code, integrate different models, or verify a new version of a simulation. This paper describes the design of a Metamodeling Support System that provides this automation.

1 INTRODUCTION

A metamodel is a mathematical approximation of the system relationships defined by a more detailed model (Caughlin et al. 1997a). Caughlin (1995) introduced a structured approach to metamodeling that separated the procedure into two steps: problem definition and an iterative metamodeling process. While we can generate a metamodel from data generated by any model structure, the discussion in this paper is limited to metamodels of simulations.

We defined a metamodeling problem as the direct sum of the metamodel requirements and the model (simulation) to be approximated. To support this definition, the problem definition step first determines the purpose of the metamodel. In the definition of this purpose we identify the input and response that we are interested in and determine the important characteristics of these data. Also for this purpose, we define the region of interest, validity measures and specify the required validity. In addition to metamodel requirements, problem definition addresses the second part of the direct sum and characterizes the simulation that is the subject of the metamodel. This characterization provides data that can be used to match the simulation's characteristics to the metamodel structure and identification method.

The second portion of the structured approach was an iterative metamodeling process which consists of the following steps (Caughlin 1997c):

1. Select an Experimental Design
2. Run the Simulation
3. Collect Data
4. Select a Metamodel Set
5. Select Identification Methodology
6. Generate the Metamodel

This approach supported development of dynamic metamodels that exhibit memory and can model phenomena where the past influences the future. In addition, a more robust identification procedure was developed that could be applied to a broader range of problems than existing techniques.

The revised process outlined above provides a direct method of sorting through the myriad of decisions necessary to develop a dynamic metamodel and reduces the number of independent decisions required to develop the metamodel. This process is supported by a new taxonomy of problems, structures, and methods and set of computer aided routines that match the problem definition with the simulation characteristics.

Even with a new approach supported by a taxonomy of metamodeling problems, solution structures, and metamodeling methods, the development of the metamodel still required a thorough understanding of model abstraction, reduced order modeling, and system identification.

In addition, even with the most robust procedures it is possible that the desired data generated by a simulation will not meet the assumptions or numerical requirements of the identification procedure.

Consequently, the widespread use of metamodeling as a method of model abstraction requires an automated support system to assist the analyst. This paper describes research into the design of a prototype Metamodeling Support System (MSS) to automate model abstraction. The prototype system will assist the analyst who is not familiar with model abstraction techniques but needs to reuse a piece of code, integrate different models, or verify a new version of a simulation.

2- DEVELOPMENT AND SYSTEM OVERVIEW

The MSS program provides a semi-automated support system to assist an analyst/modeler in developing a metamodeling abstraction of a more detailed model. This system supports the metamodeling approach outlined above and covered in the references.

2.1- Technical Program

The MSS development program is divided into two development phases:

1. Build 1
2. Prototype Metamodeling Support System

The MSS Build 1 establishes the baseline and provides the following capabilities:

1. A metamodeling system based on an object-oriented architecture that is capable of future expansion.
2. The capability to analyze the source code, generate and run the simulation, and gather data.
3. Data storage and analysis routines.
4. Metamodeling routines and procedures to generate and verify the metamodel.

In Build 1, the MSS provides an executive and automated routines to analyze and run the simulations to gather the data for the metamodel. Existing identification algorithms will be incorporated into this system to provide the basic capability to generate metamodels.

The automated system support discussed above will be provided by an expert system. An expert system is the union of declarative knowledge and inference. The knowledge base contains the declarative knowledge. The inference engine controls the application of that knowledge. It is an algorithm that dynamically directs or controls the system when it searches the knowledge base.

The Prototype Metamodeling Support System is a near-term upgrade of the basic Build 1 and adds the following capabilities:

1. An expert system.
2. Supporting Knowledge Base to support decisions required to develop metamodels.

Documentation for the program is provided in a System/Subsystem Specification (SSS), System/Subsystem Design description (S/SDD), System Software Design Description (SDD).

2.2- System Capabilities

The system must provide the general housekeeping, expert system, and knowledge base to support the objectives and decisions outlined in the metamodeling approach shown in Table 1.

There are four general capabilities that must be provided by this system. These areas are the analysis of the simulation, the correlation of the simulation and data with a metamodel structure and identification method, generation of the metamodel, and finally, the analysis of the metamodel.

First, the system needs to handle the general housekeeping associated with any experimental setup such as: user preferences; cataloging the input and output data; associating the data with parameter selections; and tracking the status of the metamodeling session.

Secondly, the system needs to support problem definition. This includes definition of the metamodel purpose and the analysis of the simulation.

Once the metamodeling problem has been defined, the system must support selection of a metamodel representation (structure) and method of identification. Given a structure and method, the system must now parameterize the metamodel. Lastly, the system should support the analysis of the metamodel. These capabilities are covered under the metamodeling process.

Specific, more detailed, requirements to support these capabilities were provided in a Statement of Work, previous research, and an analysis of current trends in model abstraction.

Table 1: Metamodeling Approach

MAJOR AREA	OBJECTIVE	DECISION/ACTION
Problem Definition	Metamodel Purpose	Scope Use
	Simulation Characteristics	External Characteristics Internal Characteristics
Metamodeling Process	System Representation Identification Methodology	System description
		System class
		Metamodel Structure
		Identification Methodology
	Generate and verify metamodel	Experimental Design
		Run the Simulation
		Collect Data
		Generate the Metamodel
		Verify the Metamodel

3 DESIGN PROCESS

The fact that the MSS must interact with a variety of different legacy simulations with unknown structure dictates a robust, modular, scaleable, and extensible design. A point design would not be able to adapt to different model or simulation structures or handle the different types of analysis to be performed. This dictate, and the fact that this was a software development, seemed to demand an Object-Oriented design approach.

System capabilities, however, stem from the requirement to support a structured sequential process. The functionality is process related and does not reside in or be derived from any of the objects that exist in the environment. Also, the MSS is not a component of another system but a system of systems under the supervisory control of the MSS executive. This analysis supports a structured Systems Engineering design approach.

While Systems Engineering provides a high-level functional architecture, Object-Oriented (OO) Modeling and Design generates a set of lower level functions that should (more properly) be called methods or operations. Unfortunately, it is usually not possible to distribute the methods of the OO classes among the different functional elements that result from Systems Engineering. Consequently, at this point there are two incompatible structures. This issue was addressed in Caughlin (1997b). The design of the MSS followed the method proposed in that paper. A summary of the method follows:

1. Follow the standard Systems Engineering process generating the system capabilities with a functional decomposition and allocation of requirements.
2. Initiate the Object-Oriented Modeling and Design process identifying the underlying objects that are basic to the problem at hand. Identify object attributes, operations (methods), relationships and associations. Develop a class structure, prototype code, and data dictionary.
3. Beginning with system capabilities (requirements), define operating "States and Modes" of the system that are consistent with the functional architecture. Display these states and modes in a flow chart.
4. Using the functional capabilities (architecture) and the States and Modes Flow Chart, connect the functionality that comes from the Systems Engineering process to the objects that result from Object Modeling Techniques by the definition of abstract "manager" and "controller" objects that connected the "top down" functionality with the "bottom up" objects.

4 SYSTEM DESIGN

Presentation of the design of the MSS is organized under Requirements Analysis, Functional Design,

Object-Oriented Design, and KnowledgBase Design. This section concludes with the resulting System Architecture.- Requirements Analysis and Functional Design followed the standard Systems Engineering Process (EIA/IS-632 1994).

4.1- Requirements Analysis

An analysis of the required functionality and the process that the MSS is to support led to 511 requirements. Requirements Traceability and Management was accomplished with a CASE tool – Requisite Version 2.0.18.

System Capabilities were organized as follows:

1. Interface Capabilities
   (a) User Login
   (b) Session Establish/Restore
   (c) Session Configure
   (d) Select Operation
2. Problem Definition Capabilities
   (a) Metamodel Purpose
   (b) Simulation Characteristics
3. Metamodel Capabilities
   (a) Select Metamodel Set
   (b) Select Identification Method
   (c) Select Experimental Design
   (d) Run Model
   (e) Fit Metamodel
   (f) Verify Metamodel

Additional (nonfunctional) capabilities and constraints were also identified. Internal and external interfaces were defined.

4.2- Functional Design

System capabilities were decomposed and allocated to functions based on the following required States and Modes: Login (Standby); Configure (Define) Session; Problem Definition; Metamodel; and Maintenance States.- The operating modes are "Manual," "Assisted," and "Automatic" and apply primarily to the Problem Definition and Metamodeling States. These modes determine the level of support provided by the Expert System.

4.2.1- States and Modes

Login (Standby) State. This is the initial state of the system prior to login to the MSS. In this mode, the system will determine who the analyst is, which process is to be modeled, and the status of the process at login. This state allows the analyst to suspend a session and come back to it at a later time. This state operates only in the manual mode although defaults are provided.

Maintenance State. This state allows the various maintenance functions. This state supports file and knowledgebase maintenance. In addition, user profiles and preferences are established in the maintenance state. Again, this state operates only in the manual mode.

Configure Session State. In this state, the analyst defines the objectives of the session. Here we identify the simulation that will be modeled and provide the data that will support the Problem Definition State. This state can operate in both the manual and assisted mode and cannot be exited until all of the data is provided.

Problem Definition State. The Problem Definition State can function in both the manual and assisted modes. This state provides all of the data defined as *a priori* information. There are two major areas that are addressed. The first area is the purpose of the metamodel. The second area is the characteristics of the simulation that is to be metamodeled.

Metamodel State. The Metamodel State provides the ability to complete the metamodeling procedure. These steps include selection of the metamodel set and identification method, selection of the experimental design, running of the model, fitting the metamodel, and finally, verification of the metamodel. This state operates in all modes.

4.2.2- Functions

Analyzing the Required Capabilities with respect to the States and Modes resulted in the following functions for the requirements allocation.

User Interface (UI). The UI component provides the multimedia control and display interface to the user. It interprets and error checks user inputs and it provides graphical, text and video displays, and audible alarms. It displays out-of-tolerance conditions visually and, if it is a critical parameter, audibly.

Data Manager (DM). The DM provides all of the functionality associated with the management of MSS data. As such, the DM supports data requests from all other functions. Data archiving is accomplished on a Load/Save basis as opposed to a data entry basis.

Scenario Manager (SM). The purpose of the SM is to structure and manage the data used to generate the metamodel. The SM provides three different types of support to the MSS.

First, the Scenario Manager supports data gathering for the problem definition steps of the process. At this stage, the Scenario Manager determines prior information for construction of the metamodel.

Next, the Scenario Manager uses the data from problem definition to generate input data for the simulation. The combination of simulation input and output becomes the input for the identification routines that generate the metamodel. The SM manages the input and output data (through the Data Manager) that will be used to generate a metamodel.

Lastly, the Scenario Manager provides the ability to link the various functions to complete the metamodeling process. Development of the metamodel is a multi-stage process. In the first stage we determine the purpose and characteristics of the simulation. Complete determination of the simulation characteristics, however, requires the output data from the simulation which is provided by the Metamodel Manager. Consequently, the process moves from the Scenario Manager to the Metamodel Manager and back to the Scenario Manager. The last type of support provides by the SM is in tracking this interaction.

These stages are iterative and the sequence of the operation can vary depending on the data and the outcome. Based on data from the Problem Status File generated by the Session Manager (discussed below), the Scenario Manager first determines the status of the solution and what data is required to proceed to the next step of the metamodeling process. From this analysis, the proper SM response is selected.

Metamodel Manager (MM). The MM provides the capability to generate the metamodel.

The first step is to postulate a metamodel. The MM assists with the initial definition of the metamodel structure and guides the selection of the metamodel set. The MM should provide a recommended metamodel set based on the problem and the simulation that is to be metamodeled.

Given the metamodel set, the next decision is the selection of the ID methodology. When we have established the metamodel set we should compare the metamodel set to the metamodeling problem to insure consistency of the metamodel problem. With the metamodel set and ID methodology determined, we use this information to define requirements for the Experimental Design. These selections constrain the Experimental Design and define the Input-Output Data requirements.

The MM subsystem should provide a recommended experimental design based on the problem definition, metamodel set, and ID methodology.

Once the experimental design is defined it should also be compared to the metamodel problem to insure that the design and problem are consistent. Based on the metamodel set, ID methodology, and Experimental Design, we can identify appropriate analysis tools. This step also identifies preprocessing data analysis required to verify the results of the design.

Once the metamodel set, ID methodology, Experimental Design, and analysis tools are defined the simulation controller can configure data capture files and then run the simulation to generate the output data. The I/O must be configured for each simulation along with the simulation run times and message passing. At this time we load simulation and configuration files and execute the simulations as defined.

This data must be analyzed (before the generation of the metamodel) to insure that it meets the restrictions of the method (Belsley 1980, Ljung 1987). In general, we:

1. Assess for collinearity
2. Remove trends and Outliers
3. Select useful portions
4. Filter to enhance important frequency ranges

The MM now gets metamodel data files and metamodel parameters using the data manager. With the data from the simulation, the metamodel set and the ID methodology, the MM now fits the metamodel to the data. After generation of the metamodel the MM then must verify that the metamodel meets the requirements of the problem definition.

Session Manager (SEM). The SEM manages the status of the current session. First the SEM must identify the user and their status as Expert / Advanced / Novice. The SEM then gets a general idea of what the objective of the session. From these objectives the SEM determines if special resources are required.

The SEM also provides the ability to suspend sessions, recover from, and continue with a previous session if requested.

The SEM then configures and manages the session and the session state via the Login and Session State Files.

We have discussed capabilities (requirements) and a functional decomposition and allocation that meets these requirements. Rather than proceeding with the

design process, our methodology dictates an Object-Oriented approach to the problem. We discuss this analysis next.

4.3- Object-Oriented Design

Beginning with the same problem statement, application of Object-Oriented Modeling and Design to the requirements results in the following primary objects:

1. Analyst
2. Project
3. Problem
4. Simulation
5. Metamodel
6. Metamodel Set
7. Metamodel Parameters
8. Data

This analysis continued with identification of object attributes, operations (methods), relationships and associations. A class structure, prototype code, and data dictionary was developed. Object-Oriented Modeling and Design was supported by OMTool that was developed by General Electric Advanced Concepts Center and implements Object-Oriented Modeling and Design as defined by Rumbaugh (Rumbaugh 1991).

In a typical OO methodology, the next Analysis step is to develop the dynamic model by preparing scenarios of typical interaction sequences, identifying events that occur between objects, preparing an event trace for each scenario and an event flow diagram for the system. A functional model is used to describe the transformation from input to output by determining input and output values and developing data flow diagrams to show functional dependencies and identify constraints.

In the design methodology that we follow, however, this data is provided by the Systems Engineering process. We do not continue with the OO design but use these object classes to populate the "player" or lower level of the architecture.

Manager objects are defined that implement the functionality defined by the system states and modes. The player level identifies the objects that must be addressed, Intermediate level "controller" objects are designed to make the connections between the manager and player levels.

4.4- KnowledgBase Design

Expert System support is provided for two purposes. The first purpose is to assist in execution of the Metamodeling process as we have defined it. The process can be executed using many different sequences. The Expert System constrains this sequence to insure that required information is available at each step and that results conform to assumptions.

The second area of support is assistance in decisions required by the process. Here, we help with selection of the metamodel structure, identification method, analysis tools, etc. This is the Decision Assistance Knowledgebase.

The Metamodeling Process Knowledgebase is part of the original specification since it's contents are well known. The MSS contains the ability to record and incorporate the metamodeling results. The Decision Assistance Knowledgebase will be developed as the MSS is used to generate metamodels by recording decisions and the effectiveness of these decisions.

4.5- System Architecture

The software architecture is a framework for the interconnection of subsystems within some major system - in this case the MSS. Each of these component systems are defined by their capabilities and are composed of functions (subsystems) which in turn are a collection of objects (modules).

The MMS is composed of six levels: the top level, the manager level, the component level, the player level, the data level, and the library level (only 4 are shown in Figure 1). The top level encapsulates the abstraction of the MMS and supports the four sequential processing steps: system login, configure the session, define the metamodeling problem, generate and verify the metamodel. The ability to maintain the system is also provided. This level is described by the "states and modes" of operation.

This functionality is implemented by subsystems derived from the functional allocation by objects of the "manager class." This class is expected to completely support MMS capability requirements in these five processing steps. This class of components are instantiated as the different objects required to provide this functionality. These objects are the User Interface (UI), Data Manager (DM), Scenario Manager (SM), Metamodel Manager (MM), and the Session Manager (SEM).

The lower "player" level consists of the objects that are generated by the Object-Oriented Design. The player level encapsulates the entity object classes that are the inputs and products of the MSS such as the

simulations, metamodeling problems, and metamodels. These are the entities that will be generated and/or manipulated in the course of the generation of the metamodel.

The connection between the manager and player levels is accomplished by the definition of an intermediate level. This intermediate level is the collection of subsystems that perform the various operations of the MSS. In the description of the system, they are called "controllers." The component level "controllers" provide the connection between the managers and player objects.

Table 2 below shows the "Manager" class, the Systems Engineering functions performed by the class and the objects of the OO design from OMT that are affected.

A data level consists of the data objects required to manage the processes and control the products. Lower level library objects also exist that are used to implement standard functions that are not explicitly named.

5- IMPLEMENTATION

Many of the components required to meet the functional requirements of the MMS already exist.

An existing code analyzer can be used to analyze the simulation characteristics. Data can be efficiently stored in any number of relational databases (e.g. external simulation characteristics are already provided in a SIMTAX database (Anderson, et al. 1989).

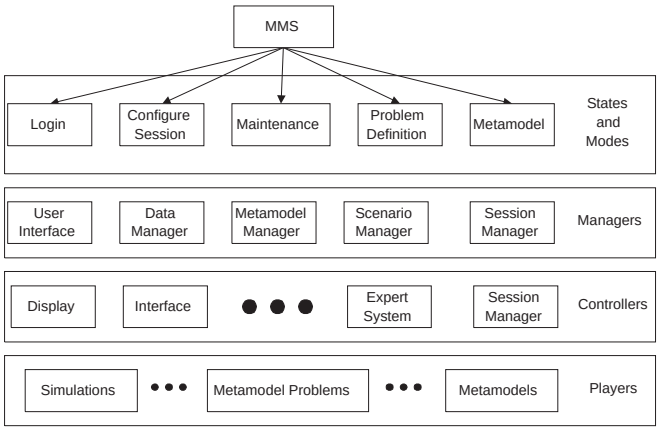

Figure 1: MMS Architecture

In addition, there are a number of expert systems that could be used to provide automation support. Identification and analysis routines are available as well as a number of numerical engines.

Rather than develop all of the components of the MSS, the decision was made to develop a shell or mainframe that would integrate and manage both new and existing components.

This shell was developed with Microsoft Development Studio and C++ language using the Microsoft Component Object Model, the Microsoft Foundation Class Library, and the DAO database interface. MSS targets the Windows NT operating system.

The OO design was accomplished in OMTool. These files will be integrated into the Microsoft Development Studio.

Table 2: Connection Between Objects and Functions

MMS MANAGER	FUNCTIONALITY (SE)	PLAYER (OMT)
User Interface	Interface	Analyst
Data Manager	Interface	Data
Scenario Manager	Problem Definition	Problem
		Simulation
Metamodel Manager	Metamodel	Metamodel
		Metamodel Set
		Metamodel Parameters
Session Manager	Interface	Project

The Expert system is provided by the C Language Integrated Production System (CLIPS) developed by the Software Technology Branch (STB), NASA/Lyndon B. Johnson Space Center. CLIPS is designed to facilitate the development of software to model human knowledge or expertise. Rules and objects form an integrated system since rules can pattern-match on facts and objects. In addition to being used as a stand-alone tool, CLIPS can be called from a procedural language, perform its function, and then return control back to the calling program.

CLIPS was embedded into the MSS as a DLL using a Wrapper Class provided by Mark Tomlinson (MTOMLINS@us.oracle.com).

The numerical engine is provided by MATLAB. Identification and analysis tools are incorporated as MATLAB "M" files.

Documentation for the system is provided in the form of Windows help files which are assessable online.

6. SUMMARY

This paper has described the design and capabilities of a prototype Metamodeling Support System that will assist the analyst who is not familiar with model abstraction techniques but needs to reuse a piece of code, integrate different models, or verify a new version of a simulation.

We presented an outline of the capabilities required to support the Metamodeling process and references where details may be found.

We demonstrated the use of a design process that integrated Systems Engineering and Object-Oriented Modeling and Design to provide a system architecture that meets functional requirements and accommodates an Object-Oriented framework.

Unfortunately, the scope of the paper does not allow a complete description of implementation details. A summary was provided.

ACKNOWLEDGMENTS

This research was partially supported by Rome Laboratory under Contract No. F30602-96-C-0040/P0001.

REFERENCES

Anderson, L. B., et al. 1989. SIMTAX, A Taxonomy for Warfare Simulation, Workshop report taken from the *Catalog of Wargaming and Military Simulation Models, 11th Edition*, Force Structure, Resource, and Assignment Directorate (J-8), The Joint Staff, Washington, DC 20318-8000.

Belsley, D, E. Kuh, R. Welsch. 1980. *Regression Diagnostics*. New York:John Wiley & Sons.

Caughlin, D. 1995. *Final Report, Modeling Techniques and Applications, Volume I.* USAF Contract F30602-94-0110, Rome Laboratory/IRAE, 32 Hangar Rd, Griffis AFB, NY 13441-4114.

Caughlin, D.,A. F. Sisti. 1997a. "A Summary of Model Abstraction Techniques". In *Enabling Technology for Simulation Science, Alex. F. Sisti Editor, Proceedings of the SPIE*, Vol. 3083:14–21.

Caughlin, D. 1997b. "Integration of Object-Oriented and Functional Modeling and Design Methods." In *Enabling Technology for Simulation Science, Alex. F. Sisti Editor, Proceedings of the SPIE*, Vol. 3083:89–99.

Caughlin, D. 1997c. "Model Abstraction Via Solution of the Inverse Problem to Define a Reduced Order Model". Accepted for publication in *SCS Transactions*

EIA/IS-632. 1994. *EIA Interim Standard, Systems Engineering*. Electronic Industries Association.

Ljung, L. 1987. *System Identification: Theory for the User*, New Jersey:Prentice-Hall.

Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. 1991. *Object-Oriented Modeling And Design*. New Jersey: Prentice Hall.

AUTHOR BIOGRAPHY

DON CAUGHLIN is Acting Director of the Space and Flight Systems Laboratory at The University of Colorado at Colorado Springs. He received a BS in Physics from the Air Force Academy, an MBA from the University of Utah and MS and Ph.D. degrees in Electrical Engineering from the University of Florida. His research interests include modeling and simulation, system identification, pattern recognition and intelligent control. Dr. Caughlin has over 28 years experience as an experimental test pilot, chief scientist, research scientist, program manager and was also Associate Dean of the School of Engineering at the Air Force Institute of Technology. He is a senior member of IEEE and AIAA and a member of SCS and the Society of Experimental Test Pilots.