AFRL-RI-RS-TR-2007-238
Final Technical Report
November 2007

A TOOLKIT FOR BUILDING HYBRID, MULTI-RESOLUTION PMESII MODELS

Charles River Analytics, Inc.

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

NOTICE AND SIGNATURE PAGE

AFRL-RI-RS-TR-2007-238 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/	/s/
DALE RICHARDS Work Unit Manager	JAMES W. CUSACK Chief, Information Systems Division Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* NOV 2007	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)* May 06 – May 07

4. TITLE AND SUBTITLE

A TOOLKIT FOR BUILDING HYBRID, MULTO-RESOLUTION PMESII MODELS

5a. CONTRACT NUMBER
FA8750-06-C-0078

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62702F

6. AUTHOR(S)

John A. Bachman and Karen A. Harper

5d. PROJECT NUMBER
558S

5e. TASK NUMBER
CP

5f. WORK UNIT NUMBER
E3

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Charles River Analytics, Inc.
625 Mount Auburn St
Cambridge MA 02138-4555

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RISE
525 Brooks Rd
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TR-2007-238

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# WPAFB 07-0302

13. SUPPLEMENTARY NOTES

14. ABSTRACT

A software toolkit for constructing, integrating, debugging, validating and verifying, and maintaining heterogeneous Political, Military, Economic, Social Information, and Infrastructure (PMESII) models in support of a Commander's Predictive Environment (CPE) was developed. This development environment is based upon Charles River Analytic's Graphical Agent Development Environment (GRADE), and provides intuitive graphical tools supporting the development of new models and adaptation of existing PMESII models. The component-based software architecture enables the injection of new modeling paradigms, e.g., semantic networks, system dynamic models, etc., and existing legacy PMESII models; as well as GUI-based model tools to enable data-level integration. The toolkit provides 1) an intuitive graphical model development environment, 2) a suite of model integration tools, 3) a suite of model verification and validation tools, 4) a model analysis infrastructure, 5) a suite of multi-resolution modeling tools and supporting infrastructure, and 6) a model management infrastructure.

15. SUBJECT TERMS
PMESII, Framework, Interoperability, Modeling, Model Integration, Commander's Predictive Environment (CPE)

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Dale W. Richards
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UL	66	**19b. TELEPHONE NUMBER (*Include area code)*** N/A

Table of Contents

Glossary of Abbreviations

AFRL	Air Force Research Laboratory
AFRL/RH	AFRL Human Effectiveness Directorate
AFRL/RI	AFRL Information Directorate
AI	Artificial Intelligence
AMBR	Agent-based Modeling and Behavior Representation
API	Application Programming Interface
C2	Command and Control
COTS	Commercial Off The Shelf
CPE	Commander’s Predictive Environment
CT	Continuous Time
DAGNEG	Distributed Air-Ground Negotiation Optimization
DARPA	Defense Advanced Research Projects Agency
DE	Discrete Event
DIME	Diplomatic, Information, Military, and Economic
DTD	Document Type Definition
FSM	Finite State Machine
GOTS	Government Off The Shelf
GRADE	Graphical Agent Development Environment
GUI	Graphical User Interface
HASMAT	Human and System Modeling and Analysis Toolkit
HBM	Human Behavior Model
IDE	Integrated Development Environment
IHMC	Institute for Human and Machine Cognition
I/O	Input/Output
MATLAB	Mathematical Laboratory (software)
MoML	Modeling Markup Language
NASA	National Aeronautics and Space Administration
NO’EM	National Operational Environment Model
OCCAM	Organizational and Cultural Criteria for Adversary Modeling
PMESII	Political, Military, Economic, Social, Information, and Infrastructure
PRISM	Platform for the Representation of and Inference over Situation-theoretic Models
RTA	Recruiting, Training, Active-duty

SAMPLE	Situation Awareness Model for Person-in-the-Loop Evaluation
SDF	Synchronous DataFlow
SNA	Social Network Analysis
SPSS	Statistical Package for the Social Sciences
SROM	Stabilization and Reconstruction Operations Model
XML	eXtensible Markup Language
XSL	eXtensible Style Language
XSLT	eXtensible Style Language Transformation

List of Figures

List of Tables

Executive Summary

The nature of modern warfare requires that commanders' military strategy development and decision-making be driven by a clear understanding of the complex socio-political dynamics of the operating environment as well as the more typical "metal on metal" analyses of more traditional warfare. To support such understanding, the full range of Political, Military, Economic, Social, Information, and Infrastructure (PMESII) dimensions must be accounted for throughout the command and control (C2) decision-making process. While progress has been made towards the development of a range of modeling frameworks to address the complex dynamics across these dimensions, e.g., system dynamics models, fuzzy networks, Bayesian models, semantic networks, etc., significant work remains to support an integrated view of the system of systems dynamics that incorporates the complex interdependencies across the PMESII dimensions.

To support this integrated system of systems analysis that will enable rapid assessment, effective understanding, and prediction of the dynamic operational context; we have developed a Toolkit for Building Hybrid, Multi-resolution PMESII Models enabling the construction, integration, debugging, validation and verification, and maintenance of heterogeneous PMESII models. The IDE is based upon our existing Graphical Agent Development Environment (GRADE), and provides intuitive graphical tools supporting the development of new models and adaptation of existing PMESII models. It is built on a component-based software architecture that easily enables the injection of new modeling paradigms, e.g., semantic networks, system dynamics models, etc., and existing legacy PMESII models. The PMESII modeling IDE also provides GUI-based model integration tools that enable the data-level integration of heterogeneous modeling paradigms.

Our approach to supporting the objectives of the Commander's Predictive Environment (CPE) provides a unique solution, that:

- Enables the integration of both newly developed modeling paradigms and existing legacy models within aggregate representations of the full PMESII environment through its component-based software architecture,
- Supports the development and adaptation of PMESII models by a wide range of multi-disciplinary model developers through intuitive graphical model specification tools that abstract the underlying complexity of the modeling formalisms; and
- Eases the barrier to entry of into the PMESII modeling environment, in that new technologies can be developed to multiple levels of completeness and functionality, and "play" in the PMESII modeling framework at each level of development.

1. Introduction

Military planning and decision-making for dealing with asymmetric threats embedded in urban environments demands a clear understanding of the complex socio-political context. In terms of planning and executing effects-based operations, this translates to the analysis of the potential effects that a given set of Diplomatic, Information, Military, and Economic (DIME) actions will have across the full range of the Political, Military, Economic, Social, Information, and Infrastructure (PMESII) context. Within the context of DARPA's Integrated Battle Command program (Allen, 2004a), these analyses are viewed in two ways, as shown in Figure 1-1 below. From left to right, the figure shows a causal analysis where, given a set of possible DIME actions to be taken, a system of complex and integrated behavior models are used to *predict the potential effects those actions may have across the PMESII dimensions*. From right to left, the figure shows a diagnostic analysis where, given a set of desired PMESII effects in the operational domain, the same system of integrated behavior models are used to *identify the candidate sets of DIME actions that might be applied to achieve those desired effects*. By conducting both types of analyses—ones that move well beyond the limits of conventional military "metal-on-metal" modeling—commanders will be able to develop significantly deeper insight into the dynamics of the "big picture" operational context.

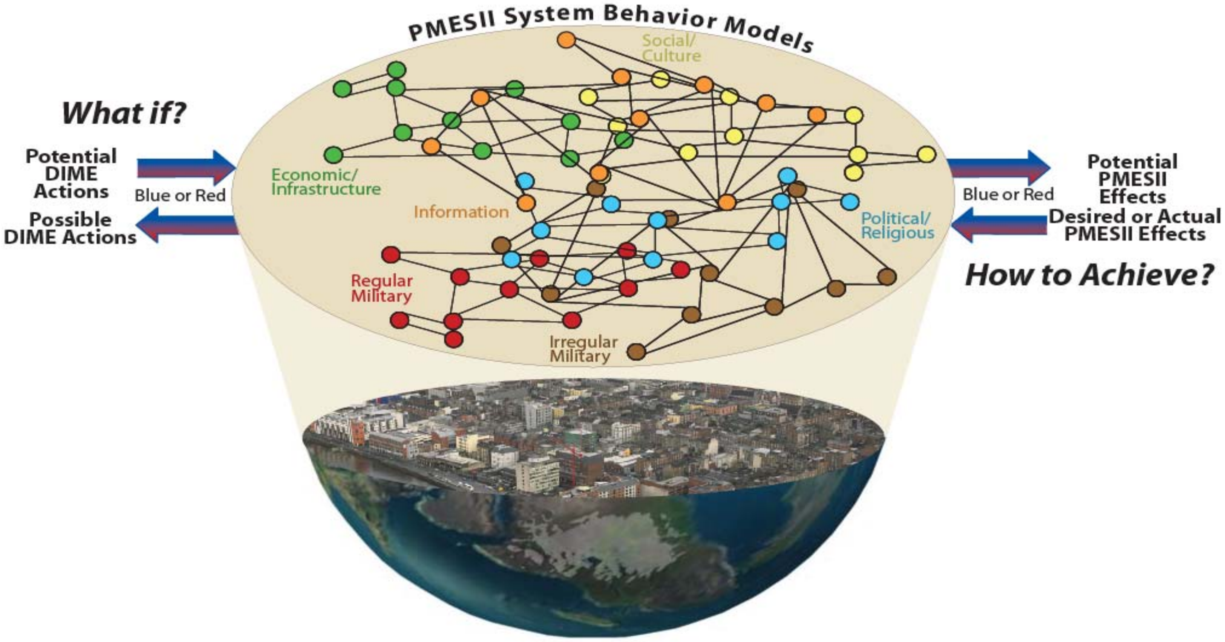


Figure 1-1: Predicting and Analyzing PMESII Effects of DIME Actions (adapted from (Allen, 2004b))

The key to successfully executing such encompassing analyses lies in the development of the embedded behavior models representing the full range of PMESII variables, and how they can be individually and collectively affected by specific DIME actions. For example, Robbins' *Stabilization and Reconstruction Operations Model* (SROM) (Robbins, Deckro, & Wiley, 2005) analyzes the organizational hierarchy, dependencies, interdependencies, exogenous drivers, strengths, and weaknesses of a country's PMESII systems using a complex set of interdependent systems dynamics representations. SROM models a country system in a holistic manner as a National Model, which is then defined in terms of its *n* regional sub-models that interact with each other and the National Model. Each regional sub-module contains six functional sub-models: demographics sub-model, insurgent and coalition military sub-model, critical infrastructure, law enforcement, indigenous security institutions, and public opinion. The utility of SROM is demonstrated using Operation Iraqi Freedom as a case study.

While approaches such as Robbins' SROM have demonstrated some success in modeling subcomponents of the PMESII environment, it is generally accepted that no one approach or modeling formalism can or should be applied to capture *all* of the complex dynamics of modern asymmetric warfare. Rather, what is required is an *integrated modeling capability* that enables: 1) the development of sub-models to represent specific PMESII features using the most appropriate to modeling those features; 2) easy integration of those sub-models into a cohesive and sophisticated representation of the overall PMESII system, and 3) effective accounting of the complex interdependencies between modeled PMESII variables within the integrated system.

To support an integrated system-of-systems analysis of the battlespace that will enable rapid assessment and effective understanding and prediction of the dynamic operational context, we have developed an integrated development environment (IDE) enabling the construction, debugging, validation and verification, and integration of heterogeneous PMESII models. The IDE provides:

- An *intuitive graphical model development environment* supporting the specification of heterogeneous sub-models using a variety of modeling formalisms, e.g., Bayesian reasoning, fuzzy logic, system dynamics models, rule-based expert systems, etc.,
- A suite of *model integration tools* enabling user-driven sharing of data and information among constituent PMESII models,
- A suite of *model verification and validation tools* enabling user-driven verification of individual and integrated PMESII model behavior as well as the large-scale data collection required to support validation of model behavior against empirical data,

- A *model analysis infrastructure* that enables user-driven causal and diagnostic reasoning within the integrated modeling framework using sampling techniques and sensitivity analysis, respectively,
- A suite of *multi-resolution modeling tools and supporting infrastructure* to support the user-driven specification of PMESII sub-models at multiple levels of modeling fidelity, and
- A *model management infrastructure* that enables the capture, distribution, and maintenance of large libraries of PMESII sub-models.

The IDE is based upon our existing Graphical Agent Development Environment (GRADE), which provides intuitive model construction, verification, and visualization tools supporting the development of integrated heterogeneous reasoning models. GRADE incorporates fuzzy inferencing, Bayesian reasoning, and expert systems as its core feature set, but it is built on a component-based software architecture that easily enables the injection of new modeling paradigms, e.g., semantic networks, system dynamics models, etc., which was our primary focus under this effort. GRADE also provides GUI-based model integration tools that enable the data-level integration of heterogeneous modeling components. Under a concurrent effort by Charles River Analytics Inc., "Framework for Building and Reasoning with Adaptive and Interoperable PMESII Models" (Air Force contract FA8750-06-C-0076), we have also explored strategies to augment GRADE's model integration capabilities to support more abstract and sophisticated techniques beyond GRADE's current data-level integration tools.

1.1 Technical Objectives

Our approach to the design of the proposed PMESII modeling IDE focused on the following technical objectives:

- Identify the technical requirements for a toolkit supporting the creation, validation, and analysis of PMESII models,
- Select and develop PMESII modeling technologies and incorporate them into the toolkit,
- Develop model integration tools to allow PMESII model components to interoperate,
- Develop PMESII model verification and validation tools to analyze and validate model component behavior,
- Develop PMESII model analysis infrastructure to support causal and diagnostic reasoning,
- Develop multi-resolution modeling tools, and

- Develop PMESII model management infrastructure to enable sharing and re-use of models among a community of model developers.

In the next section, we summarize how we have addressed these objectives.

1.2 Summary of Technical Approach and Results

During the course of our CPE Toolkit development effort, we accomplished the following results:

First, we performed a review of the state of the art in PMESII modeling capabilities to develop further insight into the full range of technical requirements to support the predictive modeling of PMESII effects as a function of possible DIME actions. The results of this review were used to identify the current and future technical requirements for the proposed PMESII modeling IDE. Our review and requirements analysis was informed by the ongoing work of AFRL/RI NO'EM research group, who shared several of their SROM models with our team.

Second, we reviewed a variety of PMESII modeling tools and chose Ptolemy as the focus of our integration effort. Based on the understanding gained through the requirements specification process, we selected, in consultation with AFRL, the Ptolemy systems dynamics modeling framework to be the primary focus of our modeling tool integration effort. As Ptolemy is the platform currently being used by the NO'EM group for PMESII model development, this choice had a number of distinct advantages: immediate relevance to the CPE domain, availability of existing models for use as examples, Java implementation with a non-restrictive license, storage of models using the Modeling Markup Language (MoML), a well-defined XML file format with a published Document Type Definition (DTD), and an existing graphical editor (Vergil) for creating and maintaining Ptolemy models.

Third, we researched advanced strategies for model integration in partnership with our companion CPE program. While GRADE provides a base level of support for interoperable PMESII modeling by virtue of the fact that it provides a common environment for model development and a common protocol (XML) in which they can communicate, there are a number of challenges associated with resolving interoperability conflicts between PMESII models beyond the data-level. During this effort we worked with the team of our sister CPE effort, "Framework for Building and Reasoning with Adaptive and Interoperable PMESII Models," to conduct research into advanced strategies for resolving model interoperability conflicts and explore practical approaches for incorporating these strategies into the PMESII modeling toolkit.

Fourth, we developed an enhanced suite of PMESII model verification and validation tools. GRADE provides a suite of model verification and validation capabilities that can be applied by the model developer to analyze model behavior, both at the individual component level and at the integrated model level, to ensure that models behave as intended by the developer. We enhanced these features by incorporating the open-source testing framework XmlUnit for model and sub-model verification, and by developing a Graphing Component to allow the model developer to visualize model output for both verification and validation.

Fifth, we developed an enhanced PMESII model analysis infrastructure. The end goal of the CPE program is to provide a modeling environment that will enable both the causal and diagnostic analysis of the complex operational environment of modern warfare, so that commanders can effectively predict the potential effects of candidate DIME actions or generate potential DIME actions that are most likely to generate some set of desired PMESII effects. In support of this objective, we created an Analyst's Interface allowing the model analyst to perform model validation and analysis at a higher levels of abstraction; we developed a Graphing Component, as indicated above, to allow the model developer to visualize model-generated data within the PMESII modeling IDE; and we developed a framework for advanced analytical techniques under HASMAT (Air Force contract FA8650-06-C-6731), a related program for simulation-based model analysis.

Sixth, we developed strategies for supporting multi-resolution modeling. It is rarely the case that all potential behaviors in a given scenario must be represented at high fidelity levels. Rather, only those behaviors that are likely to have the most impact on a PMESII outcome of interest need to be modeled, and likely only a subset of them need to be modeled at high levels of detail or fidelity, at different times over the course of a scenario. What this calls for is a mechanism by which models of varying fidelity can be "switched" in and out of the larger PMESII model environment. During this effort we explored the use of GRADE models as a means of controlling this switching process for the PMESII domain, as informed by our previous work on multi-resolution modeling in other domains.

Finally, we developed a model management infrastructure for the PMESII modeling toolkit. The development of practical PMESII models that will accurately reflect the complex operational environments of urban and national warfare will require model inputs and expertise from a wide range of disciplines, including economics, political science, sociology, psychology, and cultural anthropology. To support the development of such multi-disciplinary models our toolkit must support development and adaptation by multiple users, including maintenance of individual models and model libraries. To meet the needs of both individual model developers that require model development capabilities, as well as groups of developers that require change tracking and collaboration support, we integrated Subversion, an existing open-source public-license version control technology, within GRADE, which enables the development of shared libraries of models and careful maintenance of model development histories.

1.3 Report Outline

In Section 2, we present an overview of the GRADE graphical model development IDE. In Section 3, we describe the modeling technologies we reviewed as candidates for integration, and discuss our effort to integrate the Ptolemy modeling framework into GRADE as a component. In section 4, we discuss our research into providing enhanced support for model interoperability. In Section 5 we discuss our enhancements to GRADE for model verification and validation. In Section 6 we discuss our development effort to support model analysis, including the Analyst's Interface, the Graphing Component for GRADE, and analytical capabilities developed under HASMAT. In Section 7 we discuss our approach to supporting multi-resolution modeling. Finally, in Section 8 we describe the model management infrastructure we integrated into GRADE using the open-source version control package Subversion.

In Table 1-1 below, we provide a summary of the pre-existing capabilities of GRADE as well as the enhancements we have made to it under the program. The table is organized based on the modeling requirements laid out in the CPE solicitation that are deemed to be enabled specifically by the PMESII IDE, and identifies: 1) capabilities initially supported by GRADE, 2) capabilities that we considered to be incremental enhancements to the toolkit, and 3) capabilities that we considered as requiring advanced design and development. We also identify the report section that describes our approach to supporting the enhanced functionality.

Table 1-1: Overview of Enhanced GRADE Toolset for Heterogeneous PMESII Models

CPE Requirement	Pre-Existing GRADE Capability	Increment GRADE Capability	Advanced GRADE Capability	GRADE Enhancement Effort	Report Section
Heterogeneous modeling framework	☑		☑	o GRADE's component-based architecture enables straightforward integration of new modeling frameworks o Program effort focused on adding to GRADE's suite of modeling components by incorporating the Ptolemy systems dynamics modeling tool	2 3
Multi-resolution modeling			☑	o Program enables multi-resolution modeling through GRADE infrastructure to enable user specification of multiple models representing multiple levels of abstraction	7
Model validation and verification	☑		☑	o GRADE provides in-depth model verification capabilities allowing user to generate model inputs, and analyze model responses in step-through manner o Program supports large-scale data capture of model responses to enable comparison with empirical data for model validation	2.2, 2.3 5
Dynamic update of schema data/trends	☑			o GRADE supports dynamic update of data input/output schemas as a function of user-driven model specification	2
Model editors/ browsers/ inspectors	☑		☑	o GRADE provides visual editors supporting model editing, browsing, and inspection (i.e., visualization of model state throughout analysis) o Program augments visual components with corresponding editors/browsers/inspectors for Ptolemy component	2.1, 2.2 3.2.2, 3.2.3
Debugging	☑			o GRADE enables debugging of integrated models at both the system level and at the component level o User can "post" data to models in stand-alone fashion and monitor model's responses	2.2
Version control/model reuse		☑		o Program incorporates infrastructure enhancements to GRADE to enable the maintenance and dissemination of models through a shared library	8
Common model APIs and interfaces	☑			o GRADE's component-based software infrastructure enforces the use of common model APIs within the framework	2.4

CPE Requirement	Pre-Existing GRADE Capability	Increment GRADE Capability	Advanced GRADE Capability	GRADE Enhancement Effort	Report Section
Interoperable data and model schema (ontologies)	☑		☑	o GRADE supports interoperability between component models at the data level through data mapping interfaces (see below) o Advanced model integration capabilities would be supported through a parallel effort described in the final report for Charles River's (Framework for Building and Reasoning with Adaptive and Interoperable PMESII Models)	2 4
Model producer and consumer GUIs	☑			o GRADE's data mapping interface provides base-level support for translating model component outputs (producers) to model input (consumers)	2
Mapping of PMESII attributes to DIME actions and predictive capabilities			☑	o Program augments GRADE's component suite to include a component for graphing model output for analysis o Ongoing work incorporates advanced software infrastructure allowing for sophisticated model execution control for large-scale data collection and analysis supporting predictive capabilities	6

2. Overview of GRADE

Our framework for integration of heterogeneous PMESII models is based upon our Graphical Agent Development Environment (GRADE), which provides intuitive model construction, verification, and visualization tools supporting the development of integrated heterogeneous reasoning models. Originally targeted as a development environment for human behavior models based on Charles River Analytics' SAMPLE cognitive architecture, GRADE has grown into a flexible information processing architecture for the rapid construction and integration of a broad array of reasoning models. While it incorporates fuzzy inferencing, Bayesian reasoning, and expert systems as its elementary reasoning tools, GRADE's component API allows new tools and models to be rapidly adapted for and integrated into its overarching modeling framework. In this section we provide an overview of GRADE's capabilities as they apply to the challenge of developing and integrating PMESII models, focusing specifically on GRADE's graphical interfaces for model construction, verification, and visualization, as well as its component-based architecture.

2.1 Graphical IDE for Model Construction

A model within GRADE is represented as a network of components that exchange information through messages. The individual reasoning nodes in the network can be implementations of any algorithm or processing model, wrapped inside a component that implements the GRADE component API. GRADE's flexible component-based architecture allows diverse technologies to be integrated into a common framework, enables the reuse of existing software through encapsulation of existing software as GRADE components, and facilitates the rapid creation of new components that can operate within the GRADE framework.

GRADE provides an intuitive graphical interface allowing the user to both define model topology and drill down into individual components to configure the underlying processing models. The IDE includes the tools used to select and link components together as well as the interfaces used to configure component behavior. The main window frame of Figure 2-1 shows the base interface for the agent model. The user can add components and define data flow between components of the model within this interface.

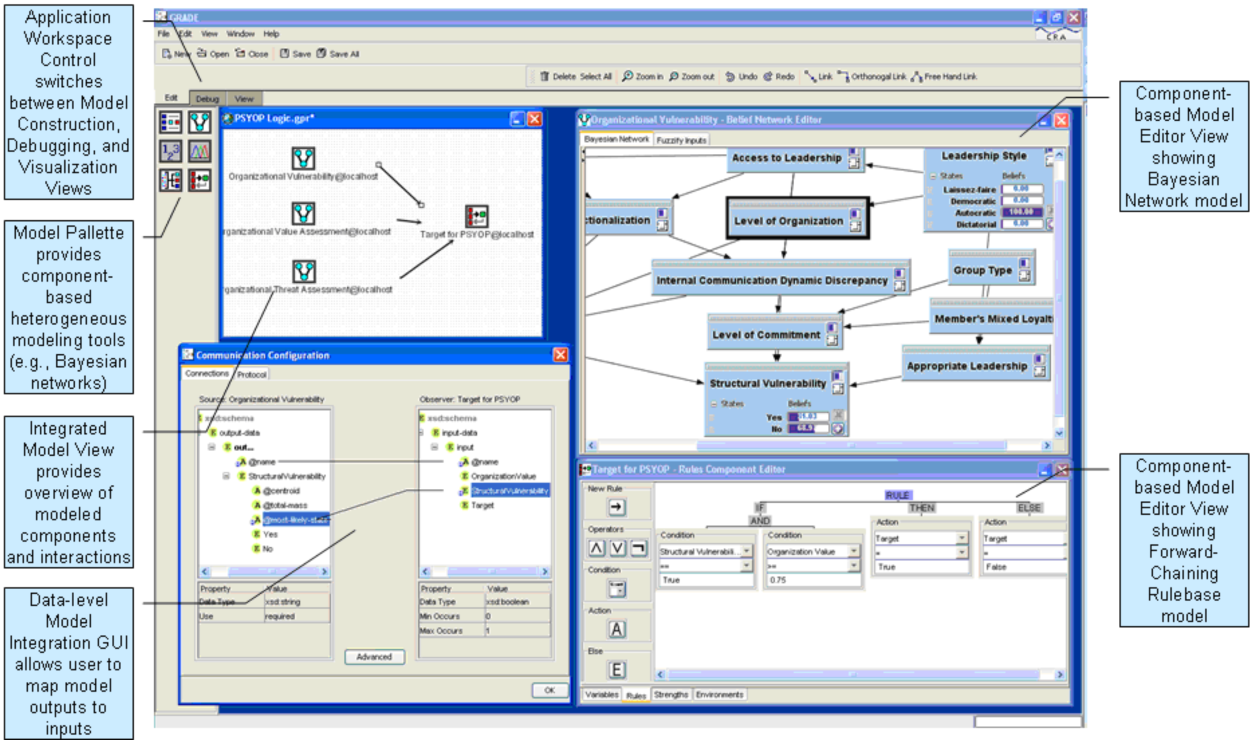
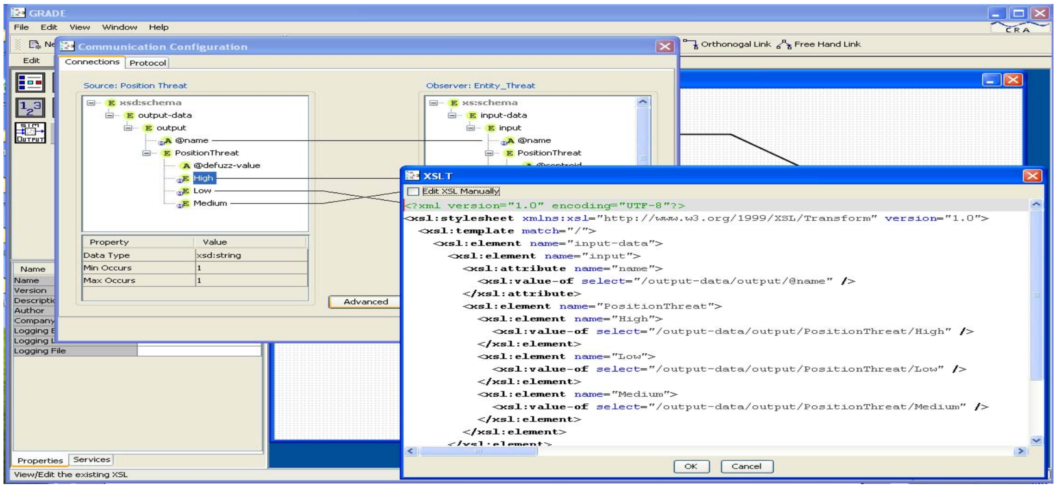

Application Workspace Control switches between Model Construction, Debugging, and Visualization Views
Model Pallette provides component-based heterogeneous modeling tools (e.g., Bayesian networks)
Integrated Model View provides overview of modeled components and interactions
Data-level Model Integration GUI allows user to map model outputs to inputs
Component-based Model Editor View showing Bayesian Network model
Component-based Model Editor View showing Forward-Chaining Rulebase model
GRADE
PSYOP Logic.gpr*
Organizational Vulnerability@localhost
Organizational Value Assessment@localhost
Organizational Threat Assessment@localhost
Target for PSYOP@localhost
Communication Configuration
Organizational Vulnerability - Belief Network Editor
Access to Leadership
Leadership Style
Level of Organization
Internal Communication Dynamic Discrepancy
Group Type
Member's Mixed Loyalti
Level of Commitment
Appropriate Leadership
Structural Vulnerability
Target for PSYOP - Rules Component Editor
RULE
IF
THEN
ELSE
AND

Figure 2-1: Building a Reasoning Model in GRADE

On the left side of the interface is a palette from which the user can select new components to add to the network. The user adds new modules to the integrated model by selecting a particular component type in the palette and dragging the node into the drawing area. Once a component has been added to a model, it can be configured by double-clicking its icon to access the component's Editor GUI. GRADE displays the menu bar and toolbar that are appropriate for that type of component.

The user specifies data interfaces between nodes by drawing links between the sending node and the receiving node in the agent network. In GRADE, components communicate by exchanging XML messages, with each component specifying its own input and output message formats using XML schemas. When two components are linked together, messages passed between them are translated from the output schema of one to the input schema of the other using Extensible Style Language Transformations (XSLT). Each component is therefore functionally decoupled from any other component, which allows current GRADE components, future components, and third-party components to communicate with one another without introducing complex dependencies between components.

To facilitate the rapid linking of components, GRADE incorporates a graphical XSLT Generator. This tool allows model developers to automatically define transformations between message formats through the use of a simple drag-and-drop interface. Figure 2-2 shows the XSLT Generator Dialog. Each communications link acts to connect a source component, which produces output messages, with an observer component that receives those messages. The left side of this dialog shows the format of messages sent out by the source component. In this case, the source component is an instance of our Belief Network Component. The right side of the dialog shows the format of the messages that may be received by the observer component. In this case, the observer is an instance of our Rule Base Component. The model developer selects elements and attributes of the source message and drops them onto the observer message. This mechanism allows the model developer to define mappings between source and observer message formats by simply dragging and dropping between the left and right panels of the dialog. Based on the user's graphical mapping between components, GRADE automatically generates the underlying XSLT statements that will execute the data translation at run-time. Sample generated XSLT code is shown in the figure.



Property	Value
Data Type	xsd:string
Min Occurs	1
Max Occurs	1

☐ Edit XSL Manually

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <xsl:element name="input-data">
      <xsl:element name="input">
        <xsl:attribute name="name">
          <xsl:value-of select="/output-data/output/@name" />
        </xsl:attribute>
        <xsl:element name="PositionThreat">
          <xsl:element name="High">
            <xsl:value-of select="/output-data/output/PositionThreat/High" />
          </xsl:element>
          <xsl:element name="Low">
            <xsl:value-of select="/output-data/output/PositionThreat/Low" />
          </xsl:element>
          <xsl:element name="Medium">
            <xsl:value-of select="/output-data/output/PositionThreat/Medium" />
          </xsl:element>
        </xsl:element>
      </xsl:element>
```

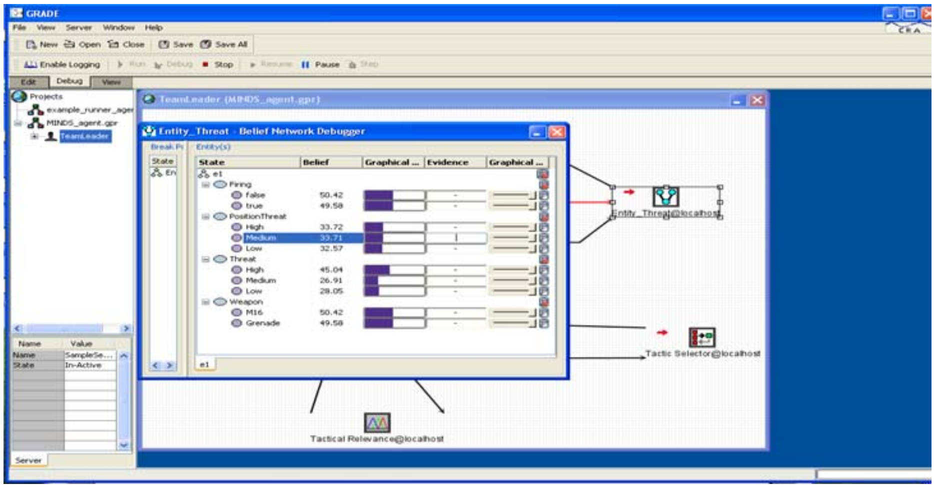
Figure 2-2: XSLT Generator Dialog

GRADE's capabilities for drag-and-drop model construction, easily accessible component configuration interfaces, and automated XSLT generation greatly reduce the time and cost associated with model development and integration.

2.2 Visual Model Verification in GRADE

A major cost typically associated with the development of complex models is the process of verifying that the model behaves as it is intended to, and diagnosing and troubleshooting behavior problems when they are discovered. We support model verification in GRADE by defining a "debug" interface for components that allows network-level (between components) as well as component-level (within components) debugging. This allows GRADE users to inspect the state of the model, including the internal state of any component while the model is running.

Before initiating debugging, the user may set breakpoints on any component in the agent. Once the model is executed in debug mode, the user can send input data into any component to initiate model processing. Once a breakpoint is reached, the model pauses execution and the debugger displays a flashing indicator for the component that triggered the breakpoint. Figure 2-3 shows an example model in debug mode. The center panel shows the current state of one of the belief network components in the model.


GRADE
File View Server Window Help
New Open Close Save Save All
Enable Logging Run Debug Stop Resume Pause Step
Edit Debug View
Projects
example_runner_ager
MINDS_agent.gpr
TeamLeader
TeamLeader (MINDS_agent.gpr)
Entity_Threat - Belief Network Debugger
Break Pt
Entity(s)
State
En
e1
Entity_Threat@localhost
Tactic Selector@localhost
Tactical Relevance@localhost
Server

State	Belief	Graphical ...	Evidence	Graphical ...
e1				
Firing				
false	50.42		-	
true	49.58		-	
PositionThreat				
High	33.72		-	
Medium	33.71			
Low	32.57		-	
Threat				
High	45.04		-	
Medium	26.91		-	
Low	28.05		-	
Weapon				
M16	50.42		-	
Grenade	49.58		-	

Name	Value
Name	SampleSe...
State	In-Active

Figure 2-3: Model Verification in the Debug Workspace

Breakpoints can be added and removed during debugging to allow the user to investigate the information processing, message passing and overall behavior of the model. We have found these debugging capabilities to be invaluable during model verification.

2.3 Post-execution Model Visualization in GRADE

The visualization environment within GRADE is intended to provide after-action review capabilities. This environment will allow the model developer to replay a model's behavior from a log file generated during previous model execution runs. This capability can be leveraged for PMESII modeling as a way of supporting model validation and causal analysis, as discussed in Section 5.

GRADE supports data collection for subsequent visualization through a distributed logging mechanism, depicted in Figure 2-4. Each component is initialized with a *Logger*, which points to a centralized logging object. While configuring the components at design time, the model developer selects elements within the component models for which state data should be logged. As components process data at runtime, they log the data selected by the user to the *Logger*. Their data is then sent to the central logging object, time-stamped, and stored. Each component may record user-specified events to a central log file. Event types may differ across the components: for instance, a belief network component is capable of recording evidence posted to a network, while the rules component may log when specific facts are asserted in the knowledge base. GRADE collects all the component log entries and produces a single log file that captures all of the user-specified information.

Once model state data has been logged, the user can "play back" the contents of the log and observe the change in component state at each step. Figure 2-5 shows the GRADE visualization environment being used to play back a log file generated during a simulation scenario by a simple example model. The playback control toolbar is shown at the top. A slider is provided to let the user easily access any point in the recorded log. As the playback controls change the time in the playback, either while the view is "playing" or because the user has selected a new time, GRADE reads the last log entry for each model component and link recorded before that time and passes the associated data to the components and links. The data is subsequently used to populate visualization interfaces for these objects.

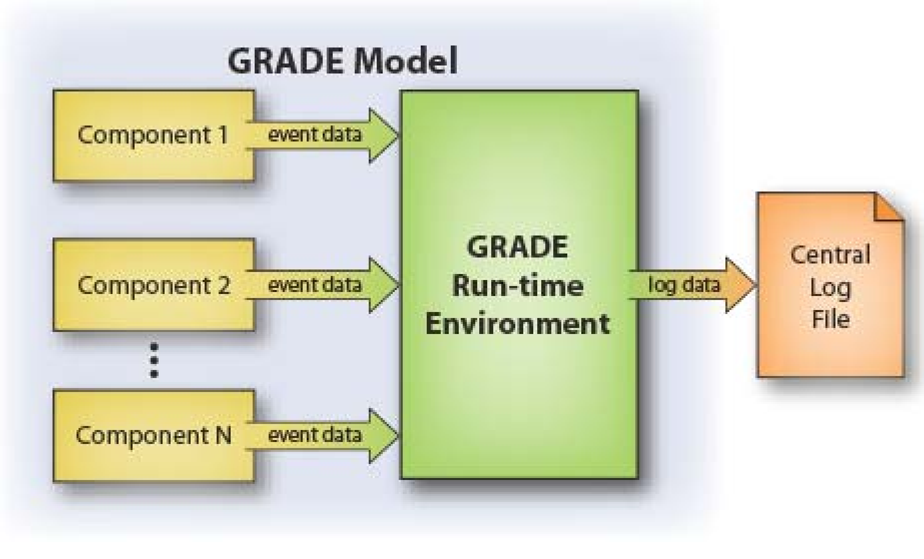
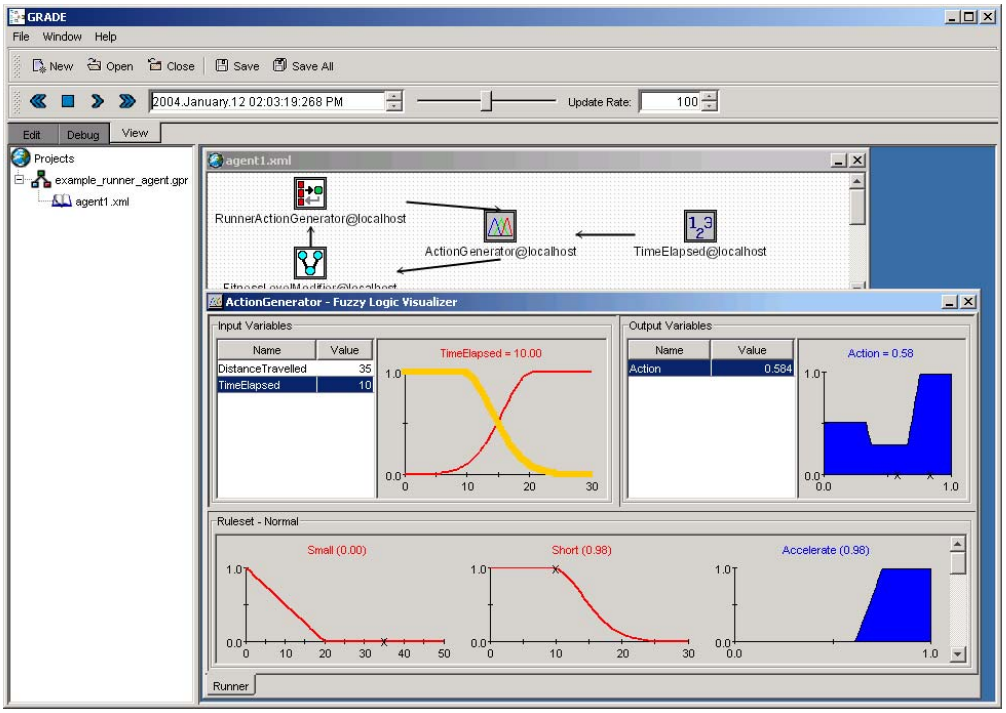


Figure 2-4: Logging Model State in GRADE

Each of our standard components now has a visualization tool developed specifically to enable after-action review. In the figure, the fuzzy logic visualization frame is shown at the bottom as an example interface. It displays the current input variable values (on the left), fuzzy rule firings (on the bottom), and output variable values (on the right). This allows the model developer to fully inspect the state of the fuzzy logic component at any point in the log.



2004.January.12 02:03:19:268 PM

Update Rate: 100

Name	Value
DistanceTravelled	35
TimeElapsed	10

Name	Value
Action	0.584

Figure 2-5: Model Visualization in the View Workspace

2.4 Component-Based Architecture

GRADE currently uses three core component technologies for modeling: fuzzy inferencing, belief networks, and rule-based expert systems. While the use of these three core components in combination has proven to be a very effective approach to meeting many modeling challenges, there remain cases where development of additional components is required. In such situations, new components can be rapidly adapted or prototyped using the GRADE component API.

To write a new component or adapt an existing product for integration within the GRADE framework, a developer encapsulates the targeted modeling algorithms and tools within a component implementing a set of four Java interfaces. The interfaces not only specify the component's runtime behavior, but also provide GRADE with the Editor, Debugger, and Visualizer user interfaces specific to that component. The four interfaces are as follows:

1. The *AgentComponent* interface specifies runtime behavior. *AgentComponent* specifies the methods used to initialize the component and to pass data to and from

the component. Upon receiving data, the component processes it and passes any results to other linked components.

2. The *AgentComponentEditor* provides the Java Swing GUI that allows the component to be configured by the model developer. The methods specified by the interface allow the component to provide a *JPanel*, a *JToolbar*, and a *JMenu* to GRADE that the modeler can use to configure the component.
3. The *AgentComponentDebugger* is used during model verification sessions. It provides a GUI that lets users examine and change component state as well as set breakpoints to control model execution for analysis of behavior.
4. The *AgentComponentVisualizer* is used to facilitate the after-action review capabilities provided by GRADE. Much like the *AgentComponentDebugger*, the visualizer provides a GUI that allows users to view the component's state. However, instead of populating the display with data from the component at runtime, GRADE passes it messages logged in a previous execution of the model and uses these to configure the visualization.

Because these interfaces can be implemented independent of one another, the component can be developed iteratively, adding progressively more functionality at each development step. Once the runtime and graphical editor interfaces are implemented, the developer can use the component in the GRADE framework for model construction, deferring development of the debugging and visualization interfaces until they are required.

3. Integration of Advanced PMESII Modeling Tools

Under this effort, we extended GRADE's component set to support the construction of PMESII models. PMESII models can be built using a wide variety of modeling tools, including GRADE's core tools of fuzzy logic, Bayesian belief networks, and expert systems, as well as other modeling paradigms, including causal graphs, concept graphs, concept maps, semantic networks, social networks, system dynamics models, neural networks, and situation theory. Because of the component-based nature of the GRADE toolkit, it is straightforward to incorporate new modeling technologies and tools into the GRADE infrastructure (as described in Section 2.4). By implementing a simple interface for each new technology, it becomes a new, integrated GRADE component. GRADE incorporates the tool's existing configuration interface and supports communication between model components built with the new technology and model components built with other technologies.

Under this effort we explored the applicability of a variety of candidate modeling techniques and chose one, the *Ptolemy systems dynamics modeling framework*, for incorporation into GRADE's component suite for PMESII model development. The motivation for this effort was both to demonstrate the ease of integration of third-party tools, and to support the modeling technology with the highest potential payoff for predictive PMESII modeling.

In Section 3.1, we present an overview of the range of other modeling technologies we considered for integration. In Section 3.2, we discuss our Ptolemy integration effort in detail.

3.1 Review of Modeling Technologies

The modeling techniques we considered for incorporation within GRADE are listed in Table 3-1. The table compares these modeling paradigms along with different features based on our experience using the listed technologies. We define these features below, and then briefly introduce each of the paradigms.

Table 3-1: PMESII Modeling Techniques and Their Features

Features: Modeling Techniques:	Expressivity	Executable	Reasoning	Adaptability	Tools	Exemplary Products
Concept Map	High	No	Forward Backward	Medium	Free (Research) COTS	CmapTools (cmap.ihmc.us) Decision Explorer (www.banxia.com/demain.html)
Concept Graph	Medium	No (Graphviz) Yes (OCCAM)	Forward Backward	Low	Free (Limited) In-House	Graphviz (graphviz.org) OCCAM (http://www.cra.com/contract-r-d/cognitive-systems-occam.asp)
Social Networks	Medium	Yes	Forward Backward	Medium	In-House	OCCAM (cra.com/contract-r-d/cognitive-systems-occam.asp)
Causal Graph	Medium	Yes	Forward Backward	Medium	In-House Free (Limited)	BNet (cra.com/bnet) C4.5 (http://www.rulequest.com/Personal/)
System Dynamics Model	Medium	Yes	Forward Backward	Low	Free (Research)	Ptolemy (http://ptolemy.berkeley.edu)

Neural Network	Low	Yes	Forward	High	COTS Free (Research)	NeuroSolutions (www.neurosolutions.com) Xerion (www.cs.toronto.edu/~xerion/)
Situation Theory	Medium	Yes	Forward	Low	In-House	PRISM (www.cra.com)

Expressivity of a modeling paradigm refers to its ability to capture and express an analyst's knowledge in terms of the constructs the paradigm offers. The expressivity of a concept graph is very high as it keeps the phrases used by the analysts intact in the model. In contrast, a neural network model is only able to keep the input-output relationships in the model. More expressive models are better able to capture the richness of PMESII domains and are typically easier to build, use, and understand by the modeler.

The e*xecutable* feature of a modeling technique refers to whether some useful information that is implicit in a model, e.g., degree of influence of one variable onto another, can be derived from the model via some kind of inferencing algorithm. A causal graph, for example, is an executable paradigm as it offers propagation algorithms, and so also is a trained neural network. In contrast, the concept-mapping model does not have such an algorithm. Non-executable modeling techniques are useful for visualizing complex models for human understanding and analysis; executable models are useful for providing automated analysis of the models.

Reasoning of a modeling paradigm refers to the paradigm's ability to detect the direction of influence (not just connection) of one variable to another. A belief network propagation algorithm, for example, incorporates both deductive and abductive reasoning, and thus is able to detect both forward and backward influences. On the other hand, the standard back propagation neural-network modeling paradigm is limited only to forward reasoning. Different modeling tasks require different kinds of reasoning. It is sometimes useful to be able to look at a state and reason about likely future outcomes (forward reasoning). For instance, one might want to attempt to predict the likelihood of social unrest by evaluating the current social, political, and economic state of affairs. Other times it is useful to look at externally available information and diagnose the likely underlying causes (backward reasoning). For instance, one might want to reason from observed social unrest back to the likely underlying political, economic, and social causes in order to properly address the causes of the unrest. For these reasons, it is important to support both forms of reasoning with the modeling tools we provide.

Adaptability of a modeling paradigm refers to automatic adjustments by models, which are necessary to take into account new observations. It is hard to adjust structures of graphical models as they are built in consultation with subject matter experts. But the strength of relationships among a set of variables within a model, e.g., probabilities in a belief network model or activation levels within a neural model, can be adjusted based on observations without changing their structure. Having models that can easily be adapted to represent new concepts and incorporate new data are generally preferable.

Tools of a modeling paradigm refers to the currently available software tools implementing the paradigm, that is, whether such a tool is in-house, COTS, GOTS, open source, or freely available for research/commercial purposes. The existence of such tools gives us confidence that we will be able to focus our effort on the integration of the various modeling techniques into our GRADE-based tool, and not be required to rebuild core technologies.

We now briefly describe the different modeling techniques that we considered.

Concept Maps

Concept maps are a result of research into human learning and knowledge construction (Novak, 1998). In concept maps, the primary elements of knowledge are concepts, and relationships between concepts are propositions. Concept maps are a graphical two-dimensional display of concepts, connected by directed arcs encoding brief relationships, e.g., linking phrases, between pairs of concepts forming propositions. Each concept node is labeled with a noun, adjective or short phrase, and each edge is labeled with verbs or verb phrases describing the relation between the connected concepts. Concepts maps are highly effective in quickly capturing domain knowledge along PMESII dimensions.

A popular tool for concept mapping is the CmapTools (Canas et al., 2004) package developed at the Institute for Human and Machine Cognition (IHMC) (www.ihmc.us). The package is freely available for both commercial and non-commercial use, and has many advantages over using sticky notes or a more general diagramming tool, e.g. it can record the entire mapmaking process. There are also COTS tools that can be used, such as Banxia’s Knowledge Explorer.

Concept Graphs

Concept graphs are a formal system of logic based on the existential graphs of C. S. Peirce and semantic networks. Concept graphs explicitly represent entities/concepts and relationships between entities as nodes in a directed graph. They are mathematically precise and computationally tractable structures, which have a graphic representation that is humanly readable. For this reason, concept graphs have been used in a variety of applications for computer linguistics, knowledge representation, information retrieval, and database design. Their ease of use and generality make them immediately useful for modeling a wade variety of domains, including PMESII domains.

Figure 3-1 is an example concept graph encoding a generic behavioral model of a terrorist leader.

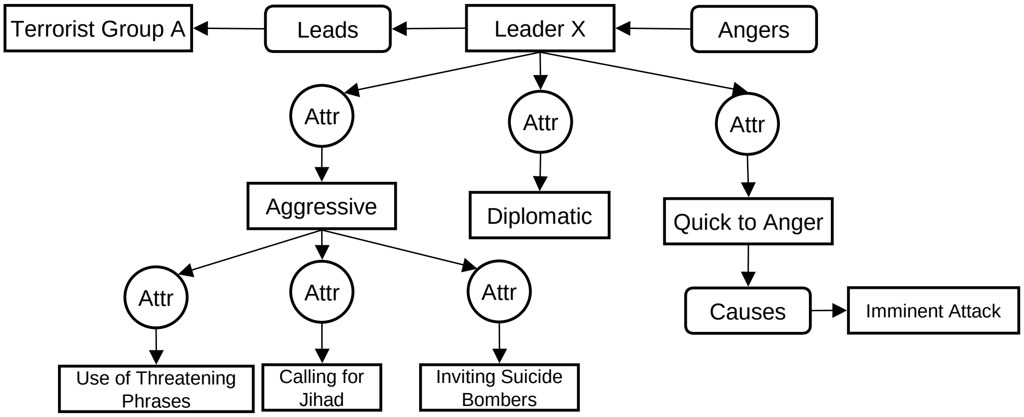


Figure 3-1: Concept graph model for terrorist leader behavior

While existing concept graph products exist, concept graph functionality is subsumed by our OCCAM network visualization and analysis tool, which also provides, for instance, social network analysis functionality. OCCAM also provides *executable* functionality for making automated inferences over concept graphs.

Social Networks

Social networks are similar to concept graphs, but they represent social structures. The nodes of the social network typically represent individuals and the links between them represent social relationships. Social network analysis (SNA) provides tools for reasoning about social networks, their strengths and weaknesses, the structural roles played by particular individuals, and their dynamics over time. Because of the focus on the analysis of social structures, SNA is directly applicable to a range of PMESII modeling tasks.

Our in-house OCCAM tool builds on traditional SNA functionality but provides additional representational and analytic power as well. In OCCAM, nodes can represent not only individuals, but also arbitrary entities, especially including groups. Links are similarly extended to represent not only individual-to-individual relationships, but also individual-to-group relationships (e.g., member-of) and group-to-group relationships, e.g., rival-political-party. OCCAM also provides built in Bayesian and rule-based reasoning capabilities to enable to automated analysis of the graph. So, for instance, a Bayesian network might represent that members of a group might have a high probability of holding views that are promoted by that group, where the group, the individual, and the ideology are all represented in OCCAM as nodes with appropriate links between them. In this case, the OCCAM tool will automatically create a new *believes* link between the individual and the ideology and annotate it with a particular probability.

Causal Graphs

A *causal graph*, e.g., a belief network, (Jensen, 1996) is a graphical, probabilistic knowledge representation of a collection of variables describing some domain. The strength of causal graphs are their ability to represent both the causal structure of a domain and the probabilistic elements of those causal relationships (X causes Y with some probability), thus enabling the modeling of both qualitative and quantitative details of the model. In addition, the ability of causal graphs to handle both forward (causal) reasoning and backward (diagnostic or abductive) reasoning, makes them especially well suited to domains with many sources of data, some of which are uncertain, unreliable, or potentially missing. Many PMESII modeling problems fall within such a scope.

Influence diagrams are a specialization of causal networks, augmented with decision variables and utility functions to solve decision problems. *Decision trees* are specialized influence diagrams that help to choose between options by projecting likely outcomes as utilities. Such extensions to causal graphs make it possible to also reason about the costs and benefits of possible decisions. This functionality can be used to both support intelligent decision making and to model likely decisions on the part of the entities being modeled.

Our COTS Bayesian reasoning construction tool and execution engine, Bnet:Builder®, facilitates construction and reasoning with causal graphs and is already integrated within the GRADE environment. There are also other existing COTS solutions to modeling influence diagrams and decision trees, such as C4.5 (http://www.rulequest.com/Personal/).

System Dynamics Models

System Dynamics Models, such as the *Stabilization and Reconstruction Operations Model* (SROM) (Robbins et al., 2005) analyze the organizational hierarchy, dependencies, interdependencies, exogenous drivers, strengths, and weaknesses of a country's PMESII systems to enable more efficient resource expenditure. SROM models PMESII systems at the national and regional levels, including the interactions between regions. They also take into account: demographic data, insurgent and coalition military, critical infrastructure, law enforcement, indigenous security institutions, and public opinion. As SROM models were designed to model PMESII systems, they are obviously directly relevant to the task as hand.

The SROM models developed by the AFRL/RI NO'EM group were built using the Ptolemy heterogeneous modeling software (http://ptolemy.berkeley.edu), which is developed and supported by the Electrical Engineering and Computer Science department of the University of California, Berkeley. While developed primarily for modeling of real-time embedded systems, its heterogeneous processing model makes it an effective tool for integrating a variety of data processing algorithms.

Because of its current and ongoing use in Air Force PMESII modeling efforts and the representative nature of the software engineering challenges involved in integration, we chose the integration of Ptolemy into GRADE as a primary focus of our toolkit development effort. This effort is described in detail in Section 3.2.

Neural Networks

A *neural network* is a nonlinear information-processing paradigm that models complex systems with a large number of highly interconnected processing elements (a.k.a. neurons or nodes), arranged in multiple layers, working in unison to solve specific problems. Neural networks offer some of the most versatile ways of mapping or classifying a nonlinear process or relationship. Neural networks have been successfully used in diverse paradigms, such as recognition of speakers in communications, diagnosis of hepatitis, recovery of telecommunications from faulty software, interpretation of multi-meaning Chinese words, undersea mine detection, texture analysis, three-dimensional object recognition, hand-written word recognition, and facial recognition. Neural networks would be useful in building PMESII models for those domains that have highly complex non-linear relationships between input and output variables.

Existing Neural Network construction kits and runtime engines exist, including the Xerion tool from the University of Toronto and the NeuroSolutions tools from NeuroSolutions.

Situation Theory

Situation theory models information processing and flow, i.e., how an agent extracts information from the world and how it is subsequently transferred between agents. Situation Theory provides a paradigm for describing the world, an ontology for representing it, and a suite of inferences for reasoning about it. Situation theory is unique in that it places situations alongside individuals, relations and locations as first-class members of its ontology. Situations provide partial descriptions of the world in terms of the features individuated by some agent. They are defined in terms of the relationships they support, i.e., they represent relationships between relationships. Situations provide a powerful representation of complex events spread over both space and time and therefore, serve as a natural representation of a variety of PMESII models. Situation theory has been applied to a variety of fields including natural language understanding (Barwise & Perry, 1983), information visualization (Lewis, 1990), cooperative social interaction (Devlin & Rosenberg, 1991), and both Level 2 (Steinberg & Bowman, 2004) and Level 3 (Steinberg, 2005) data fusion.

Charles River Analytics has developed the Platform for the Representation of and Inference over Situation-theoretic Models (PRISM), a Java-based framework for developing Situation theory-based models. Integration of Situation theory-based modeling capabilities provided by PRISM into the GRADE toolkit could be a valuable direction for future research and development efforts.

3.2 Ptolemy Integration

As described above, we selected integration of the Ptolemy systems dynamics modeling framework as the primary focus of our effort to incorporate new technologies for PMESII modeling into GRADE. This choice had the advantages of

- Immediate relevance to the CPE domain,
- Availability of existing models for use as examples,
- Java implementation with a non-restrictive license,
- Storage of models using the Modeling Markup Language (MoML), a well-defined XML file format with a published Document Type Definition (DTD), and
- An existing graphical editor (Vergil) for creating and maintaining Ptolemy models.

Integration of Ptolemy provided a representative example of the types of software engineering steps that would be required to integrate any third-party tool with a pre-existing configuration interface (such as Vergil) into GRADE. Our integration of Ptolemy followed the path outlined in Section 2.4: implementation of runtime functionality first, followed by the creation of configuration and debugging interfaces. The results of each of these steps in our development effort are discussed further below.

3.2.1 Ptolemy Runtime Component Integration

Model Input and Output Definition

The first step in our Ptolemy integration effort was the implementation of runtime functionality—the ability to execute Ptolemy models within the GRADE environment, and enable the Ptolemy component to exchange data with other GRADE components.

To allow a GRADE Ptolemy Component to exchange data with other components, we first had to define a way to specify the input and output formats (schemas) for any given Ptolemy model. A Ptolemy model can contain any number of *actors* (individual processing elements), which can themselves be composed hierarchically of other actors. It is unreasonable to expose every actor in the Ptolemy model as a potential input or output variable because Ptolemy actors are often low level numerical operators, e.g., addition, absolute value, etc., of which any Ptolemy model of reasonable complexity could have a vast number. Exposing such a large number of variables by including them as elements in the input and output schemas would create unnecessary complexity and difficulty for the model developer.

For this reason, we established a convention for the Ptolemy component requiring that the model developer identify the salient inputs and outputs for the model by specifying them as top-level *Parameter* objects inside the Ptolemy model. One of the benefits of this design is that *any* value in the Ptolemy model, whether a final output variable or an intermediate value generated during the course of model execution, can be linked to a Parameter and therefore be exposed to other components in the GRADE modeling framework for processing.

Once the top-level Parameters have been created inside the Ptolemy model, additional characteristics of the parameters must still be defined. By selecting the "Define I/O" button in the GRADE Component editor interface (discussed in further detail in Section 3.2.2 below); the PMESII model developer specifies the following characteristics of the parameter:

- *Direction.* Indicates whether the parameter is an input or an output parameter. Parameters designated as input parameters are included as elements in the Ptolemy component’s input schema, whereas output parameters are included in the component’s output schema.

- *Type.* One of: Boolean, string, double, or integer.
- *Default.* An optional value that provides a default value for the parameter if none is produced by the model.

Once input and output parameters have been defined, the model developer can use GRADE’s XSL wizard to link the parameters in the Ptolemy model to the inputs and outputs of other components.

Model Execution Control

Ptolemy supports multiple models of computation using its *Directors.* Example directors include CT (continuous time), FSM (finite state machine), DE (discrete event), and SDF (synchronous dataflow). Without a Director, a Ptolemy model is simply a block diagram of components; the Director controls how information is exchanged and synchronized between blocks.

One of the initial issues we had when attempting to integrate Ptolemy models into GRADE was that when using certain Directors, Ptolemy would not yield control of model execution to GRADE. We first encountered this problem when working with the NO’EM group’s recruiting, training and Active-duty (RTA) model, which used the Discrete Event director. When another GRADE component would pass the Ptolemy Component encapsulating the RTA model an input message representing state information about a particular moment in time, the model, rather than processing the input for the given time-tick and producing the output for that time-tick, would instead run to completion and output only the final results of its computation to GRADE.

We found that by switching from the Discrete Event director to the Synchronous Dataflow (SDF) director we were able to enable the Ptolemy component to run “iteratively,” running one execution for each message received, and providing the results of its computation as output at each step. The requirement that Ptolemy models used in GRADE use the SDF director is not an unreasonable limitation for the purpose of PMESII model development as this director is the one that is most useful and typical for discrete-event based systems dynamics modeling. Upon conferring with AFRL we learned that the NO’EM group had reached the same conclusion in their work with Ptolemy and had independently chosen to switch to the SDF director.

Summary of Steps to Run a Ptolemy Model in GRADE

Integration of any particular Ptolemy model into grade involves the following steps:

1. Create a new Ptolemy component within GRADE.
2. Using the Ptolemy component editor interface, import the .MoML file for the Ptolemy model into the Ptolemy component.
3. Create top-level Ptolemy Parameters to identify values or variables inside the model to expose to other GRADE components.
4. Using the "Define I/O" interface of the Ptolemy Editor, identify these parameters as being either input or output parameters, and specify their type.
5. Link the inputs and outputs of the Ptolemy component to the inputs and outputs of other components using GRADE's XSL Wizard.

The Ptolemy component editor, the configuration interface used to perform several of these steps, is discussed further in the following section.

3.2.2 Ptolemy Editor Integration

Once a component can be executed at runtime within GRADE and exchange data with other GRADE components, the next step in the component development path is to provide a graphical user interface allowing the GRADE model developer to configure it. There are generally three approaches to doing this, depending on the nature of the software being integrated:

1. *Develop a GUI from scratch using Java Swing.* This is most commonly done when the component itself is being developed from scratch and no third-party tools or software is being integrated.
2. *Provide a file chooser.* In cases where third-party tools are being integrated, but the third-party software does not provide an open-source platform for model configuration, a GRADE component developer can provide a file chooser allowing the model file for the third-party software to be selected. A typical example of this is CRA's MATLAB component, which simply allows the GRADE developer to select the .M file to load inside the component; any configuration steps required to create the .M file must be done within MATLAB itself.
3. *Encapsulate an available third-party configuration GUI* inside GRADE as the component editor. This approach can be taken when an open-source GUI for model configuration is available, as with Ptolemy's Vergil. While this provides an easy way to provide the GRADE model developer with a full-fledged configuration GUI inside

the GRADE IDE, there are some implementation challenges that can arise when taking this approach, which we address in this section.

Because the Ptolemy model development platform includes Vergil, an open-source, graphical application for creation and execution of Ptolemy models, we chose the third approach for the Ptolemy component, that of encapsulating Vergil inside the Ptolemy component editor.

Figure 3-2 shows a screenshot of an example Ptolemy component editor instance. The editor provides access to the full list of actors, directors, and other objects provided by Ptolemy for model creation and configuration. The Ptolemy component editor can be used to create new Ptolemy models from scratch within GRADE, or to edit and configure existing models that have been developed separately using Vergil. In addition to the capabilities provided by Vergil, the Ptolemy component editor also provides additional features to facilitate integration of Ptolemy models within GRADE, including the *Define I/O* button and a series of model import and export buttons which are provided in the toolbar for the Ptolemy Component Editor.

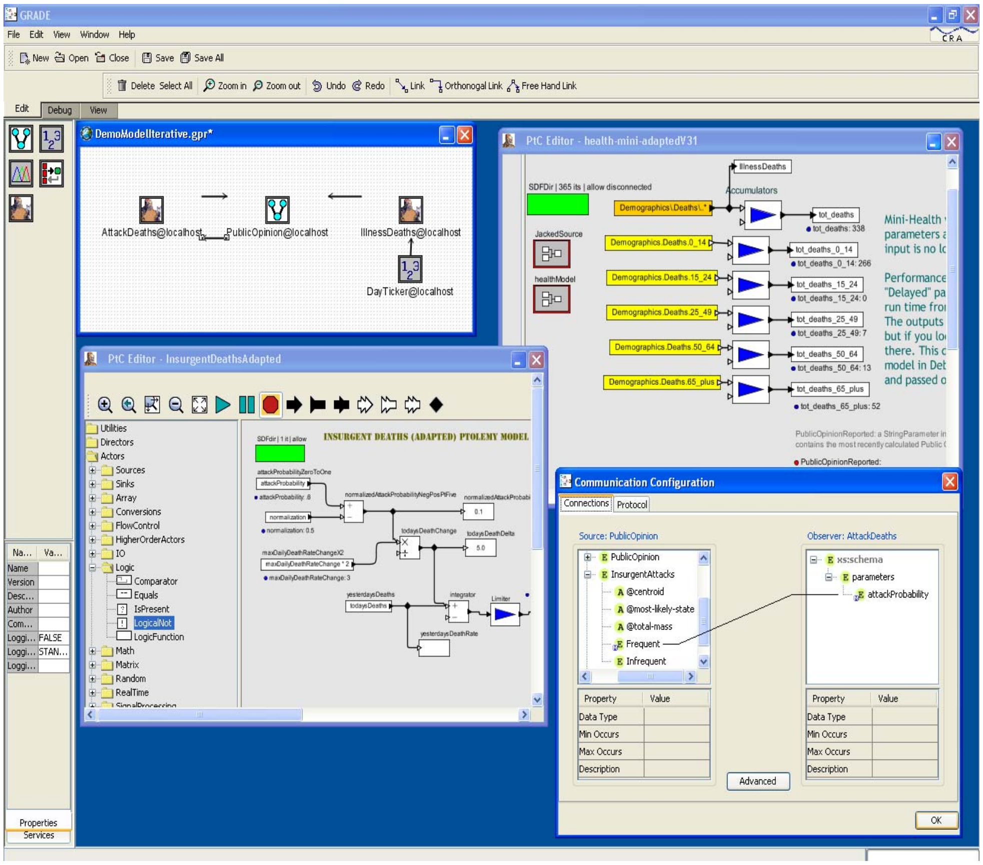


Figure 3-2: Editors for Two Ptolemy Components in the GRADE Edit Workspace

In general the Ptolemy component editor serves as an effective tool for creation and configuration of Ptolemy models within the GRADE modeling IDE. However, as a result of technical constraints imposed by the Vergil software, there are some minor outstanding issues associated with the editor integration. These stem from the fact that the Vergil GUI elements are coded in such a way as to assume that they are running inside a PtolemyFrame as their top-level container. When running inside the GRADE Ptolemy component, however, Vergil's configuration panel's top-level container is *GRADE*, not Vergil. This results in two known problematic behaviors:

Limited non-functional elements. When the user right-clicks an actor in the model configuration window, a context menu is opened which provides access to certain features. When the Customize→Ports element is selected within this context menu, the embedded Vergil code iterates up the Swing container stack, casts the top-level container to a PtolemyFrame, and attempts use the reference to the PtolemyFrame object to open up the appropriate dialog box. Because the top-level frame in this case is GRADE's JFrame, which is not a PtolemyFrame, the cast fails and no dialog box is opened. This menu item is shown in the context of Ptolemy model configuration in Figure 3-3 below.

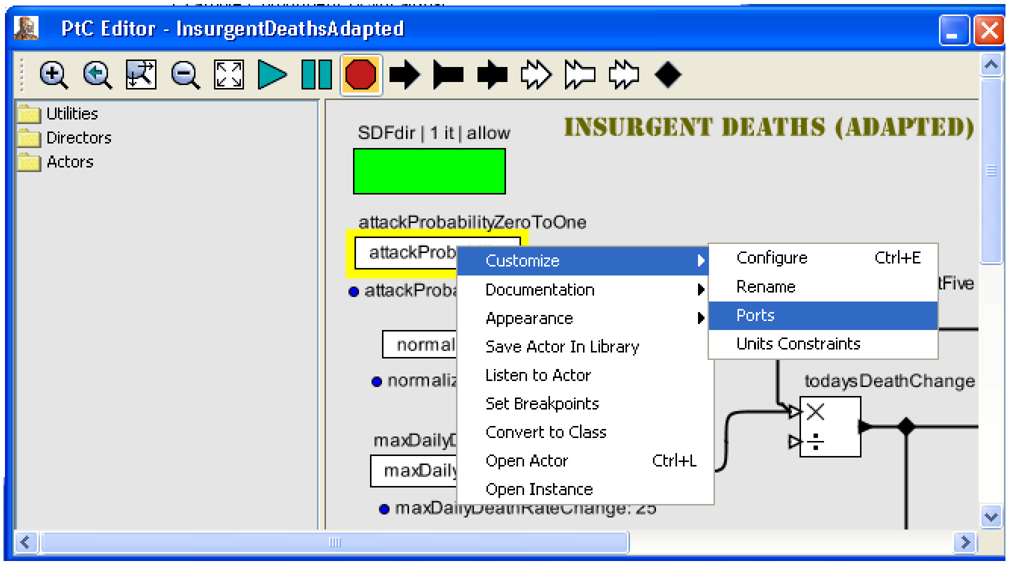


Figure 3-3: Selecting the Customize→Ports Context Menu Item in the Ptolemy Component Editor

Orphaned Vergil Application Windows. Because Ptolemy models are hierarchical, Vergil allows the model developer to "drill down" into composite actors by selecting the "Open Actor" context menu item, which opens up a configuration window for that specific actor. However, because it uses a non-standard approach to window management, Vergil does not open an internal window; rather, it opens up a top-level Vergil application window. The implication of this for GRADE is that when the user opens an actor within the Ptolemy component editor, it opens up a new Vergil application window, which contains Vergil's top-level menus rather than GRADE's, and which does not close when the Ptolemy component editor is closed, but rather remains open until GRADE is closed or the user closes the window manually. While this behavior does not in itself limit the model developer's ability to create or configure models, it is an unexpected side-effect of the integration process which could lead to unexpected behavior for a model developer who was not aware of it.

Since Ptolemy is open-source software, one potential avenue for addressing these issues would have been to modify the Vergil source code to make it more suitable for encapsulation within GRADE. However, doing so would have precluded the end user of the Ptolemy component, in this case AFRL, from upgrading to future versions of Ptolemy and Vergil, as any modifications would be lost in the upgrade process. We therefore felt that integrating the software as-is and documentation the unusual behavior would better serve the users of the software.

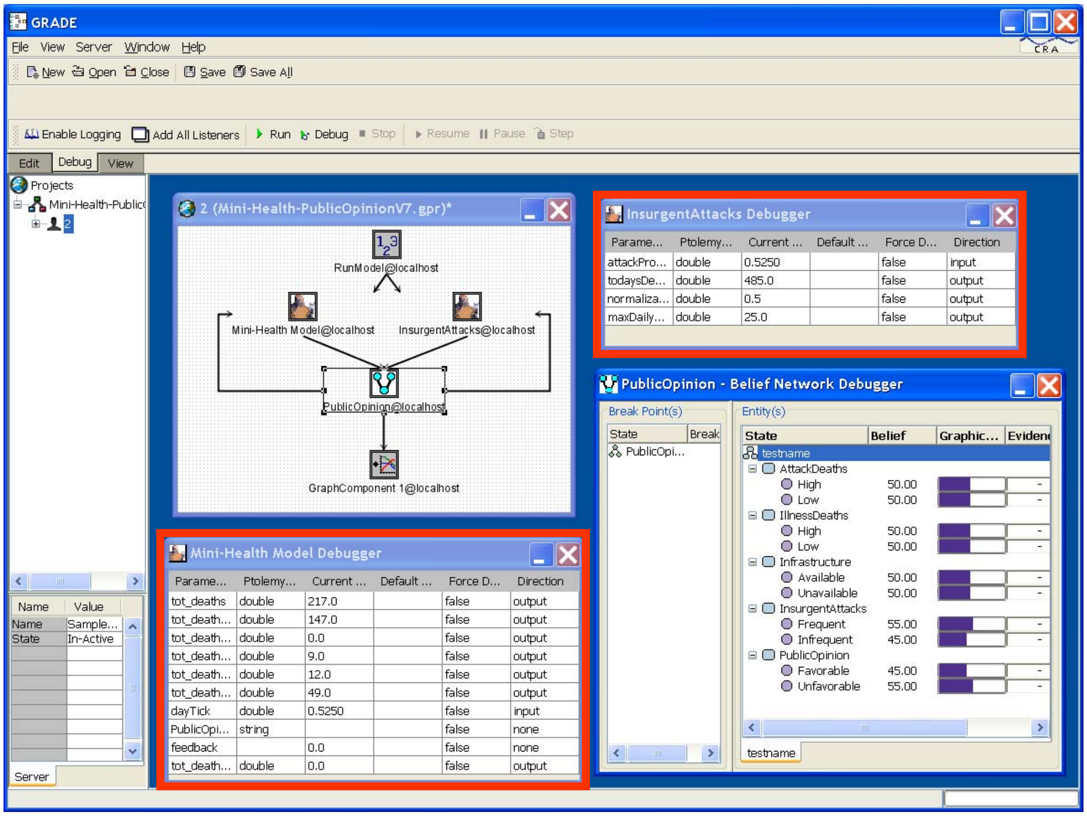
Despite these issues, the Ptolemy component editor remains a fully-featured interface for creating and configuring Ptolemy models within GRADE. The problems we encountered are representative of the types of issues that can arise when attempting to encapsulate GUI elements from one application into another application with a different architecture and design.

3.2.3 Ptolemy Component Debug and View Integration

The final step in our Ptolemy integration effort was to develop Debug and View interfaces for the Ptolemy component. The Ptolemy component debugger allows the model developer to monitor the component's state during execution of the model, while the Ptolemy component visualizer allows the model developer to observe changes in the component's state from a logged run of the model. Though the source of the data is different, both the debugger and the visualizer display component state information to the user, and therefore the same interface can be used for both components.

For both the Ptolemy component debugger and visualizer we created a simple table-based display format that is modeled after the "Define I/O" configuration window used by the Ptolemy component editor. For each top-level parameter in the model, the interface specifies the current value of that parameter, which is derived from real-time data in the case of the debugger, and from logged data in the case of the visualizer. The table also displays basic information about each parameter, such as its name, its type, and its direction (input or output). A screenshot showing example instances of the Ptolemy component debugger open in the GRADE Debug workspace is shown in Figure 3-4, with the Ptolemy debugger windows highlighted in red.

The Ptolemy component debugger and visualizer provide the ability for the model developer to verify and validate the execution of Ptolemy sub-models in the context of the GRADE PMESII IDE framework, and complete the software development path for Ptolemy integration into GRADE.


GRADE
File View Server Window Help
New Open Close Save Save All
Enable Logging Add All Listeners Run Debug Stop Resume Pause Step
Edit Debug View
Projects
Mini-Health-Public
2
2 (Mini-Health-PublicOpinionV7.gpr)*
RunModel@localhost
Mini-Health Model@localhost
InsurgentAttacks@localhost
PublicOpinion@localhost
GraphComponent 1@localhost
Server
CRA

Name	Value
Name	Sample...
State	In-Active

InsurgentAttacks Debugger

Parame...	Ptolemy...	Current ...	Default ...	Force D...	Direction
attackPro...	double	0.5250		false	input
todaysDe...	double	485.0		false	output
normaliza...	double	0.5		false	output
maxDaily...	double	25.0		false	output

Mini-Health Model Debugger

Parame...	Ptolemy...	Current ...	Default ...	Force D...	Direction
tot_deaths	double	217.0		false	output
tot_death...	double	147.0		false	output
tot_death...	double	0.0		false	output
tot_death...	double	9.0		false	output
tot_death...	double	12.0		false	output
tot_death...	double	49.0		false	output
dayTick	double	0.5250		false	input
PublicOpi...	string			false	none
feedback		0.0		false	none
tot_death...	double	0.0		false	output

PublicOpinion - Belief Network Debugger

Break Point(s)

State	Break
PublicOpi...	

Entity(s)

State	Belief	Graphic...	Eviden
testname			
AttackDeaths			
High	50.00		-
Low	50.00		-
IllnessDeaths			
High	50.00		-
Low	50.00		-
Infrastructure			
Available	50.00		-
Unavailable	50.00		-
InsurgentAttacks			
Frequent	55.00		-
Infrequent	45.00		-
PublicOpinion			
Favorable	45.00		-
Unfavorable	55.00		-

testname

Figure 3-4: Ptolemy Component Debugger Windows (Highlighted in Red)

4. Support for Interoperable PMESII Modeling

While GRADE provides a base level of support for interoperable PMESII modeling by virtue of the fact that it provides a common environment for model development and a common protocol (XML) in which they can communicate, there are a number of challenges associated with resolving interoperability conflicts between PMESII models beyond the data-level. During this effort the primary research into advanced strategies for resolving model interoperability conflicts was conducted under our sister CPE program, "Framework for Building and Reasoning with Adaptive and Interoperable PMESII Models." In this section we provide a summary of the issues associated with interoperable PMESII modeling and the approach taken under our sister CPE program.

There are several fundamental issues (and associated "hard" problems) that need to be addressed in supporting interoperability between PMESII sub-models in an IDE for model development. The fundamental problem is one of incompatibility between one model's information products and another's information requirements. There are several types of model incompatibilities that must be managed, however, as shown in Figure 4-1 below.

- The first problem shown in the figure concerns *interface incompatibility* between two PMESII models that either already exist or are being developed independently. If we intend to feed output from model A about a certain object X as input to model B, then mismatch between the output and input may occur in terms of the assumptions about the numbers and types of X's attributes. This is often straight-forward but tedious to deal with, often merely involving "translation" from one descriptive framework to another, e.g., from numerical values – 1, 2, 3, -- to "fuzzy" values (low, medium, high,…). A bigger problem ensues when different levels of resolution are used to represent the same object in two different models, where low- and high-resolution objects must be merged between models.
- The second problem illustrated in the figure is about *ontological incompatibility* between models that arises due to differing vocabularies and expressive powers in their respective ontologies. Different teams of engineers and subject matter experts with a diverse range of expertise, knowledge, and cognitive capabilities independently creating PMESII models will inevitably develop and use different underlying ontologies, which, in turn, will give rise to incompatibilities across models.
- While ontological incompatibility creates problems due to multiple ways of designating an entity, the *formalism incompatibility* shown in the figure is concerned with multiple ways of instantiating the object entity computationally represented in the model. For example, uncertainty can be expressed not only in terms of probability values, but also via various other formalisms such as certainty factors, the Dempster-Shafer measure of beliefs, and numerous other qualitative dictionaries.

These are incompatible with each other, both in terms of their underlying conceptual representation of uncertainty and probabilistic reasoning, as well as in the sense of having different types of scales.

- Finally, if one wants to feed the output from a model in one PMESII domain to another, it will require an analyst or domain expert with knowledge of both domains to bridge the *subdomain gaps*. This is due not only to the ontological gaps between the domains being considered, but also to differing dynamics between the domains.

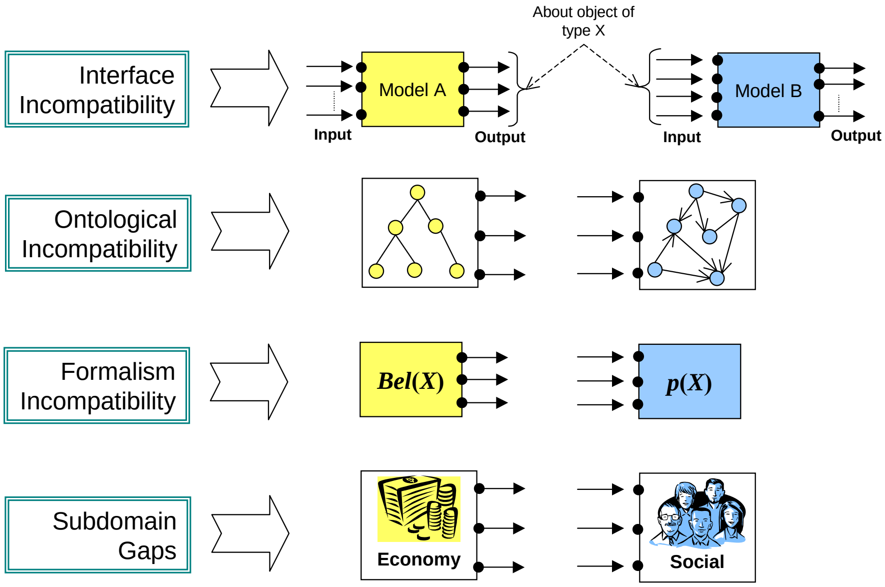


Figure 4-1: Gaps and incompatibilities between models

The research effort under our companion CPE program, "Framework for Building and Reasoning with Adaptive and Interoperable PMESII Models", focused on the development of methodologies and formalisms to address these issues through the application of AI techniques such as ontology learning, automated reverse engineering, general function inversion, semantic analysis, and advanced interpolation techniques. The objective is to minimize the level of effort on the part of the model developer to define detailed mappings and translation functions between component PMESII models through advanced AI tools.

There are some practical ways in which the results of the companion program could be integrated into the PMESII development toolkit. Since the goal of the research into interoperability in practical terms is to reduce the burden on the model developer (rather than to automate the process of model integration), there are ways in which we could augment the toolkit to suggest linkages between sub-models to the model developer that would be based on an underlying reasoning process. A straightforward way to implement this would be in

GRADE's XSL wizard tool for linking component inputs and outputs. As previously discussed in Section 2, GRADE provides a model integration capability through the XSLT Generator that allows for XML messages generated by one model component to be translated into XML messages that are consumable by another component. Using the ontology matching concepts developed under our sister CPE program, the XSL wizard could recommend potential matches between the outputs of one model and the inputs of another model at linking time. The data mapping and translation functions remain primarily driven by the model developer, but would be supported using underlying model-matching algorithms. We believe that this provides a rapidly accessible solution to the model integration problem that could be greatly enhanced in the course of future work.

5. Support for PMESII Model Verification and Validation

While the processes of validation and verification are often grouped together, they present very different challenges to the model developer. Verification determines whether the model behaves as it is intended to behave; validation determines whether the model behaves in accordance with the real world phenomenon it is intended to model. For the purposes of developing a modeling framework, we can consider verification as a testing and debugging issue, and validation as a data collection and analysis issue. In this section we describe our enhancements under this effort to GRADE to provide additional support for these capabilities.

5.1 Verification

GRADE incorporates a number of features to facilitate model verification, including the ability to halt and resume execution and view model state at runtime. GRADE's testing and debugging capabilities have already been described in Section 2.2 above.

In addition to verification at the model level within GRADE, we added the capability to perform low-level verification at the component level, a development strategy commonly used in software development called *unit testing.* Unit testing describes a technique that requires a model developer to create a dedicated and corresponding test for each and every feature they create. As models evolve from prototypes through beta testing to real-world use, the underlying unit tests can provide reassurance that fundamental and core pieces of a model are behaving as expected, even if the model as a whole is incomplete or undergoing modification. The use of unit tests to ensure correct behavior during the course of model changes is known as *regression testing.*

To allow model developers to employ the unit testing methodology to verify PMESII models and sub-models, we used the XmlUnit[1] open source project to expand the existing GRADE unit testing capabilities to enable verification of model inputs and outputs for regression testing. GRADE's existing unit testing capabilities relied on native Java API calls within the GRADE code, and were limited to features and functionality that were embedded within the GRADE modeling environment. The inclusion of XmlUnit enables a similar testing structure that can operate on the inter-model and intra-model messages, as opposed to the native Java API calls.

[1] http://xmlunit.sourceforge.net/

As a result, this capability is now available to non-programmers, such as analysts and model developers, to define valid model inputs and outputs for regression testing.

5.2 Validation

A framework enabling the integration of PMESII models must also provide support for the validation of the behavior of the individual and aggregated models. Some relevant lessons can be learned from AFRL's Agent-based Modeling and Behavior Representation (AMBR) program, which focused on the development of model comparison and validation strategies in the context of human behavior modeling (Gluck & Pew, 2001). One of AMBR's clear conclusions was that the primary requirement for model validation is that the model being evaluated *must run in the same problem space* as the empirical data collection process, and furthermore, must enable the collection and analysis of the same behavioral features between the empirical study and the modeled behavior. This guarantees that the data captured from the model and the data captured from the world are of equivalent types, and can provide a useful basis for evaluation. Therefore, what is required of a framework for validation are the flexibility and tools to collect the appropriate data from the model. If the framework is adequately equipped, the user can capture data from the model and compare it with the data gathered empirically.

Validation of the model's behavior may take a number of different forms, from an intuitive assessment of realistic behavior by a subject-matter expert to a statistical comparison of the model's output with real-world data. However, all forms of validation require that a model framework have the ability to: 1) collect output/model state data, 2) visualize model-generated data for examination and evaluation, and 3) export collected data in appropriate formats for statistical analysis in external software.

Our IDE supports the first requirement (data collection) through its logging API, which allows the user to specify the variables in the model that he/she wishes to log for analysis and evaluation. GRADE's logging API was described in detail in Section 2.3 above. GRADE supports the second requirement, that of visualization, through its visualization workspace and *AgentComponentVisualizer* interface, which allow component state to be examined from historical data. Support for visualization and evaluation for custom component technologies depends on the component developer to create an appropriate GUI to be returned by the *AgentComponentVisualizer* interface (such as the visualization GUI we implemented for the Ptolemy Component, described in Section 3.2.3). Moreover, the Graphing Component created under this effort provides an additional tool for examining and evaluating model-generated data.

The third capability, that of data export and interoperability with statistical analysis packages, is currently only partially supported. Model-generated data is logged and stored in an XML format, which can be converted into formats enabling analysis with standard statistical analysis packages such as MATLAB (http://www.mathworks.com) or SPSS (http://www.spss.com). However, this process must currently be performed manually through the use of XSLT or other XML processing tools. The addition of native conversion tools to the PMESII IDE itself is a potential direction for future work.

6. Support for PMESII Model Analysis

As the key requirement of the Commander's Predictive Environment is to enable the commander to anticipate possible futures, any framework for the integration of heterogeneous PMESII models must provide support for such a capability. Under this effort, we have focused our efforts in three areas to provide an analytical capability for the Commander's Predictive Environment. The first of these was our prototype of an example "Commander's Interface," which was intended to abstract out the technical details of model implementation and allow a commander to interact with the PMESII models in such a way as to support active decision-making. During the course of the program, however, it became clear that a more appropriate end user of such a tool would be the model analyst rather than the commander, so we clarified our use case accordingly and re-labeled the tool as the "Analyst's Interface." A detailed description of the tool is included in Section 6.1 below. Our work with the Analyst's Interface led us to conclude that its most valuable feature was its ability to interactively graph model outputs, so to make this functionality directly accessible to the model developer within the PMESII IDE itself we developed the Graphing Component, described in Section 6.2. Finally, we implemented a number of advanced analytical capabilities under a related program, HASMAT, which we describe in Section 6.3.

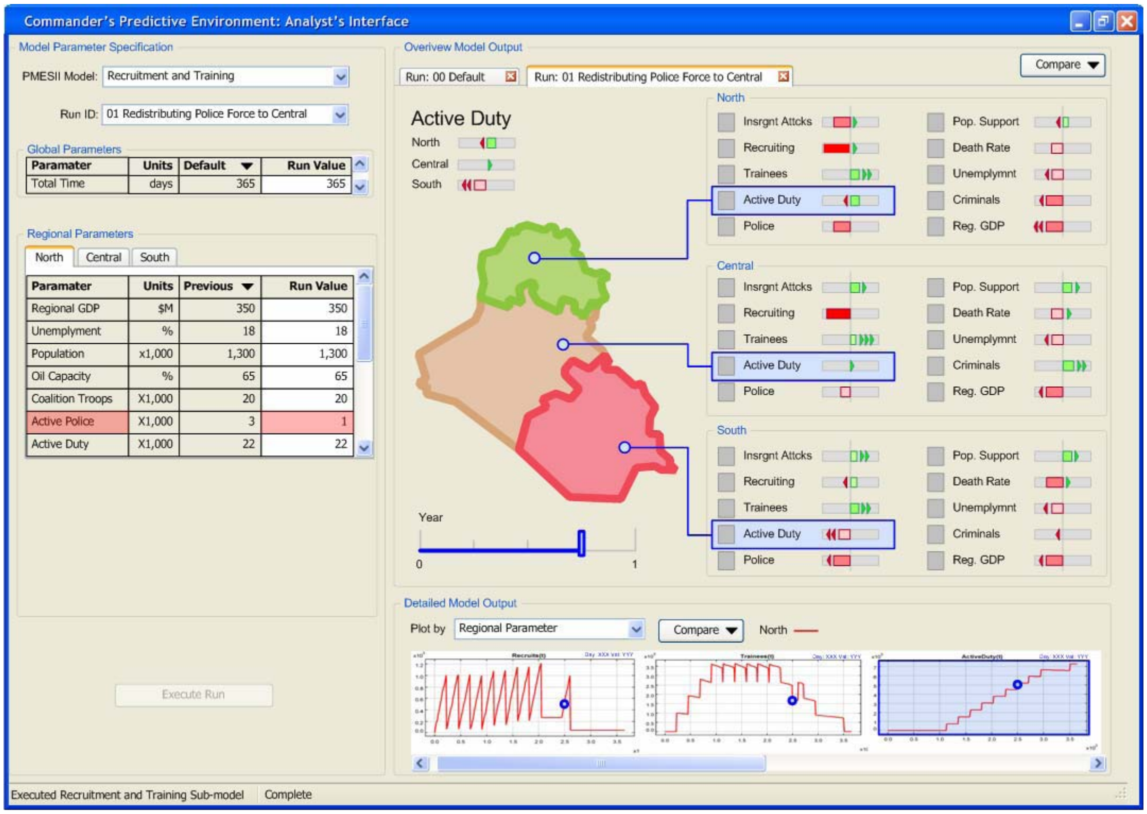
6.1 Analyst's Interface

The motivation behind our prototype of the Analyst's Interface was to provide an aggregated, accessible view of model results that could be used to support decision-making by a commander or other decision-maker. Such a tool would allow the commander or a member of his staff to easily generate model inputs (representing DIME actions that could be taken), and monitor model responses over time in a presentation framework that would be more intuitive than the PMESII IDE itself. For example, in the context of the recruiting and training SROM model, the commander might specify a specific allocation of troops to the recruiting and training of specific capabilities within the modeled region. The models would then be executed against this input set, and the commander could monitor the overall effects on high-level PMESII variables, e.g., unemployment, economic stability, crime rates, etc., in an intuitive graphical interface. This would isolate the commander from the detailed outputs and implementation details that would be of interest to the model developer, while providing the commander with intuitive and targeted real-time decision support leveraging the models constructed using the GRADE tool set.

As indicated above, while we initially envisioned these tools as directly supporting the commander's decision-making, during the course of the effort it became clear through conversations with AFRL that the more appropriate end user of the output to which we are providing access through these graphical tools is the Model Analyst. This decision does not compromise the added value to the overall GRADE-enabled PMESII modeling environment, but rather, simply isolates a more intuitive use case of the resulting tools.

In Figure 6.1 below, we provide an overview design vision for the PMESII model analysis tools that we designed. This set of model-centered tools supports the analyst's interactions with the underlying GRADE-based models defining the dynamics of the systems of interest, e.g., nation state models of economics, insurgency activity, etc. In the upper left of the graphic, we provide a simple selection tool for the analyst to select from among the range of available GRADE models defining the PMESII environment for execution and analysis. The selection of a specific model will result in the input fields for that model being captured from the selected GRADE model (via its XML Schema-based input/output specification) and populating the tabular input sets shown below on the left. Here, the analyst can configure the input parameters to the model to examine model responses to some set of "controllable" DIME actions, e.g., US troop allocations to support recruiting and training of indigenous security forces. The model inputs are defined as either "global" parameters or "regional" parameters. This is motivated by our program's ongoing relationship with the national and regional modeling paradigm of the SROM/NO'EM efforts at AFRL/RI. In future work, we would revisit this structure to define more abstract and generalized input categories for the analyst to interact with the underlying models. Once model inputs are specified, the analyst "executes" the model with the given input set via the mechanism in the lower left.

On the right side of the figure, we display the outputs of the model's execution. Also due to our program's interactions with the NO'EM modeling methods, we present results in a national and regional structure. This would also be revisited under future work to generalize the interface to other potential representative structures. We envision providing a map-based overview of the result of the model execution. Model output parameters (defined by the underlying GRADE model's output XML schema) are captured and populate the callout regional data sets on the right of the map. Because the complexity of the underlying model cannot be predicted, and the resulting output parameter set may be significantly larger than the representative set provided here, we envision that the analyst will be able to dynamically select the output parameters of greatest interest in a given study for display in these data sets. The data shown in the callout regional data sets show instantaneous values of specific parameters through the size of the green (above nominal) and red (below nominal) value bars shown, as well as the first derivative rate of change of those parameters shown through the decorator arrows that indicate rate of positive or negative change in value, i.e., more arrow decorators imply higher rate of change.



Commander's Predictive Environment: Analyst's Interface

Model Parameter Specification

PMESII Model: Recruitment and Training

Run ID: 01 Redistributing Police Force to Central

Global Parameters

Paramater	Units	Default	Run Value
Total Time	days	365	365

Regional Parameters

North | Central | South

Paramater	Units	Previous	Run Value
Regional GDP	$M	350	350
Unemplyment	%	18	18
Population	x1,000	1,300	1,300
Oil Capacity	%	65	65
Coalition Troops	X1,000	20	20
Active Police	X1,000	3	1
Active Duty	X1,000	22	22

Executed Recruitment and Training Sub-model | Complete

Figure 6-1: An Overview of the Analyst's Interface

As the user selects specific outputs in the callout data sets, the overview map shows the comparison of that value set across the modeled regions through fill color. In the figure above, we show the number of indigenous active duty security personnel operating within each region. Beneath the map view, we provide a modeled timeline that the analyst can manipulate to examine the trends of model output parameters over simulated time. As the user drags the timeline back and forth, the output data sets will display the values as described for that point in time.

Finally, in the lower right, we also provide the analyst with graphical displays of selected model outputs in time-series data plots, which allows for easier access to trend data throughout a model run for more detailed analysis. As the user manipulates the timeline (as described above), these data plots shift a marker to identify the value at the selected time.

During this effort we also created an initial implementation of the analyst's interface in software, and have demonstrated the integration of an example GRADE-defined PMESII model to populate the interface.

6.2 Graphing Component

As described above, the Analyst's Interface provides time-series data plots to visualize the results of a given model execution. We found this to be an essential tool both for verifying that example models were behaving as expected during model development, and for studying the dynamics of models during analysis. Because of its demonstrated utility for both model development and analysis, we focused a final part of our development effort on adding a graphing capability directly to GRADE. In addition to its utility for PMESII model development, the Graphing Component serves as a useful case study to illustrating the considerations involved in adding new analytical capabilities to GRADE.

We had two principal design options for integrating a graphing capability into the IDE. They were to 1) build a graphing capability into the IDE itself as an additional feature of the Debug workspace, or 2) create a new component for graphing that would receive model output and display it in its Debug GUI. The advantages and disadvantages of each of these approaches are summarized in Table 6-1.

Table 6-1: Advantages and Disadvatages of Graphing Feature Design Choices

	Advantages	Disadvantages
Built-In IDE Capability	• Graphing capability is automatically incorporated into every component; the developer does not have to configure a connection between each component he would like to graph • Graphing tool can be incorporated into the existing Debug workspace GUI	• Requires an additional interface to identify which elements in the input or output schema should be graphed, and which should be considered as the X or Y axis • Graphing capability is very tightly coupled to Debug workspace; it will be relatively more difficult to change graphing approaches or packages • The graphing capability would only be able to display input from a single component; it would not be able to graph data received from multiple components
Component-Based Approach	• Leverages the existing XSL wizard interface for mapping data from other components to input dimensions (i.e., axes) in the graphing component; the	• Requires a new component to be instantiated and mapped for each unique graph configuration

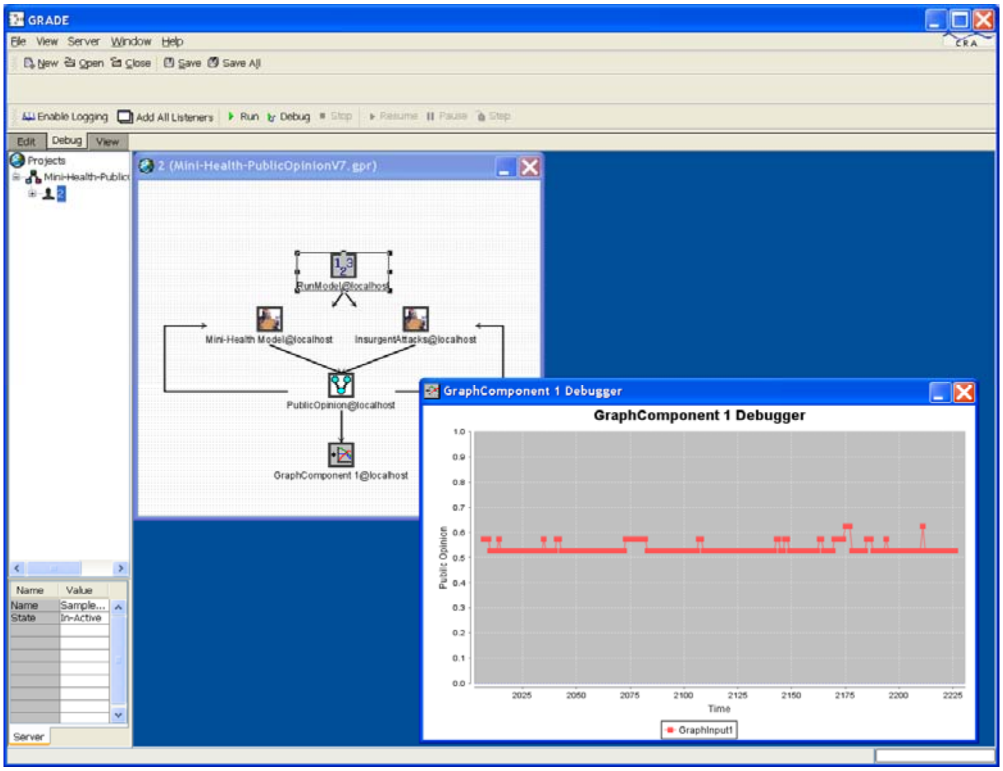
	XSL wizard can also provide basic type checking • The component could (using input groups if necessary) process information from multiple components and allow the user to graph one component's output against the other • The graphing component could also have a View GUI that would allow interactive graphing of component data model logs • Component development has a clear development path whereas making modifications to the Debug GUI would be marginally more complicated/unfamiliar • Congruent with the future direction of GRADE, which will represent GUI elements as components	• Introducing a component whose only "output" is its Debug display is a departure from the current GRADE development paradigm (but is congruent with the future direction of GRADE)

We chose the component-based approach for its flexibility, extensibility, and its ability to leverage existing features such as the XSL wizard. A screenshot of the Graphing Component being used to graph model-generated data is shown in Figure 6-2. The Graphing Component is configured like all other GRADE components: it is instantiated by dragging and dropping it from the component palette, configured by opening up its Editor GUI, and linked to other components using the XSL Wizard. The configuration interface for the Graphing Component allows the model analyst to specify whether the data to be graphed is time-series data or X-Y data, and the number of series to plot.

The Graphing Component provides a variety of useful functions to the model developer:

- *Intuitive, interactive model verification.* The graphing component is a much more useful tool for most applications for determining if a model is behaving as expected than the component listeners alone (which allow for monitoring of individual XML messages)
- *The ability to simultaneously graph outputs of multiple instances of a model.* The graphing component allows the user to simultaneously graph and compare the outputs of models that have been initialized with different parameters. An example of this would be to compare trajectories of public opinion across different regions using multiple regional models.

- *Graphing output of one component against the output of another component.* The component can be configured to receive input in pairs (using input groups) and use one component's input for the X-axis and use another's for the Y-axis. An example of this would be to chart the value of public opinion as a function of infrastructure availability.


GRADE
File View Server Window Help
New Open Close Save Save All
Enable Logging Add All Listeners Run Debug Stop Resume Pause Step
Edit Debug View
Projects
Mini-Health-Public
2
2 (Mini-Health-PublicOpinionV7.gpr)
RunModel@localhost
Mini-Health Model@localhost
InsurgentAttacks@localhost
PublicOpinion@localhost
GraphComponent 1@localhost
GraphComponent 1 Debugger
GraphComponent 1 Debugger
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Public Opinion
2025
2050
2075
2100
2125
2150
2175
2200
2225
Time
■ GraphInput1
Server

Name	Value
Name	Sample...
State	In-Active

Figure 6-2: Using the Graphing Component to Visualize Model Data in the Debug Workspace

While we still believe that specialized statistical capabilities are better left to analysis packages such as MATLAB or SPSS, the ability to graph model output within the PMESII IDE using the Graphing Component proved to be an essential capability for model verification, validation, and analysis.

6.3 Advanced Analysis Capabilities Using HASMAT

Making predictions using PMESII models requires two types of "what if" analysis, depicted in Figure 6-3. The first type, *causal reasoning*, enables analysis from causes to effects. This allows the user to consider the effects of potential DIME actions on the PMESII models under consideration. The second type, *diagnostic reasoning*, enables reasoning from effects to causes.

This allows the user to specify the desired (or actual) PMESII effects and determine the DIME actions that are most likely to achieve this result while minimizing undesirable second- or third-order effects. Supporting these two types of reasoning using PMESII models requires specific statistical sampling and analysis techniques which we have studied and implemented under a related program for AFRL/RHCS, the Human and System Modeling and Analysis Toolkit (HASMAT). While the focus of HASMAT is the modeling of social networks rather than nation states, both HASMAT and CPE require software tools to gather and analyze data generated from repeated simulations of model behavior. The approaches taken by both HASMAT and CPE to support these advanced analysis capabilities are discussed further in the subsections below.

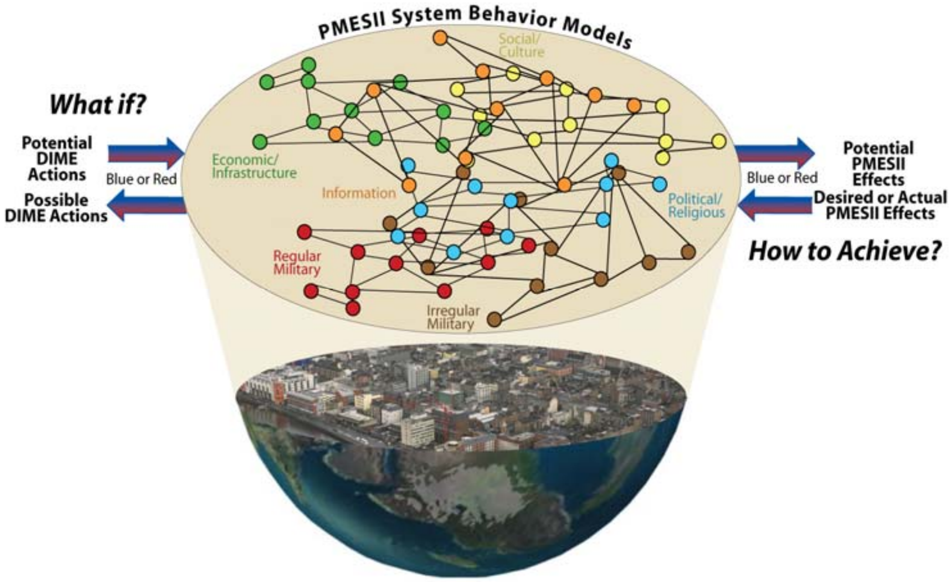


Figure 6-3: Analytical Reasoning Enabled by PMESII Model Integration

6.3.1 Causal Reasoning

In this type of analysis, the user specifies a set of DIME actions and the analysis indicates how these actions would influence the given PMESII models. Due to the nonlinearity of the systems being modeled, and the incompleteness of information about system state, it is unreasonable to expect that PMESII models will provide high-fidelity predictive capabilities. Rather, the predictive value of the models lies in their ability to generate the *distribution of plausible outcomes* across multiple courses of action. Under HASMAT we have implemented a generic Monte Carlo sampling capability to support this type of data collection and analysis. This sampling framework can be used both with HASMAT models and PMESII models generated using GRADE.

Monte Carlo sampling refers to a family of algorithms that approximate a function *f* by calculating *f(x),* for a randomly chosen *x*, over many iterations. Sampling is a useful approximation technique in cases where the function to be computed is difficult or impossible to calculate exactly. For complex nonlinear models such as PMESII models, randomized sampling provides an effective approach to approximating model outputs because it is independent of the underlying formalisms being used by the model.

Sampling can be used to analyze any model that incorporates both: 1) a representation of the cause-effect relationships between model elements; and 2) a specification of the relative likelihoods of inputs or initial states of model elements for which such conditions are not explicitly specified by the user. The first condition requires only that the model being sampled have some predictive capability. For example, Hidden Markov Models, belief networks, neural networks, and rule bases all meet this criteria; a purely analytical tool such as a Topic Tree or Concept Map does not. The second condition requires that the model specify a distribution of initial conditions for model elements, including the likelihood that various actions (either Blue or Red) will be observed. This allows the sampling algorithm to select random inputs according to a plausible distribution.

Given that the PMESII models in the system meet these two criteria, a user would perform a causal analysis using the HASMAT sampling tool in the following manner:

1. *Specify Conditions.* The user first specifies the set of assumptions to be evaluated by the analysis; this includes not only the DIME actions of interest but also assumptions about the state of hidden variables in the models. The user also specifies the number of iterations to be performed by the sampling algorithm.
2. *Select Data Collection Parameters.* The user then selects the elements within the models for which state data will be collected.
3. *Begin the Simulation.* The HASMAT tool samples the PMESII models repeatedly. At each iteration, the states of variables not explicitly set in Step 1 are randomized to a permissible state given information about the relative likelihood of the initial states of the variable. The effects of the model inputs are propagated through the model and the framework collects system state data for the variables selected by the user in Step 2.
4. *View Collected Data.* The user then views the data collected in Step 3, viewing the relative frequencies of various outcomes.

A screenshot of the HASMAT tool is shown in Figure 6-4. The tool allows the model analyst to specify initial conditions for input variables and view the resulting simulation data in a graphical format for analysis. By performing this type of simulation-based analysis for multiple DIME actions, the user would be able to determine which actions result in a greater likelihood of achieving the desired effects as well as which actions result in a greater likelihood of causing undesired effects.

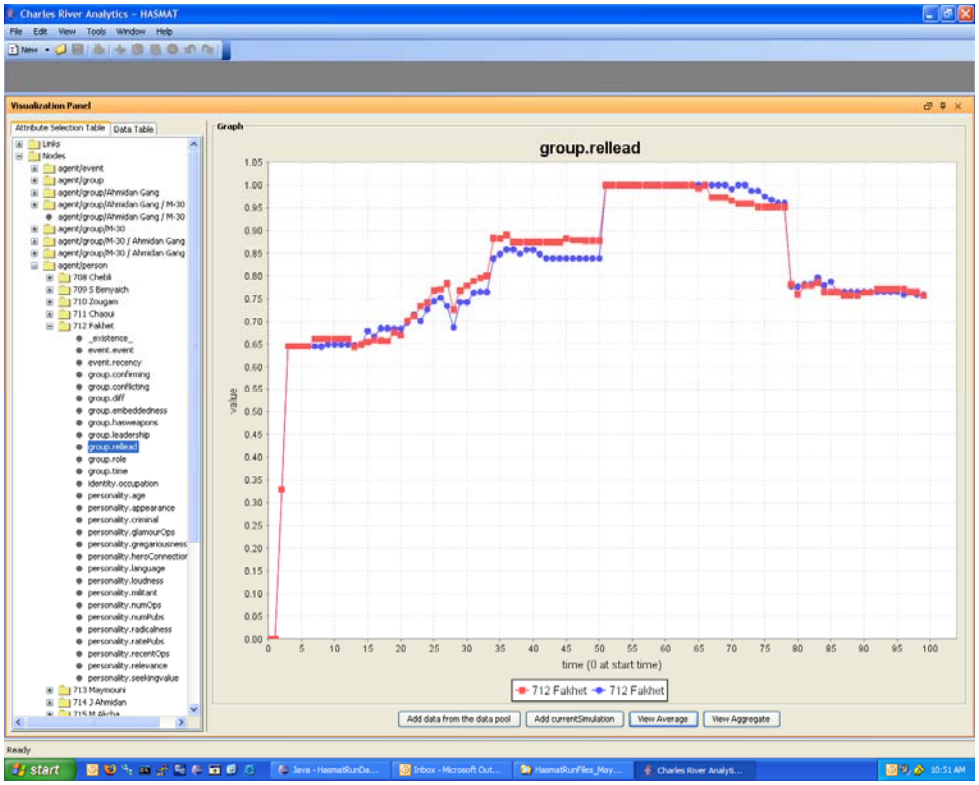


Figure 6-4: Graphing Aggregated Model Output Usng HASMAT

6.3.2 Means-Ends and Sensitivity Analysis

The second type of reasoning of interest to a PMESII modeler is *means-ends analysis*: for a given effect or system state, what are the actions that can be taken to achieve the desired state? This type of analysis is very difficult to do using heterogeneous models and remains an area of future work for both the PMESII modeling toolkit and HASMAT. Outlined here are some potential approaches to supporting this type of analysis.

One approach to answering these types of queries would be to perform a forward-chaining analysis for each set of actions under consideration; the set of actions most likely to achieve the desired result could be selected empirically based on the results of each analysis. Such an approach is clumsy and inefficient, however, since the forward-chaining reasoning process is itself computationally expensive, and performing a brute force means-ends analysis in this fashion with the large number of possible action sets that are likely to be possible would quickly become prohibitively complex and computationally expensive.

One solution is to reduce the search space of possible actions or input states using a technique known as sensitivity analysis. Sensitivity analysis computes, typically using black-box sampling techniques, how variability in the output of a model depends on variation in its inputs. Because it uses sampling, sensitivity analysis can also be applied to any type of model formalism: only the inputs and outputs are observed. In the case of reasoning using PMESII models, we can use sensitivity analysis to determine which actions or input variables are most relevant in determining the outcome or effect in which we are interested. Once we have identified a subset of relevant actions, we can then perform a brute-force, means-ends analysis in the manner described above to determine the optimal combination of those actions.

To illustrate this process further, consider the following example. Suppose a group of modelers have developed a network of PMESII models specifying the interrelationships between the economic and political elements of a particular country. A user of the CPE framework wishes to use the aggregated model to gain insight into the types of actions that can be taken to boost public confidence in the existing government. Because of the complexity of the model and the number of possible inputs and actions, the user performs a sensitivity analysis and determines that the factors most critical in determining public confidence are the supply of electricity, the visibility of police in the community, and the price of gasoline. Having identified this subset of factors, the user performs a brute-force means-ends analysis and determines that public confidence can be maximized by increasing electricity supply by 20%, maintaining the current high level of police forces, and reducing taxes on gasoline by 3%.

Because sensitivity analysis determines the variability of model output according to its inputs, it can provide results of interest other than just the relevance of an input. For example, the *rate of change* of model output as a result of input may be of even greater significance for a model user in selecting an optimal course of action. For example, if the model indicates a strong non-linearity or “tipping point” in the output variable under consideration, this would indicate the importance of gathering additional information to determine how close to this tipping point the system being modeled actually is. Or, the model may indicate that the results of an action on an output variable may be highly variable, with a large standard deviation; this would indicate a higher risk associated with the action, especially in cases where the impacts of the action being taken are difficult to control.

7. Support for Multi-Resolution PMESII Modeling

Selecting the appropriate level of modeling resolution to apply to a given PMESII analysis is an art in and of itself, especially considering the large number of variables and dimensions representing the complete operational environments. Weighing the cost in terms of computational performance against the benefits of high fidelity behavior modeling can be very difficult. However, it is rarely the case that all potential behaviors in a given scenario must be represented at high fidelity levels. Rather, only those behaviors that are likely to have the most impact on a PMESII outcome of interest need to be modeled, and likely only a subset of them need to be modeled at high levels of detail or fidelity, at different times over the course of a scenario.

What this calls for, then, are two things: 1) a pre-built "library" of models that represent the same entity/function, but at different levels of resolution; and 2) a mechanism by which these models can be "switched" in and out of the larger PMESII model environment, as a function of the dynamically-varying resolution need or fidelity requirement.

We believe that the ability to *automatically* determine how to match models (map inputs to outputs) when substituting a higher-fidelity representation for a lower-fidelity one, or vice versa, at runtime, is *not* an essential feature for multi-resolution modeling using the toolkit. In the most typical use case for multi-resolution modeling, the model developer would specify the input-output mappings between components for both the high and low fidelity versions at *design* time, while at runtime the toolkit would perform the appropriate substitutions of one version of the model for the other. By performing this step at design time the model developer creates the "pre-built library of models" identified above as the first essential element for multi-resolution modeling.

To enable the second of these key elements, the "switching" mechanism to automatically substitute one version of the model for another, we must find a way to define and dynamically recognize when it is appropriate to adjust the level of resolution of a given PMESII model component. While we did not make any specific enhancements to GRADE during this effort to support real-time switching between models of varying levels of fidelity, we have experience with using GRADE's existing capabilities to perform multi-resolution modeling from previous work in the air traffic modeling domain. This work is instructive here because it shows how, by virtue of the flexibility of the GRADE programming model, a GRADE model can itself be used to control the execution of other GRADE models.

For the project “Distributed Air-Ground Negotiation Optimization,” (DAGNEG), we modeled the air traffic dynamics of large numbers of aircraft as part of NASA’s inquiry into whether a decentralized model of air traffic management would result in fewer airspace conflicts. The decision-making and negotiating behavior of pilots and air traffic controllers was modeled using human behavior models (HBMs) implemented as GRADE agents. Because instantiating an HBM for each aircraft in the simulation was computationally prohibitive, we used a multi-resolution approach, in which an HBM for a pilot or controller was instantiated only in cases in which flight plans indicated a potential conflict; in non-conflict situations the aircraft followed pre-defined flight plans and only position and direction were tracked.

In this scenario, the logic governing model-switching was clear-cut: employ a higher-fidelity representation (Human Behavior Model) in conflict situations and a low-fidelity representation (aircraft position and flight plan) otherwise. To perform the switching, we encapsulated this decision-making logic into another GRADE model, the *Management Agent,* which consisted of three components. The Management Agent received information on the status of simulation entities as input, and instantiated other GRADE agents to serve as high-fidelity HBMs for selected entities as appropriate.

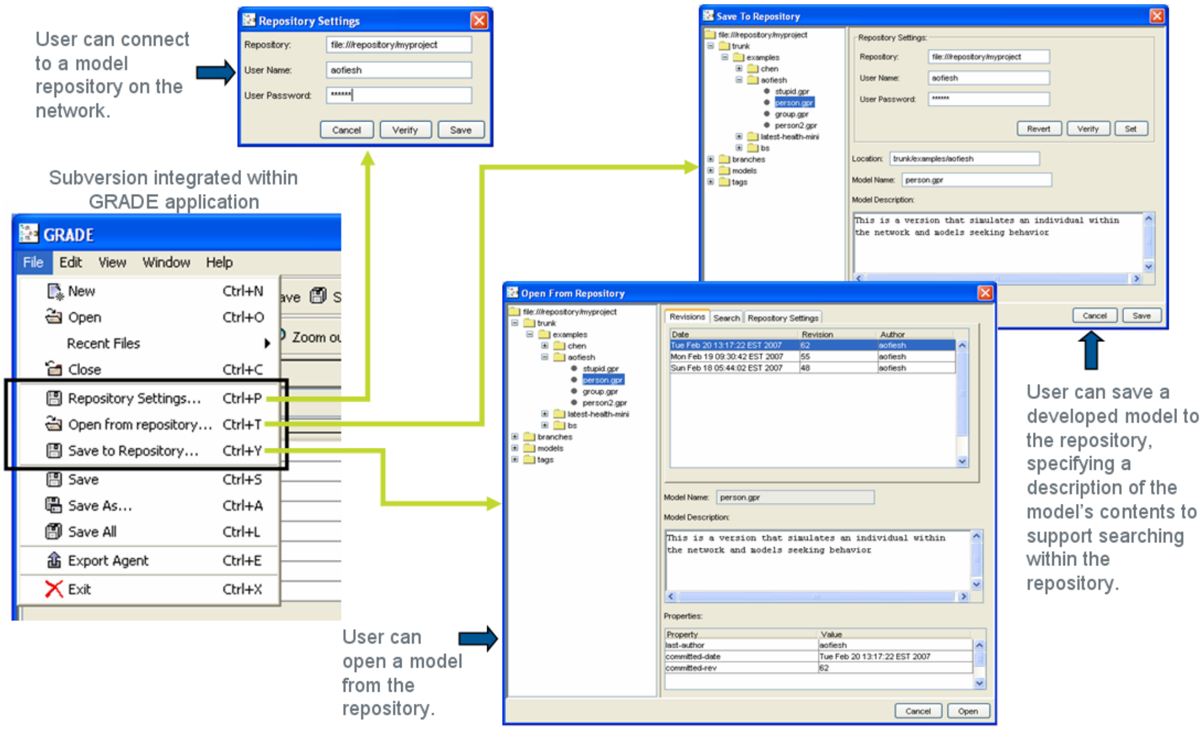
This example demonstrates the ability of GRADE models to control the instantiation of other GRADE models. While this represents a fairly sophisticated use of GRADE that would require some skill on the part of a model developer, a similar approach could be used to manage model switching for multi-resolution PMESII modeling as well. A potential direction for future development would be to build this simulation analysis and model substitution capability into GRADE as one of its native features to make this process simpler and more straightforward for the model developer.

8. PMESII Model Management Infrastructure

The final development challenge that we addressed within our PMESII modeling IDE development effort focused on the maintenance and distribution of model components and aggregates through the community of potential users, which would include not only military analysts, but also economists, psychologists, sociologists, cultural anthropologists, etc. Expertise from all of these communities will be critical to the development of complete and accurate PMESII models that can be used reliably within the CPE. As such, the IDE must support the straightforward sharing of model components, collaboration in model development, and strict model management practices.

To meet the needs of both individual model developers that require model development capabilities, as well as groups of developers that require change tracking and collaboration support, we integrated the existing open-source public license version-control technology, Subversion, within GRADE, which enables the development of shared libraries of models and careful maintenance of model development histories.

Within GRADE, there are now additional menu-driven tools that allow the model developer to "connect to" a Subversion repository maintained on an available network resource. Screenshots and a workflow diagram for these tools are shown in Figure 8-1. Subversion provides typical file-based version control features that allow us to construct a database repository of versioned GRADE model files (the configuration of a GRADE model is maintained in an XML specification file, and these are the versioned instances of models). We have also provided graphical interfaces for the model developer to open GRADE models from the connected Subversion repository, and version and save models within the repository, along with identifying and descriptive information describing the model contents and capabilities, as shown in Figure 8-2. Finally, we have provided an interface for the model developer to search for specific models within the repository, based on key terms in the model description data. The search interface is illustrated in Figure 8-3.


User can connect to a model repository on the network.

Subversion integrated within GRADE application

User can open a model from the repository.

User can save a developed model to the repository, specifying a description of the model's contents to support searching within the repository.

Repository Settings

Repository: file:///repository/myproject

User Name: aofiesh

User Password: ******

Save To Repository

Repository Settings:

Repository: file:///repository/myproject

User Name: aofiesh

User Password: ******

Location: trunk/examples/aofiesh

Model Name: person.gpr

Model Description:

```
This is a version that simulates an individual within
the network and models seeking behavior
```

Open From Repository

Date	Revision	Author
Tue Feb 20 13:17:22 EST 2007	62	aofiesh
Mon Feb 19 09:30:42 EST 2007	55	aofiesh
Sun Feb 18 05:44:02 EST 2007	48	aofiesh

Model Name: person.gpr

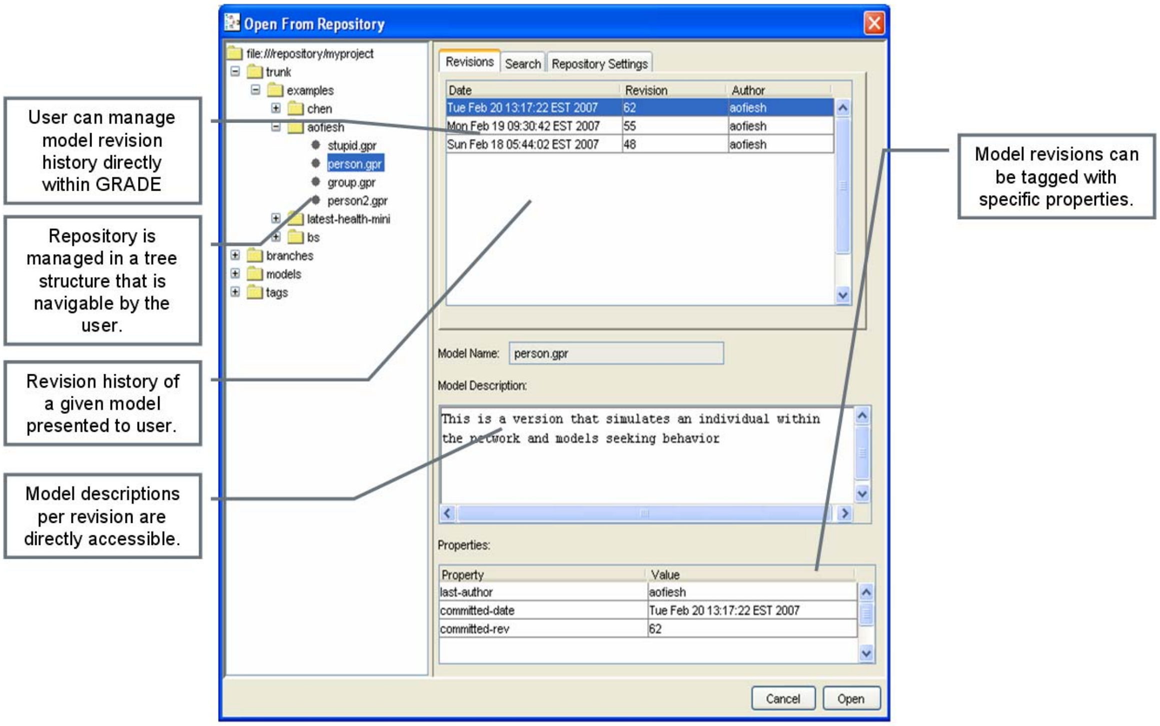
Model Description:

```
This is a version that simulates an individual within
the network and models seeking behavior
```

Properties:

Property	Value
last-author	aofiesh
committed-date	Tue Feb 20 13:17:22 EST 2007
committed-rev	62

Figure 8-1: Integrated Model Management Tools within GRADE


User can manage model revision history directly within GRADE

Repository is managed in a tree structure that is navigable by the user.

Revision history of a given model presented to user.

Model descriptions per revision are directly accessible.

Model revisions can be tagged with specific properties.

Open From Repository

Date	Revision	Author
Tue Feb 20 13:17:22 EST 2007	62	aofiesh
Mon Feb 19 09:30:42 EST 2007	55	aofiesh
Sun Feb 18 05:44:02 EST 2007	48	aofiesh

Model Name: person.gpr

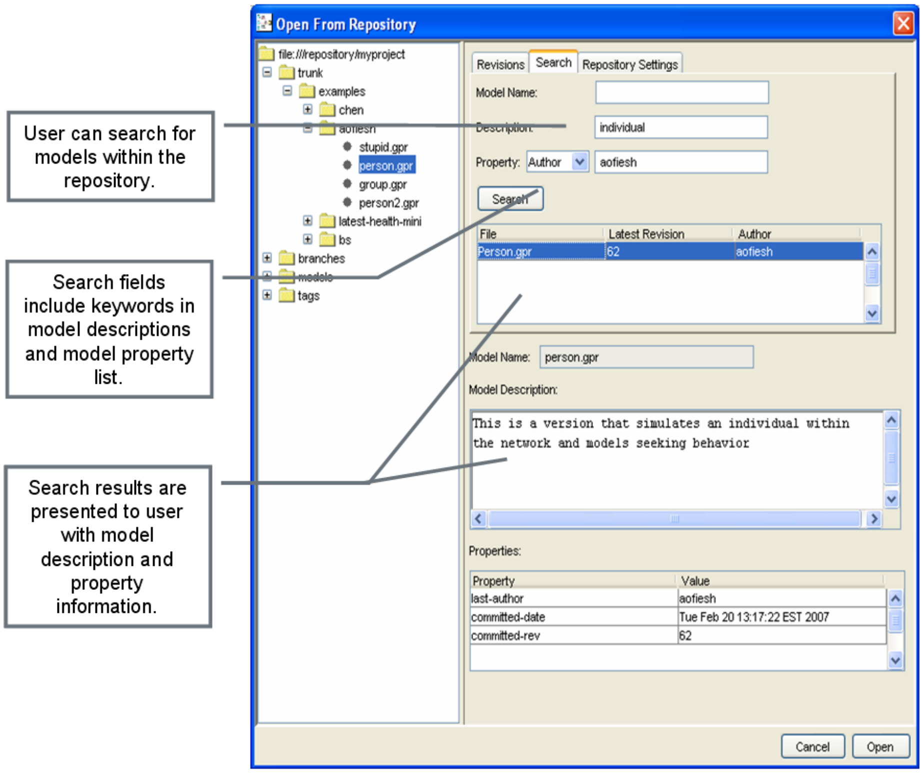
Model Description:

```
This is a version that simulates an individual within
the network and models seeking behavior
```

Properties:

Property	Value
last-author	aofiesh
committed-date	Tue Feb 20 13:17:22 EST 2007
committed-rev	62

Figure 8-2: Managing Model Revision History



User can search for models within the repository.

Search fields include keywords in model descriptions and model property list.

Search results are presented to user with model description and property information.

Model Name:

Description: individual

Property: Author aofiesh

File	Latest Revision	Author
Person.gpr	62	aofiesh

Model Name: person.gpr

Model Description:

```
This is a version that simulates an individual within
the network and models seeking behavior
```

Properties:

Property	Value
last-author	aofiesh
committed-date	Tue Feb 20 13:17:22 EST 2007
committed-rev	62

Figure 8-3: Open From Repository (Search Tab)

8.1 Designing a Model Ontology: Internal GRADE Model Study

While the search tool provides the model developer the ability to retrieve version-controlled GRADE models from the repository based on simple keyword searches of text associated with models, we also pursued a more advanced and structured approach to the descriptive “tags” associated with these models. Our initial objective was to provide an “ontology” of model descriptors that would allow us to capture the critical features of a given model within a searchable structure that could then enable more advanced search and integration of heterogeneous model components by the PMESII modeling community.

In an effort to gain a better appreciation for how an enhanced model property set could be used to help developers integrate unfamiliar models, we undertook an informal in-house study. The purpose of the study was to identify the most relevant and critical information that could be encoded in such a descriptive ontology. In the study, current and former GRADE model developers were paired together and assigned a model with which they were unfamiliar. The study task was to go through the process of understanding and running the model while documenting the types of additional information that would have been most useful for understanding the model.

The results from the study are shown in Table 8-1. The feedback from the study participants emphasized a combination of unstructured descriptive text (comments) and low-level model attributes that could be derived programmatically by the GRADE IDE for presentation to the user, e.g., component input schemas. There are a number of potential reasons for why a consensus on high-level model or domain-specific properties did not emerge. As the types of metadata recommended by study participants are analogous to the free-text comments and keyword information that are currently encoded in software documentation formats such as Javadoc (http://java.sun.com/j2se/javadoc/), the familiarity of the study group with software engineering tools and practice might have led them to look for similar tools and practices to support GRADE model development. Another possible reason could be that the study task may not have been sufficiently demanding or the models of sufficient complexity to require more sophisticated metadata for model understanding.

Table 8-1: Model Metadata Parameters Recommended by Study Participants

Metadata Entered by the Model Developer	
Parameter Name	**Description**
Top-Level and Module-Level Comments	(free text input)
Domain	Domain of the model (text input)
Required Services	List of the GRADE services required for the model (selected from a list of available services)
Input/Output Components	List of the components of the model that can be an input or output node for the model (selected from list of all components in the model)
Metadata Determined Automatically by GRADE	
Parameter Name	**Description**
Required Java Version	E.g., 1.4.2, 1.5, 1.6, etc.
Component Types Used	E.g., Belief Networks, Ptolemy, Fuzzy Logic, etc.
Component Names	List of the names of all the components in the model
Input/Output Format	The format of input or output for this model including any schemas defining the format
Other Metadata	E.g., date, author of last modification, version number, etc.

Ultimately, the most effective way to develop an effective ontology for PMESII model sharing will be to work with a population of model developers and analysts that is actively working with PMESII models. We consider this a promising direction for future work.

9. Conclusions and Recommendations

In this section, we summarize our findings in Section 9.1 and offer recommendations for a follow-on effort in Section 9.2.

9.1 Summary

During the course of our CPE Toolkit development effort, we accomplished the following results:

First, we performed a review of the state of the art in PMESII modeling capabilities to develop further insight into the full range of technical requirements to support the predictive modeling of PMESII effects as a function of possible DIME actions. The results of this review were used to identify the current and future technical requirements for the proposed PMESII modeling IDE. Our review and requirements analysis was informed by the ongoing work of AFRL/RI NO'EM research group, who shared several of their SROM models with our team.

Second, we reviewed a variety of PMESII modeling tools and chose Ptolemy as the focus of our integration effort. Based on the understanding gained through the requirements specification process, we selected, in conjunction with AFRL, the Ptolemy systems dynamics modeling framework to be the primary focus of our modeling tool integration effort. As Ptolemy is the platform currently being used by the NO'EM group for PMESII model development, this choice had a number of distinct advantages: immediate relevance to the CPE domain; availability of existing models for use as examples; Java implementation with a non-restrictive license; storage of models using the Modeling Markup Language (MoML), a well-defined XML file format with a published Document Type Definition (DTD); and an existing graphical editor (Vergil) for creating and maintaining Ptolemy models.

Third, we researched advanced strategies for model integration in partnership with our companion CPE program. While GRADE provides a base level of support for interoperable PMESII modeling by virtue of the fact that it provides a common environment for model development and a common protocol (XML) in which they can communicate, there are a number of challenges associated with resolving interoperability conflicts between PMESII models beyond the data-level. During this effort we worked with the team of our sister CPE program, "Framework for Building and Reasoning with Adaptive and Interoperable PMESII Models," to conduct research into advanced strategies for resolving model interoperability conflicts and explore practical approaches for incorporating these strategies into the PMESII modeling toolkit.

Fourth, we developed an enhanced suite of PMESII model verification and validation tools. GRADE provides a suite of model verification and validation capabilities that can be applied by the model developer to analyze model behavior, both at the individual component level and at the integrated model level, to ensure that models behave as intended by the developer. We enhanced these features by incorporating the open-source testing framework XmlUnit for model and sub-model verification, and by developing a Graphing Component to allow the model developer to visualize model output for both verification and validation.

Fifth, we developed an enhanced PMESII model analysis infrastructure. An end goal of the CPE program is to provide a modeling environment that will enable both the causal and diagnostic analysis of the complex operational environment of modern warfare, so that commanders can effectively predict the potential effects of candidate DIME actions or generate potential DIME actions that are most likely to generate some set of desired PMESII effects. In support of this objective, we created an Analyst's Interface allowing the model analyst to perform model validation and analysis at a higher levels of abstraction; we developed a Graphing Component, as indicated above, to allow the model developer to visualize model-generated data within the PMESII modeling IDE; and we developed a framework for advanced analytical techniques under HASMAT, a related program for simulation-based model analysis.

Sixth, we developed strategies for supporting multi-resolution modeling. It is rarely the case that all potential behaviors in a given scenario must be represented at high fidelity levels. Rather, only those behaviors that are likely to have the most impact on a PMESII outcome of interest need to be modeled, and likely only a subset of them need to be modeled at high levels of detail or fidelity, at different times over the course of a scenario. What this calls for is a mechanism by which models of varying fidelity can be "switched" in and out of the larger PMESII model environment. During this effort we explored the use of GRADE models as a means of controlling this switching process for the PMESII domain, as informed by our previous work on multi-resolution modeling in other domains.

Finally, we developed a model management infrastructure for the PMESII modeling toolkit. The development of practical PMESII models that will accurately reflect the complex operational environments of urban and national warfare will require model inputs and expertise from a wide range of disciplines, including economics, political science, sociology, psychology, and cultural anthropology. To support the development of such multi-disciplinary models our toolkit must support development and adaptation by multiple users, including maintenance of individual models and model libraries. To meet the needs of both individual model developers that require model development capabilities, as well as groups of developers that require change tracking and collaboration support, we integrated the existing open-source public license version control technology Subversion within GRADE, which enables the development of shared libraries of models and careful maintenance of model development histories.

9.2 Recommendations for Future Work

Potential directions for future work include the following:

- *Integration of additional PMESII modeling technologies.* In particular the integration of our in-house tools OCCAM, for social network modeling and concept graph development, and PRISM, for reasoning using situation theory, could prove valuable for modeling and relatively straightforward from a software engineering perspective.
- *Implementation of advanced model incompatibility resolution tools.* During their research effort, our companion CPE program explored a number of strategies that could be employed to support the model developer in integrating heterogeneous models with different types of model incompatibilities. A potential avenue for future work would be to select one or more of these approaches for implementation and integration into the toolkit.
- *Advanced model analysis tools.* Currently rudimentary support for Monte Carlo simulation runs and large-scale data capture is provided by the HASMAT toolkit. In future work these analytical tools could be extended to provide a richer interface for specifying model initialization parameters and support for diagnostic reasoning using sensitivity analysis.
- *Built-in support for model-switching in multi-resolution modeling scenarios.* Currently model-switching logic can be defined by using an additional GRADE agent; by providing this functionality as a built-in feature of GRADE this process could be made easier and more straightforward for the model developer.
- *Metadata for model management.* In partnership with a group of PMESII model developers and analysts, we would determine a set of descriptive parameters to apply to models to support model integration and re-use. Based on the results of this study we would then develop tools in the PMESII model management infrastructure to leverage these parameters.

While any of the areas listed would prove to be valuable directions for future work, the most essential avenue would be to perform a study in partnership with a group of model analysts and subject matter experts to determine the effectiveness of the toolkit as it now stands for creating useful and meaningful PMESII models. Such a study would inform the future development of all aspects of the tool, including additional modeling technologies, advanced analytical capabilities, metadata for model management, etc. At this point in the development of the toolkit it is essential to refine its capabilities in partnership with the potential user community to maximize the benefit to that community.

10. References

Allen, J. G. Commander's Automated Decision Support Tools: Briefing to Proposers' Symposium for DARPA's Integrated Battle Command Program. 12-15-2004a. DARPA.

Allen, J. G. Commander's Automated Decision Support Tools: Briefing to Proposers' Symposium for DARPA's Integrated Battle Command Program. 12-15-2004b. DARPA.

Barwise, J. & Perry, J. (1983). *Situations and Attitudes*. Cambridge, MA: MIT Press.

Canas, A., Hill, G., Carff, R., Suri, Lott, J., Eskridge, T. C. et al. (2004). CmapTools: A Knowledge Modeling and Sharing Environment. In *Proceedings of First International Conference on Concept Mapping*, (pp. 125-133). Pamplona, Spain: Universidad Pública de Navarra.

Devlin, K. & Rosenberg, D. (1991). Situation Theory and Cooperative Action. In *Proceedings of Third Conference on Situation Theory and Its Applications*. Oiso, Japan.

Gluck, K. A. & Pew, R. W. (2001). Lessons Learned and Future Directions for the AMBR Model Comparison Project. In *Proceedings of Proceedings of the 10th Conference on Computer Generated Forces and Behavioral Representation*. Orlando, FL: DMSO.

Jensen, F. V. (1996). *Bayesian Networks Basics.* Aalborg University, Denmark.

Lewis, C. M. (1990). Visualization and Situations. In *Proceedings of Second Conference on Situation Theory and Its Applications,* J. Barwise, J. M. Gawron, G. Plotkin, & S. Tutiya (Eds.). Loch Rannoch, Scotland.

Robbins, J. D., Deckro, R. F., & Wiley, V. D. (2005). Stabilization and Reconstruction Operations Model (SROM). In *Proceedings of 4th Workshop on Critical Issues in Information Fusion*. New York.

Steinberg, A. (2005). An Approach to Threat Assessment. In *Proceedings of the Eighth International Conference on Information Fusion*. Philadelphia, PA.

Steinberg, A. & Bowman, C. (2004). Rethinking the JDL Data Fusion Levels. In *Proceedings of National Symposium on Sensor and Data Fusion*. JHAPL.