AN APPROACH TO EVALUATING COMPUTER NETWORK SIMULATION TOOL SUPPORT FOR VERIFICATION AND VALIDATION

by

Jonathan Douglas Gibbs

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

August 2009

AN APPROACH TO EVALUATING COMPUTER NETWORK SIMULATION TOOL SUPPORT FOR VERIFICATION AND VALIDATION

by

Jonathan Douglas Gibbs

has been approved

May 2009

ABSTRACT

Domain-specific tools are frequently used in modeling and simulation (M&S) of computer networks. While such tools are important to varying degrees across different stages of the modeling and simulation lifecycle, they can also impose major constraints on the expressivity of models, and limit the ability to carry out model verification and validation (V&V). An approach is devised for evaluating domain-specific M&S tools, in which one analyzes the model development process for each tool, and identifies key advantages and limitations that affect different aspects of development, particularly the ability to perform V&V. One then chooses a set of features based on one's requirements, which broadly capture the observed characteristics of the selected tools. After assigning a quantitative or qualitative rating to each tool with respect to each feature, one can assess the effectiveness of each tool for use in a given M&S project, based on the assigned ratings, and the importance of each feature to the project in question. This approach is demonstrated using OPNET IT Guru and the INET Framework, two domain-specific computer network M&S tools. It is found that both tools have certain distinct advantages and limitations, with IT Guru's user interface (UI) allowing for faster and more user-friendly model construction, while the source code included in INET provides a thorough and unambiguous description of model structure and behavior, which can be a crucial asset during V&V; conversely, model construction in INET tends to proceed more slowly, and requires a greater level of software engineering maturity on the part of the model developer, while IT Guru does not provide enough information on model structure and behavior to strongly support V&V. These and other observations determine the scores assigned for the set of features used to compare the two tools. Such an analysis allows one to determine if a given tool is likely to support V&V and other activities of model development. Some observations regarding future research in the evaluation of M&S tools are also presented.

TABLE OF CONTENTS

SECTION Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

1 INTRODUCTION

Modeling and Simulation (M&S) is a powerful tool that can provide insight into the dynamics of complex systems, both natural and man-made. When used correctly, M&S allows one to perform experiments that may be impractical to carry out by other means, and the knowledge which is gained from experimenting with a model may be useful in predicting the behavior of a corresponding real-world or hypothetical system. Of course, this capability is inherently contingent on the relationship between the model and the system which it is intended to depict. Due to limitations in time, financial resources, computing capability, and knowledge regarding the real-world system, a simulation model is almost always merely an approximation, designed to represent particular features and behaviors of the system under a given set of conditions, with certain degrees of accuracy and precision. It is within the context of such a finite set of parameters that one can determine not only the usefulness of a model, but also the correctness of its design and implementation. This second determination is achieved by the process of verification and validation (V&V). According to current U.S. Department of Defense policy, verification and validation consist, respectively, of ascertaining that the "implementations and associated data" of the models and simulations "accurately represents[sic] the developer's conceptual description and specifications," and that these same implementations and associated data "are accurate representations of the real world from the perspective of the intended use(s)" of the models (US DoD 2008, 3). At a more abstract level, verification seeks to determine the correctness of the model relative to its intended implementation, and validation seeks to determine the correctness of simulation behavior relative to data regarding the real-world system.

One of many fields where Modeling and Simulation is of significant use is in research regarding computer networks (Bhargavan et al 2002; Wong, Dalmadge, and Fiedler, 2007). The flexibility of M&S means that it can be used to model anything from physical data transmission to enterprise-level services and applications. Computer networks – and computer-based systems in general – intuitively appear to lend themselves to M&S, since a great deal of their dynamics and composition can be inferred from industry-wide standards, manufacturer's specifications, and – in many cases – publicly available source code or software documentation. This situation has given rise to a number of modeling and simulation tools specifically tailored to the domain of computer networks, such as OPNET's IT Guru Network Planner (OPNET Technologies n.d. b) and the open source "Network Simulator" ns-2 (The Network Simulator –

ns-2 n.d.), as well as libraries of simulation models depicting computer network components, such as the open-source INET Framework (OMNET++ Community Site n.d. a) of models for the likewise open-source OMNET++ simulator (OMNET++ Community Site n.d. b). While the availability of such software is certainly convenient, it does not alleviate the need for judicious planning and design of simulation models of network systems, nor for those simulation models to undergo thorough verification and validation in the context of their intended operating parameters.

It is also important to bear in mind that the use of pre-built tools carries with it several potential liabilities. Particularly in the case of closed-source environments, these tools can severely restrict the flexibility of model construction, and can make it difficult or impossible to employ components built with the ideal granularity for modeling the particular characteristics of the system which are of interest. Using closed-source tools can also make an exacting V&V of the resulting simulation models effectively impossible: without access to sufficient knowledge regarding the behavior of the models and the underlying simulator, one can never really guarantee that the models will behave as one intends. In effect, the correctness of the tool must either be taken for granted, or rationalized in some way. This is the approach taken in (Wong, Dalmadge, and Fiedler, 2007), where the authors argued that the market position and track record of OPNET's IT Guru (OPNET Technologies n.d. b) network modeling and simulation software implied that the logic of its simulation models was "acceptable" for their purposes. Similar assumptions are routinely made regarding the correctness of compilers, API's, database programs, and a host of other commonly used software tools. Many times, such assumptions are borne out; occasionally, they are not. It is up to the individual user of such a tool to determine how much trust is warranted, and how much uncertainty can be tolerated.

However, regardless of whether or not the source code for a model library or simulation environment is available, a thorough understanding of the tool's behavior and limitations is necessary before one can properly determine how (or even if) that tool can be applied to the M&S task at hand. Such knowledge is equally vital when attempting to verify and validate a simulation model built using a preexisting framework; and the need for this knowledge persists throughout the entire lifecycle of the model.

1.1 Contributions of this Work

The goals of this work are to:

- Devise and demonstrate an approach for assessing how well a given component-based, domain-specific modeling tool supports verification and validation, and how the tool impacts the V&V process.
- Present an approach for evaluating the general benefits and limitations of a given component-based, domain-specific modeling tool, and exemplify this approach through an analysis of two modeling tools within the domain of computer network systems.

In the remainder of this thesis, an overview will be given regarding some of the specific considerations which need to be addressed when planning and designing simulation models of computer network systems. Several works related to verification and validation of simulation models will be presented, to both provide an overview of the range of existing research in this area, and describe how this thesis seeks to address needs which are not adequately met by previous efforts in this field. The process of model development in IT Guru and the INET Framework will be discussed in order to demonstrate some of the characteristics of current domain-specific M&S tools designed for use in modeling computer networks. A pair of models developed in IT Guru and INET, together with a description of how those models were verified and validated, will be presented for illustrative purposes. Tools such as IT Guru and INET will be contrasted with general-purpose simulators such as DEVS-Suite (ACIMS n.d.), which lack the detailed hardware and protocol behaviors built into heavily domain-specific environments, but often provide increased flexibility for constructing models of nearly arbitrary scale and composition. We will conclude by describing an evaluation framework which can be used to assess and compare domain-specific M&S tools such as IT Guru or INET in terms of their potential impacts on the development process as a whole, and on verification and validation in particular.

2 CONCERNS WHEN DESIGNING COMPUTER NETWORK SIMULATION MODELS

It should be noted that the word “network” will occasionally be used in this work as shorthand for the phrase “computer network.” “Computer network,” in turn, is being used in the colloquial sense, to refer to a collection of (typically) self-contained computing devices of sufficient scale, such as laptops, hubs, routers, wireless sensors, web servers, etc, together with an automated mechanism by which data can be exchanged between those devices. These devices, known as “nodes,” are connected to some degree by individual “links,” which incorporate some physical medium across which data is transmitted between nodes. This thesis, and the research that underlies it, is primarily concerned with “wired” networks, in which the physical medium used to propagate data is typically a manufactured cable. Several existing tools provide varying levels of support for modeling networks that incorporate a range of wireless communications technologies, including specialized wireless networking M&S tools such as GloMoSim – the Global Mobile Information Systems Simulation Library (MLS n.d.) – as well as IT Guru and the INET Framework themselves, which provide models incorporating descriptions of select wireless protocols through either optional extensions to the standalone product (in the case of IT Guru), or through experimental models packaged with the tool (in the case of INET). However, it was determined that expanding the scope and complexity of the domain considered as part of the research underlying this thesis would likely detract from the intended focus on relatively general characteristics of individual systems, models, and tools. Based on the commonality of concepts and methods in modeling and simulation of network systems, we expect the research outcomes described in this thesis to still be useful with regards to wireless networks.

2.1 Comparison to Software Design

As with software design in general, the selection of abstractions that will describe the structure of a simulation model is a crucial phase of simulation model design, since these abstractions will frame the remainder of the development process. Good abstractions can facilitate the creation of a meaningful and intuitive design, which can reduce the difficulty of software implementation, testing, and maintenance. Since the direct product of model development is effectively an approximate representation of a system, one can argue that the abstractions used to describe that representation play an even more prominent role than the abstractions which underlie a more general software design. Decisions made during software design can have lasting effects on the complexity and performance of the final product, another characteristic shared by simulation model design. The most obvious driving force behind software design is the required functionality of the resulting product; if the final design is an extremely poor-fit for these requirements, or is incapable of addressing them at all, then the development process has almost certainly failed to achieve its objectives. This is again applicable to simulation model design as well; throughout the entire lifecycle of model development, an overriding concern must be the required functionality of the model. A "good" simulation model design must, at a bare minimum, describe a system which is adequately equipped to address the specific questions or needs which prompted the model's development.

2.2 Overdesign of Simulation Models

Although it is undeniably imperative that a simulation model be sufficiently detailed to fulfill its requirements, there are many practical reasons to avoid overdesigning a simulation model; that is, to avoid producing a design which has significantly greater complexity and functionality than what is dictated by the requirements. While extra features, if well thought-out and properly implemented, can often be appreciated by end-users of typical software products, extra functionality in a simulation model can significantly complicate the verification and validation process and increase the difficulty of maintaining the model, and can also reduce the generality of the model, which may inhibit reuse. The impact of overdesign on model implementation is even more obvious; adding unnecessary features to the design of a model can significantly increase the time required to implement that design. Another potential consequence of overdesign is increased simulation time: as one raises the complexity of the behaviors which a model exhibits, one generally also expands the execution time needed to implement those behaviors within the simulation environment. If simulating a model requires too much time on available hardware, or if the simulation consumes too much memory, then it may well turn out to be of no practical use.

2.3 Choosing Abstractions in Network System Modeling

Computer network systems are a modeling domain in which information on the composition and behavior of much of a given real-world system is often available in excruciating detail. A key consideration in avoiding overdesign in general – and particularly in such cases – is to assess the scale and fidelity at which the various aspects or layers of the system must be modeled in order to fulfill the requirements for the modeling and simulation effort. Is the exact behavior of a given transport-layer, network-layer, or even link- or physical-layer protocol actually relevant to the results of the simulation? If not, which behaviors or aspects of this protocol are worthy of consideration? Similar questions must be answered regarding the physical links between nodes in the network, the various software and hardware components of each node in the network, and any other objects found in the real-world or imaginary system one is seeking to model, such as people who use the various nodes, and characteristics of the environment which impact the behaviors of links between nodes. Beyond assessing the structures and behaviors of individual model components, one needs to also consider the interactions between those components. What level of detail is necessary and appropriate for representing a particular interaction? Are there interactions which are effectively irrelevant to the objectives of the modeling effort? Which ordering constraints must be obeyed by sequences of interactions within the final model, and which ordering constraints can be safely ignored while still achieving the goals of the M&S project?

It is easy to see that the concerns listed above are closely tied to the objectives of the M&S effort. Details of a computer network system that are crucial when analyzing the spread of an Internet worm may be quite different from those that need to be addressed when evaluating the performance of a routing protocol, estimating the average bandwidth use along a given link, or selecting a compression scheme and bit-rate for streaming live multimedia content. However, the scale and composition of the computer network system itself must also be taken into account. As Floyd and Paxson (2001, 393-396) note, dissimilar technologies hidden beneath relatively uniform veneers such as the Internet Protocol (IP) can still have a noticeable impact on network behavior. Depending on one’s objectives and the composition of the system being modeled, it may be necessary to account for a variety of distinct behaviors exhibited by different links, nodes, or subnets. Floyd and Paxson (2001, 392-395) also demonstrate that the traffic and technologies found within the Internet have often evidenced nonlinear and – sometimes – seemingly unpredictable patterns of growth. One consequence of such issues is that the technologies and traffic patterns which a model needs to depict may vary greatly depending on whether one is modeling a historical computer network system, a currently deployed system, or a proposed system for use several years in the future. The interactions which must be depicted in order to capture the behaviors of such a system that are relevant to one's objectives will clearly play a critical role in determining the appropriateness of a given set of abstractions used for designing the model.

2.4 Using Prebuilt Domain-Specific Tools

The potential for overdesign of a simulation model is an obvious consideration to keep in mind when determining how appropriate a given simulation environment or M&S tool is for a particular M&S task. Adopting complex and detailed models such as those found in IT Guru or the INET Framework can greatly increase the number of variables which must be addressed during V&V, as well as the number of specific questions which must be answered during model design and implementation. Depending on the tool one employs, these questions may even extend beyond the models themselves. When Sarjoughian and Shaukat (2008, 21-22) compared modeling under the general-purpose DEVS framework and the domain-specific ns-2 tool, they found it was "necessary to delve into the inner workings and implementation of the ns-2 simulation engine" in order to properly understand the models they were constructing in the latter environment. However, it is also important to consider the ways in which domain-specific tools can constrain the development of a simulation model. By forcing one's model to conform to a particular domain-specific environment, one can reduce the types of interactions which are permitted between model components, and even restrict the types of components which can be built. Sarjoughian and Shaukat (2008, 22) note that general-purpose tools are often used to conduct M&S "at high levels of abstraction"; such tools can free both the model developer and the simulation model from the low-level details inherent in many domain-specific tools, and allow for a range of abstractions not easily accommodated by a domain-specific tool. The use of prebuilt tools also creates potential problems when maintaining a simulation model, since updates and changes to the tool itself are generally beyond the control of the individual modeler, yet these changes must still be accommodated by the modeler if he/she wishes to preserve compatibility with their existing models.

Of course, these concerns must all be weighed against the benefits which a domain-specific tool can provide, to determine if the tool should or should not be employed. Often, a tool will incorporate useful abstractions that assist in formulating and visualizing characterizations of the structure and behavior of a variety of systems. Simulation models from a commonly used model library will generally undergo testing and evaluation by a number of users, and be employed to model a wide range of systems. A prebuilt tool can provide significant savings in time and money during the development process, especially if the functionality of that tool is ideally suited to the requirements for the simulation model. In order to assess exactly how applicable a given tool is to a specific M&S project, one first needs to understand the capabilities and limitations of that tool.

Table 2.1. Comparison of IT Guru and INET

OPNET IT Guru	OMNET++ and INET Framework
Licenses must be purchased to obtain full functionality, or to use the software for commercial purposes	Licenses must be purchased only when using the simulator for commercial purposes
Model and simulator source code is not included	All model and simulator source code can be viewed and edited
New models cannot be coded directly by the user	New models can be coded directly by the user
Additional model libraries can be purchased from OPNET and other vendors.	Additional model libraries are available for free from the user community
Models and simulator are developed and maintained by OPNET Technologies, Inc. and its corporate partners	Included models and simulator are developed and maintained by Simulcraft, Inc., while additional models are developed and maintained by the user community
Numerous adjustable attributes permit customization of prebuilt models, but further changes to models cannot be made	Fewer attributes are available for customizing prebuilt models, but arbitrary changes can also be made to the models
Prebuilt models support a wider range of protocols and devices	Prebuilt models support fewer protocols and devices
Prebuilt models of vendor-specific devices are included or available for purchase	Prebuilt models of vendor-specific devices are not available
Simulator is intended only for use with the computer networks domain, and models from other domains cannot easily be included	Simulator can be adapted to a variety of domains, and models from other domains can be incorporated into the INET Framework
Software-hardware composition of prebuilt node models is hidden from the model developer	Software-hardware composition of prebuilt node models can be determined from source code
Degree of coupling between model components is unknown, and components cannot be separated	Model components are sometimes tightly-coupled, but they may be separated
Little or modest level of basic programming is sufficient	Maturity in software engineering including programming is required

Table 2.1(above) presents a comparison of several basic qualities of IT Guru and the INET Framework, the two tools that will be considered in this thesis. Many of the items in the table – and some of their consequences – will be explained in more detail in sections 4 and 5.

3 VERIFICATION AND VALIDATION

3.1 Goals of V&V

Verification and validation represents an essential part of simulation model development, as it is through V&V that one "seeks to establish the credibility of a model and its results within the context of one's requirements for that model" (Gibbs and Sarjoughian 2009, 2). In paraphrase of the *VV&A* (verification, validation, and accreditation) *Recommended Practices Guide* (RPG) (MSCO 2006) hosted by the Modeling and Simulation Coordination Office (MSCO) of the United States Department of Defense (DoD), one can view verification as attempting to answer the question "Did I build the model or federation correctly?" while validation addresses the question of "Did I build the correct model or federation?" (Gibbs and Sarjoughian 2009, 2). In describing the need to address both of these questions, another analogy to software engineering may be of use. For all but the most trivial of software applications, it is virtually impossible to conduct truly exhaustive testing of all possible contingencies; hence, software developers typically rely on a combination of so called black-box, white-box, or gray-box testing of the application's behavior, together with a careful review of the underlying design and code used in the application. This parallels the V&V process applied to simulation models, in which validation of simulation results is used in combination with verification of the model's design and implementation, due to the impracticality of directly validating the model's behavior under all possible conditions. In effect, "verification provides an essential means of understanding how a model may respond to circumstances or configurations that could not be directly addressed during validation" (Gibbs and Sarjoughian 2009, 6). One also finds that verification without validation is likewise insufficient; such an approach can be likened in some ways to trying to test a software application without actually compiling and executing the code. In fact, the resulting problem will be even worse in the case of a simulation model, due to the fact that the most such models are just an approximation of the intended system; barring a rigid theoretical proof of the correctness of one's design, there is no reliable way of determining if the approximations used still adequately describe the intended system without executing the simulation to observe how it behaves.

Freigassner, Sarjoughian, Praehofer, and Zeigler (2001, 1) state that one can view M&S as involving "three types of entities: real system, model, and simulator." Among these entities exist two prominent relationships referred to by Freigassner et al (2001, 1) as the “modeling relation” and the “simulation relation”; the modeling relation is defined between real systems and models, while the simulation relation exists between simulators and models. They describe verification as “the process of assessing the correctness of the simulator, with respect to the model”; that is, verification can be said to be concerned with the correctness of the simulation relation (Freigassner et al 2001, 1). Validation, in turn, is focused upon the modeling relation, and deals with determining “how well model-generated behavior agrees with observed system behavior” (Freigassner et al 2001, 1). In describing validation, Freigassner, Sarjoughian, Praehofer, and Zeigler (2001, 1) present the concept of an experimental frame (EF) (Zeigler, Praehofer, and Kim 2000), which they describe as “an operational formulation of the objectives that motivate an M&S project”; they emphasize the necessity of conducting validation “with respect to an experimental frame of interest.” According to Freigassner et al (2001, 1), the “primary question” during validation – and a question which is effectively equivalent to, though significantly less vague than, the MSCO RPG’s “Did I build the correct model or federation?” (MSCO 2006) question – is “[Do] the real system and the model… [demonstrate equivalence] with respect to the experimental frame?”

3.2 Role of V&V in the M&S Development Process

To illustrate how verification and validation can be involved in a typical simulation model development process, we will briefly describe the stages of the Federation Development Process (FEDEP) modeling and simulation development cycle, and discuss those stages that are closely related to the focus of this thesis. The development cycle chosen is termed the “M&S Development / Preparation Process,” part of a diagram of an even broader “Problem Solving Process,” presented in the Key Concepts section of the MSCO’s VV&A RPG (MSCO 2006). Fig. 3.1 (next page) depicts a partial view of the original “Problem Solving Process” diagram, presenting the stages of that “M&S Development / Preparation Process” (MSCO 2006).

The process described in Fig. 3.1, which is partly influenced by the perspective of the DoD’s High Level Architecture (HLA) concept (Dahmann, Fujimoto, and Weatherly 1997), is broken into three paths of development, based on the nature of the intended product of the M&S effort. The first path, “Prepare Legacy M&S,” deals with the reuse of existing, and generally non- HLA compliant, legacy M&S products.

The second path, “Develop New M&S,” deals with developing new simulation models, and the third path, “Construct Federation,” addresses the construction of a federation of simulation models that have themselves been developed using the first and second paths. All three paths begin with the common step of refining the requirements of the M&S effort. For convenience, labels have been assigned to the paths and their various steps in Fig. 3.1. The figure also shows the associated “V&V Process” presented in the Overall Problem Solving Process in (MSCO 2006). The general role of V&V in each of the thee paths of development will be discussed here, although it should be noted that the main focus of the remainder of this thesis is on development in accordance with path II, the “Develop New M&S” path, in order to restrict the scope to a manageable level.

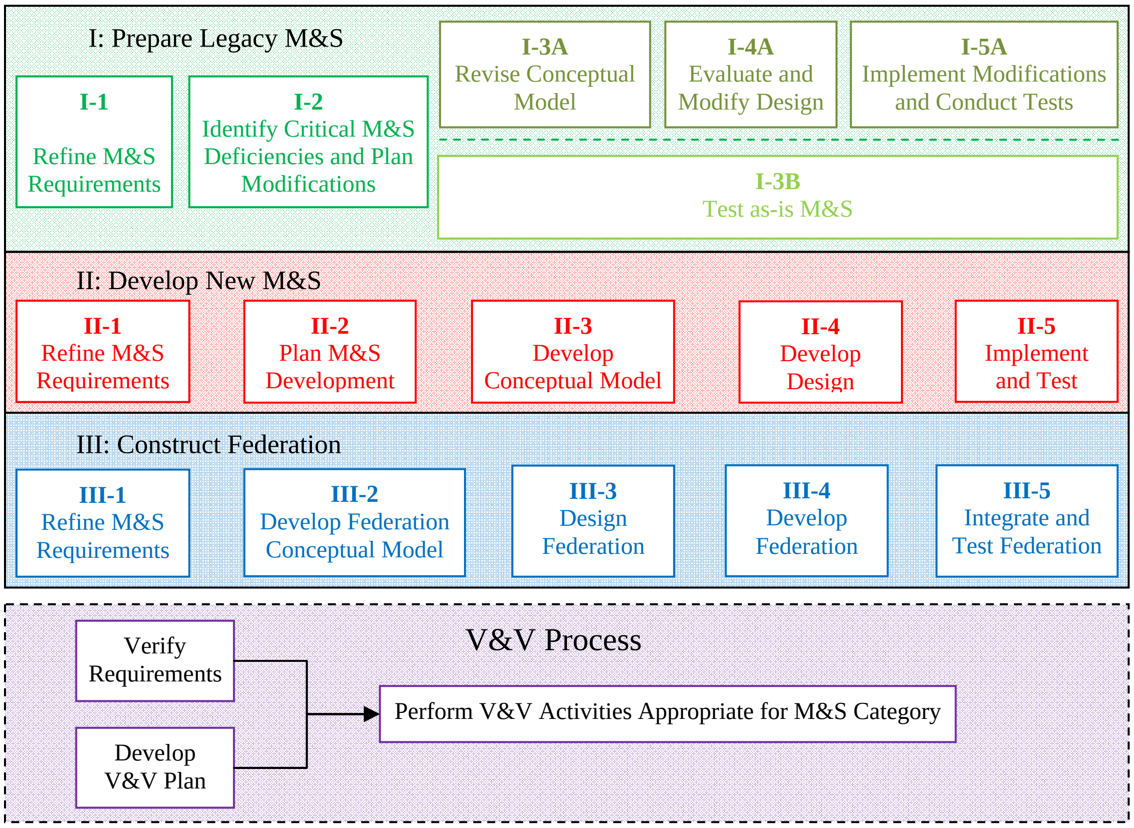

Fig. 3.1. M&S Development / Preparation Process

When preparing legacy simulation models, step I-2 requires the model developer to identify critical deficiencies of the legacy model, and to plan how those deficiencies should be addressed in order to meet the requirements for the model. One can also envision the first of these two tasks as an attempt to verify that the legacy model and its design are consistent with one’s requirements. The model developer then proceeds to one of two remaining sequences of steps, following either steps I-3A, I-4A, and I-5A, or step I-3B. The first such sequence, employed when changes to the legacy simulation model are required, involves first revising the conceptual model underlying the legacy model to accommodate the necessary changes (I-3A), then evaluating and modifying the design of the legacy model (I-4A), and finally implementing the modifications and testing the resulting simulation model (I-5A). Verification is involved in determining that the revised conceptual model addresses one’s requirements, and that the modified design captures the revised conceptual model. Verification is also needed to ascertain that the final implementation corresponds to the modified design, and that both the design and implementation address the requirements. Validation, meanwhile, is a key component of testing (I-5A), and is needed to determine that the behavior of the final simulation model adequately represents the desired aspects of the intended system. Validation is also called for the in the alternative sequence of concluding tasks, which consists of the sole step (I-3B) of testing the existing legacy simulation model in the event that no modifications were deemed necessary.

In developing a new simulation model, the process depicted by the RPG calls for the model developer to plan the development of their simulation model (II-2), develop a conceptual model of the system (II-3), develop a design of the simulation model (II-4), and then implement and test the simulation model (II-5). Verification must again be applied to the conceptual model, design, and implementation, to determine that each properly represents the results of the previous stages of development. In addition, validation is once more called for during testing of the implemented simulation model (II-5), to determine if data obtained from simulations of the model are sufficiently consistent with data describing the particular aspects of the behavior of the system which the simulation model is intended to depict.

In constructing a federation, the RPG calls for developing a conceptual model of the system (III-2), designing the federation (III-3), developing the federation (III-4), then integrating the federates and testing the resulting federation (III-5). Similar to the previous two paths of development, this third path again necessitates verification of the conceptual model and design, to help ensure that the federation will be capable of addressing one’s requirements. However, verification of the implemented federation is likely to be more limited in detail, focusing on how well the federation as a whole addresses the requirements and represents the conceptual model and design. The majority of verification of a federation generally takes place during the development of the individual federates (i.e., paths I and/or II). The implemented federation will still call for further validation during the concluding testing phase (III-5), however, as the effects of interactions between federates cannot generally be accounted for completely by validating the behaviors of the federates in isolation.

3.3 Characterizing V&V and its Associated Activities

The overview given in section 3.2 is by no means comprehensive, and it glosses over several additional activities. The RPG presents V&V itself as part of a larger Verification, Validation, and Accreditation (VV&A) process, in which accreditation is additionally employed to assess the overall creditability of the model or federation. The V&V Process presented in Fig. 3.1 is noticeably short on details, as are the descriptions that have just been provided regarding how V&V fits into the model development process. This is partly due to the varying and potentially ambiguous scope and scale of verification and validation: the table of V&V techniques presented in (Balci 1998, 45-47) hints at the tremendous variety of activities that can be said to relate to V&V, and at the challenge of indentifying an "optimal" approach for verifying and validating a given simulation model.

One can find a number of ways of characterizing verification and validation, both by themselves, and in terms of a particular model development process. Balci (1998, 42) sets down a series of fifteen "Principles" relating to VV&A, describing why VV&A is necessary, as well as some of the inherent challenges and limitations of VV&A of simulation models. Several of these principles have already been described in this thesis, including the need to evaluate the "credibility" of a simulation model "with respect to… the M&S objectives" for that model, as well as the observation that "simulation model validity does not guarantee the credibility and acceptability of simulation results," due in part to the effective impossibility of achieving "complete simulation model testing" (Balci 1998, 42). Balci further presents two distinct model development life-cycles, one of which – depicted in Fig. 3.2 (next page) – explicitly incorporates an "experimental model," derived from the "programmed model," which is used to obtain the final "simulation results" (Balci 1998, 44). The other life-cycle model is a pre-HLA ancestor to the Overall

Problem Solving Process presented in the MSCO’s VV&A RPG; it is interesting to note that this earlier DoD M&S life-cycle (Balci 1998, 43) still explicitly calls for planning of VV&A to begin at an early stage of the development process, while the separate life-cycle developed by Balci instead places emphasis on the view of V&V as Verification, Validation, and Testing (VV&T) performed throughout the model life-cycle, where testing explicitly refers to the individual activities needed to verify and validate various aspects of the model (Balci 1998, 41).

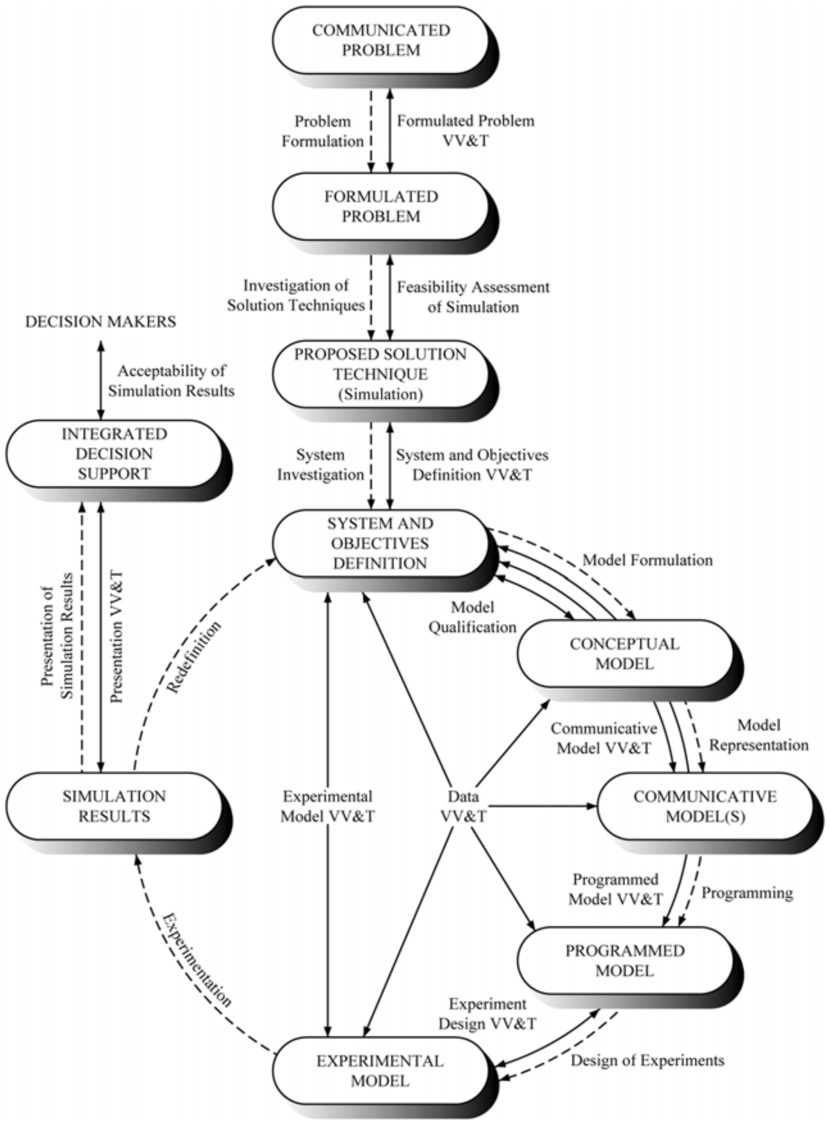

Fig. 3.2. “VV&A in Another M&S Life Cycle” (Balci 1998, 44)

The DoD’s M&S life-cycle has undergone a number of changes during the 10 years separating the version described by Balci and the one presented in build 3.0 of the MSCO’s VV&A RPG; in particular, one observes that many of the individual steps of V&V presented in the older version (including “V&V Conceptual Model,” “V&V Design,” “V&V Implementation,” and “V&V M&S Application”) (Balci 1998, 43) have been consolidated into “Perform V&V Activities Appropriate for M&S Category,” likely because the older approach was not flexible enough to describe the activities needed for VV&A of federations or legacy simulation models. One also finds that the word “test” is conspicuously absent from the older DoD M&S life-cycle; presumably, software testing was intended to be implied either as part of the software engineering process underling the implementation of the model, or as part of verification and validation of that implementation. In contrast, Balci’s model development cycle (Fig. 3.2) presents testing as a fundamental component of simulation model V&V, while the DoD’s most recent M&S development process, depicted in Fig. 3.1, presents testing of the final model or federation as a separate activity from the V&V Process. One can observe that both Balci’s M&S life-cycle, and the older version of the DoD’s M&S life-cycle, contain arrows explicitly indicating that individual stages of the development process can be reiterated to address “deficiencies identified by a VV&A activity” (Balci 1998, 42-44). It is possible that such arrows were simply left out of the Overall Problem Solving Process in the VV&A RPG for aesthetic reasons, since the guide does note in several places that it may be necessary to reiterate portions of the development process if problems are encountered (MSCO 2006). The model development process, and resulting perspective on VV&A, offered by the RPG can be considered to be relatively specialized, since it is both directed at a specific audience – the defense community – and built around the conceptual framework of HLA. The decision to depict model development V&V, and accreditation as three separate processes conducted in parallel also reflects the separation of the roles of “Developer,” “V&V Agent,” and “Accreditation Agent” described in the RPG (MSCO, 2006). By comparison, the life-cycle constructed by Balci by can considered relatively generic, as it describes stages of simulation model development and accompanying activities in a manner which is not expressly dictated by the needs and policies of a particular organization or community.

One can also consider the particular techniques and approaches that can be employed to achieve verification and validation of a simulation model. Following Balci (1998)'s description of the pre-HLA DoD M&S life-cycle and Balci's own M&S life-cycle, the author lists 77 specific activities that can be used in V&V, and indicates the steps of each model development cycle in which each activity can be considered applicable. Sargent (2004) discusses a number of ways of envisioning and conducting simulation model validation, including a variety of strategies for testing and gathering data on the implemented simulation model. Sargent (2004, 22-23) also briefly describes how one can approach the task of verification of the software implementation of a simulation model, by applying various static or dynamic testing techniques. There are also works treating verification and validation in a distinctively domain-specific context; for example, (Holzmann 1991) describes validation from a different perspective, in terms of the use of a variety of formal modeling techniques for validating the behavior of a computer protocol. One can consider, as Harmon and Youngblood (2005) do, the evaluation of a verification or validation process as a whole. In (Harmon and Youngblood 2005), a model is given for describing the maturity of a process for validating simulation models. A requisite component of the highest level of maturity proposed by Harmon and Youngblood (2005) is the usage of a "formal proof" of a model's validity, in place of a subjective evaluation by a subject matter expert or an independent observer. While efforts have been made to allow a degree of automated testing of simulation models in certain domains – such as the Verisim tool for analyzing simulations of computer network protocols (Bhargavan et al 2002) – Harmon and Youngblood (2005) themselves acknowledge that the mathematical underpinnings required to support formal proofs of simulation validation have not yet been sufficiently developed.

One thing which most of the above approaches share in common is that they primarily focus on the process of V&V implementation, describing how it should be incorporated into a practicable model development process. This can be contrasted with the methodology evidenced by Freigassner, Sarjoughian, Praehofer, and Zeigler (2001), who describe V&V within the FEDEP process primarily from the perspective of systems theory. As was mentioned in section 3.1, one concept which is prominently featured in (Freigassner et al 2001) is the "experimental frame" (EF). An EF, which is designed around the specific objectives of a modeling and simulation project, describes a conceptual approach or interface for

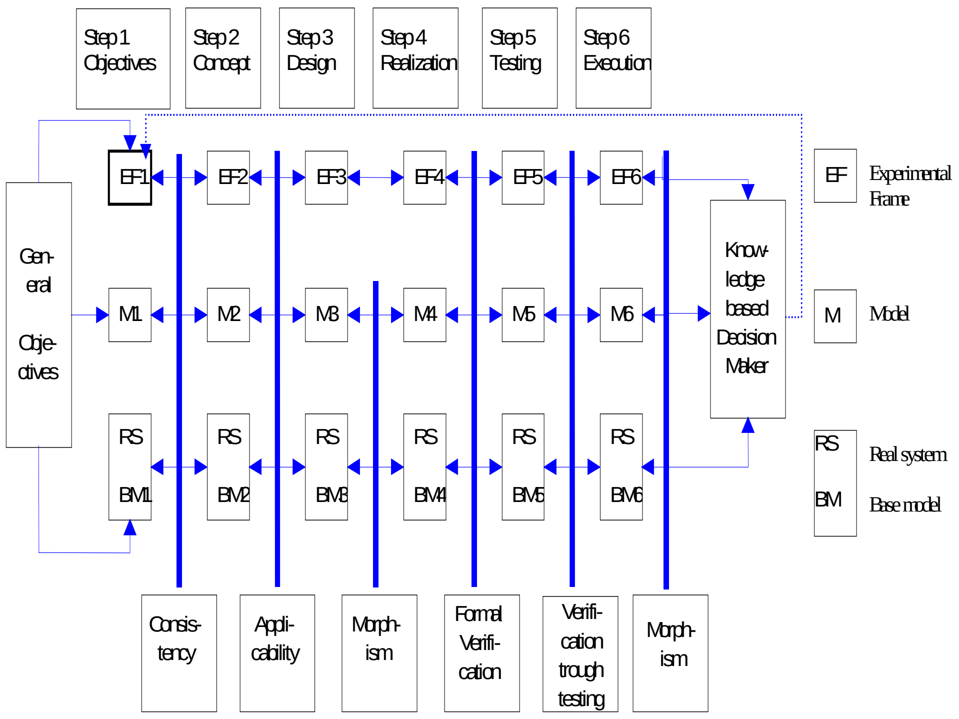

Fig. 3.3. "6-Step-Process and V&V" (Freigassner et al 2001, 2)

stimulating a simulation model or a real-world system and then observing and analyzing the resulting "output." The notion of the experimental frame provides a useful vehicle for pursuing a detailed theoretical treatment of simulation model verification and validation: in (Freigassner et al 2001), the authors describe experimental frames applicable in each step of the FEDEP process. These EFs consist of the "Requirements level" EF_1, the "Conceptual level" EF_2, the "Design/realization level" EF_3 and EF_4, and the "Parameterized experimentation level" EF_5 and EF_6 (Freigassner et al 2001, 3). These EFs correspond to a series of 6 "Model Specifications," one for each step of FEDEP: these are described as the "Observation system" M_1, the "IORO [(Input / Output Relation Observation)] and IOFO [(Input / Output Function Observation)] systems" M_2, the "IO [(Input / Output)] system" M_3, the "Coupled system" M_4, and the "Parameterized coupled system[s]" M_5 and M_6 (Freigassner et al 2001, 3). Freigassner et al (2001) describe a "$FEDEP_{EF}$" process, in which they explicitly dictate that each step of FEDEP development should be accompanied by V&V with respect to the corresponding EF. In Fig. 3.3 (above), the overall $FEDEP_{EF}$ process is shown (Freigassner et al 2001, 2); as with the design processes presented in Balci (1998), this figure uses bi-directional arrows to emphasize the potentially iterative nature of model development. At the bottom of Fig.

3.3, six boxes are shown, depicting the six steps of V&V which Freigassner et al (2001, 2-3) describe: Consistency, Applicability, Morphism, Formal Verification, Verification through testing, and Morphism.

3.4 Scope of V&V Addressed in this Thesis

The perspectives on V&V presented in section 3.3 cover a range of practical and theoretical concerns, and address V&V as a component of a general or somewhat specialized model development process, as a separate process unto itself, and as a series of specific activities or techniques that one can employ. However, none of these directly address the concept of V&V of models developed using domain-specific tools. While (Balci 1998, 45-57) offers dozens of techniques for verifying and validating a simulation model, one finds that the ability to implement each technique depends heavily on the degree of access one has to both the software implementation of the model and its underlying design and code. In the M&S life cycles that have been presented, there is also little, if any, consideration of when and how a pre-built M&S tool – particularly one which provides a heavily-restrictive environment – can impact the development cycle and V&V process. For example, without access to the source code of a model and its simulator, one cannot effectively employ any of the static testing techniques for verification described by Sargent (2004, 22) – all of which rely on reviewing the code that will be used in the simulation – and one may also be limited in options for employing dynamic testing. If, due to the complexity or restrictiveness of the M&S tool, one is unable to sufficiently understand the attributes and behaviors of the implemented model, one may be unable to meaningfully perform validation in excess of the "Subjective (Level 1)" process maturity rating described by Harmon and Youngblood (2005, 186-187).

These and other effects which a domain-specific tool can have on model development and V&V will be made the central focus of this thesis. In particular, attention will be paid to how the two specific tools considered, IT Guru and the INET Framework, can prove beneficial or detrimental to the ease of model design, the ease of model implementation, and, above all, the ease and feasibility of simulation model verification and validation. This thesis will not attempt to match the breadth of material provided by authors such as (Balci 1998) and (Sargent 2004) for implementing verification and validation. Instead, the ability to acquire knowledge that is necessary or useful in verifying and validating a simulation model will be a key topic of discussion. This thesis will also seek to illustrate how this ability impacts the degree of confidence that can be obtained regarding a simulation model and its results. At the end of each of the next two sections, we will discuss some of the specific aspects and characteristics of IT Guru and INET which are likely to affect the ability to verify and validate simulation models developed in each tool.

4 OPNET IT GURU

IT Guru is a commercially-developed, closed source environment for modeling and simulating computer networks. It includes a number of prebuilt models of generic and vendor-specific nodes, such as routers, workstations, servers, and hubs, as well as a variety of links. Beneath these user-accessible models displayed by the program's GUI, there is code which represents the behaviors of various protocols, such as the elements of the TCP/IP stack, as well as the functionality of various hardware elements of each node. Licenses for additional model libraries are also available through OPNET. Along with IT Guru Network Planner, OPNET offers IT Guru Academic Edition (OPNET Technologies n.d. a), a version of the software with limited functionality. Since many of the principles behind development and V&V of models created in either product are essentially interchangeable, both products will be referred to as "IT Guru" throughout rest of this thesis, except when a distinction between them is clearly warranted.

The company also offers the OPNET Modeler (OPNET Technologies n.d. c) product, which they claim provides both "the functionality of IT Guru," as well as "source code for protocol and technology models" (OPNET Technologies n.d. d). However, this does not guarantee that the inner workings of the platform will be completely exposed to the model developer: in particular, no claim is made that the source code for the simulation engine itself is included, nor does OPNET's statement necessarily imply that the entire code base for every model – including models which depict applications employed by end-users of a computer network – will be available. The fact that Modeler supposedly includes a proper superset of the features available in IT Guru would allow one to reasonably infer that Modeler is likely be priced as a higher-tier product than IT Guru, and the financial investment required by each tool would have to be considered when determining which one is more practical to employ for a given project. At present, no version of Modeler is available under licensing terms comparable to those of IT Guru Academic Edition; as a result, the Modeler software fell outside the budgeting constraints of the research underlying this work.

IT Guru was ultimately chosen to be part of this research effort due to its own standing as a widely-used (Wong, Dalmadge, and Fiedler, 2007, 5) computer network systems modeling and simulation environment, the availability of IT Guru Academic Edition, and the significant contrast which IT Guru presents relative to the open-sourced INET Framework. This last aspect was helpful in allowing for a comparison which could better reflect the wide range of tools available for this domain. When a model developer decides to employ IT Guru, rather than Modeler, it results in the developer being unable to directly acquire any of the source code which is normally provided by Modeler; it is from this perspective that IT Guru is being considered as an effectively “closed source” environment. Similarly, although further functionality can be added to IT Guru by purchasing additional products or modules (which are not even available for purchase in the Academic Edition), only the basic functionality offered by IT Guru itself will be considered in this thesis proposal.

4.1 Model Development in IT Guru

Model construction in IT Guru is entirely GUI-based. Users can place prebuilt models of nodes, links, and various “utilities” onto a grid depicting the environment, view exposed attributes of those models in the form of tables of editable values, and apply certain changes to groups of models and/or attributes via predefined commands offered in a drop down menu. Top avoid confusion, we will refer to these individual user-selectable models as “components” of the larger model of the computer network system. One can characterize the process of developing and implementing a typical computer network model under IT Guru in terms of a series of tasks, which do not always have to be accomplished in a completely fixed order: defining the network topology, describing the individual nodes and links, describing the “background” traffic load of the network, describing the “foreground” software applications of interest in the network, and assigning roles in software applications to individual nodes. The layout of the GUI, the set of commands and wizards provided by the tool, and the documentation which accompanies the tool appear to reinforce these general abstractions for describing the design process. In this section, we have chosen to group the majority of these tasks – all tasks besides network topology construction and node and link configuration – into a single subsection, referred to as “Modeling Applications and Traffic Loads,” to allow for a more direct comparison with the less-structured model development process in INET.

4.1.1 Network Topology Construction

By default, the topology of the network evolves as the user drags-and-drops nodes and links into the main editing window. Commands are available to automatically organize nodes and associated links into a handful of common topologies, such as a star or a bus. Groups of nodes can be explicitly organized into subnets, which can be nested up to a (nearly) arbitrary depth to create complex hierarchies within the network's topology. Each subnet (as well as the top-level network) can be assigned units of distance (feet, meters, kilometers, miles, or degrees) which determine the meanings of the two-dimensional coordinates assigned to objects placed within that subnet. One additional choice of units, "logical," is also allowed: logical coordinates determine only the visual layout of the objects within the GUI, while the distance between any pair of components residing in the domain of a common logical coordinate system is effectively modeled as being 0. The transmission delay across a link connecting a pair of nodes can be defined in terms of the distance between those nodes, or it can be set to an arbitrary length of time. When a link is added between a pair of nodes, IT Guru automatically binds that link to the next empty interface of the appropriate type in each endpoint node; this choice of interface can be manually changed if needed. If an appropriate interface is not available, the link will be added to the network model, but it will be disconnected at the end(s) where it failed to bind to an interface.

4.1.2 Node and Link Configuration

A number of attributes can be configured for individual components within the network model. These settings can impact the behavior of the component during simulation execution, or merely affect the visual representation of the component. Some attributes, such as the specific IP address assigned to a given interface, have obvious counterparts in the configuration settings available in real-world network systems, while many others, such as the transmission delay of a given link, relate to (somewhat) fixed properties of objects in real-world systems. Attribute values can be used to control protocol configurations, the performance characteristics of a node or link, and the deployment of models of software applications. The hardware elements of certain nodes (other than their network interfaces) can be modeled in one of two modes, depending on how the node is configured. The first of these modes is "simple cpu," in which the only additional hardware resource of the node which gets modeled is its cpu; activities such as routing packets occupy the cpu for a given length of time during simulation, and contention between multiple actions which each require access to the cpu can result in some of those actions being delayed. The second mode is "Detailed Server," in which RAM and disk storage are also modeled.

4.1.3 Modeling Applications and Traffic Loads

A number of techniques can be used in IT Guru to introduce network traffic during a simulation without explicitly modeling the applications generating the traffic. IT Guru allows some of the effects of network traffic to be depicted by configuring a constant background load on some of the resources of the various nodes and links, or by directly modeling the effects of steady flows of packets between pairs of endpoint systems. Such flows can also be configured to generate a certain amount of individual packets during simulation, producing what OPNET terms as "mixed" traffic modeling – in which some of the traffic load is modeled "discretely" as individual packets, while the rest of the traffic load is represented by a background load placed on the relevant resources. In general, "discretely" modeled traffic more closely represents the behavior of a real-world system, while modeling traffic in terms of resource consumption can reduce simulation runtime.

Explicit modeling of application software is primarily based on IT Guru's system of Profiles, Applications, Tasks, and Phases. An Application can be declared to be one of the eight prebuilt types – Database, Email, Ftp, Http, Print, Remote Login, Video Conferencing, or Voice – each of which can be parameterized to a limited extent by the model developer, or it can be a Custom application, which is itself comprised of a list of Tasks. A Task, in turn, consists of a series of Phases. A Phase, the most basic of these components, is defined in terms of the "symbolic names" of the two nodes which participate in the Phase, the volume and distribution of the traffic between the two participants, the Phase's starting and terminating conditions, and the amount of processing time required for each participant at various stages of the Phase. In addition, one can choose a "Job" which a given participant should run as part of the Phase; when the "Detailed Server" mode is used on that participant, the specified Job will be run by that participant as part of the Phase; Jobs, unlike Phases, can be explicitly defined in terms of their usage of resources such as RAM and disk storage. Multiple Phases can be run simultaneously (although they must still contend for available resources), or dependencies between Phases can be used to produce a partially- or completely-sequential order of execution.

To actually deploy a software application model on a node, it is necessary to first create a Profile; a Profile consists of a list of Applications, which can be run in parallel, or in a fixed or random sequential order. A Profile is intended to represent the behavior of a given user or group of users, depicting the pattern in which they employ a given set of software Applications. If an Application used in a Profile contains multiple participants (which will be the case for any Application that produces traffic), the nodes must be configured so that the symbolic names are mapped to the appropriate node(s) representing each participant. In general, a Profile is only deployed on a node when that node plays an active role in the associated Applications – that is, the node can begin executing an Application without any direct dependency on the actions of other participants. Nodes which play passive roles – relying on another node to commence execution of the Application – still have their behaviors dictated by the role which they play in the Application, but the Application is never explicitly modeled as part of the node from the perspective of the model developer. One consequence of bundling client- and server-side application behavior together in this fashion is that it is effectively impossible to decouple these behaviors; an Application must be designed, created, verified and validated, maintained, and reused as a single unit. In fact, IT Guru cannot really be said to have a single model which is directly analogous to a conventional process or thread, aside from the Job – and Jobs in IT Guru are incapable of sending or receiving network traffic.

4.2 Impact on Verification and Validation

Since IT Guru is a closed source environment, and since model construction takes place at such a high level of abstraction, it can be difficult, if not impossible, for a developer to determine the composition of a given pre-built component. Perhaps more significantly, one cannot expect to reliably identify the relationships between the individual components or among their various sub-components. Without access to this knowledge, verification of the network model is largely limited to reviewing the topology of the network and the individual attribute values of the various nodes; any predictions made regarding the behaviors that will be exhibited by the model during simulation are effectively little more than conjectures. This situation also hampers the usefulness of validation, since one cannot expect to be able to properly interpret the results of a simulation without an understanding of the behaviors of – and interactions between – the individual components within the model. It should be emphasized that these concerns do not depend on the “correctness” of the IT Guru components themselves; even if the components behave precisely as OPNET’s developers intended, and even if they are somehow capable of accounting for even the most trivial of behaviors exhibited by real-world network systems, this does not mean that the structure and behaviors of the individual components must be identical to their real-world counterparts. One must understand how the components themselves behave, how they should be configured and composed together into a model of a network system, in order to make any definitive assertions regarding one's own use of those components.

As with software testing, verification and validation is frequently a multistage process, in which one considers not only the product as a whole, but the individual pieces which are used to form that product. The closed-source nature of IT Guru clearly makes it impossible to directly inspect the inner-workings of individual components, but V&V of individual components is also inhibited by the tightly-coupled nature of many of those components. For example, one cannot view or test the sub-components of a node by themselves; a node must effectively be considered as single, monolithic object. Given the purportedly detailed nature of IT Guru's depictions of network protocols, routing algorithms, and a range of hardware devices, an individual node seems like a poor level of abstraction to use for thoroughly verifying and validating a computer network model developed in IT Guru. Nevertheless, this is the perspective which is forced upon the model developer by this particular tool. V&V of software application models is also constrained by tight coupling between components. It has already been noted that one cannot isolate the individual software applications run by each participant in an IT Guru Application instance, which has the potential to cause problems throughout all phases of the development process. However, it is also impossible to stimulate or observe an IT Guru Application or protocol model during a simulation except by means of the surrounding node. This means that, fundamentally, one cannot truly validate a software application or protocol model in IT Guru directly.

IT Guru is helpful in streamlining the V&V process, even though it limits the potential depth and thoroughness of that process. The design of the GUI makes it easy to compare the configurations of individual components against ones design of the model. In particular, IT Guru allows the user to compose queries with respect to a computer network model, defined in terms of component types, attributes, and attribute values, which provides a powerful and flexible method of identifying, comparing, and changing configuration settings, and of inspecting a desired aspect of the network model. Validation is also assisted by the interface supplied for selecting statistics, which are values, computed based on the states and events observed during simulation execution, that describe or characterize individual aspects of the simulation model’s behavior. IT Guru offers a wide selection of built-in statistics to choose from, relating to a number of aggregate measurements that can be taken regarding the behavior of a real-world network, node, link, or type of application. Statistics can be easily chosen for individual nodes, or for the network as a whole, and the user is allowed to determine the frequency with which measurements are taken, as well as the method of aggregation which is applied to the data. IT Guru will additionally compute certain statistical properties of the collected values, allow the user to view graphs of the values collected for one of more statistics across one or more simulation runs, and allow the user to export the values depicted in a graph into a csv (comma separated values) file for use in third-party applications such as Microsoft Excel.

5 INET FRAMEWORK

The INET Framework, in contrast with IT Guru, is a library of open-source models, built for use with the open-source OMNET++ modeling and simulation environment. Although the complete source code for both INET and OMNET++ is made available to the user, they cannot be considered as “free software” in the sense defined by licenses such as the widely-used GPL (Free Software Foundation n.d.): in particular, OMNET++ cannot be used for commercial purposes without paying for a license (OMNET++ Community Site n.d. b), although the license does grant the user the ability to view and edit all source code (Simulcraft n.d.). As with IT Guru, INET contains detailed prebuilt models of a number of protocols and devices, although it does not match the wide range of protocols, interface types, and hardware components which IT Guru offers; nor does it contain any models of vendor-specific devices. However, INET and similar open-source tools provide several advantages over an environment like IT Guru. With access to the complete source code of each model, the developer can freely investigate the structure and behavior the model, which allows them to understand the exact relationships among the various components. Using an open source model library also means that the developer can easily add new models to the framework, and even make changes to the prebuilt models themselves. This second course of action should not be taken lightly, however, as it can significantly increase the workload required during Verification and Validation, as well as impairing the maintainability and reusability of the model – since the model now relies on a non-standard version of the underlying tool, which must be maintained independently by the model developer.

The design of the INET Framework, and of any network system model built using the components of the framework, must inevitably be influenced by the nature of the OMNET++ simulator. Unlike the precompiled “op_runsim” executable used by IT Guru, the inner-workings of the OMNET++ simulator are completely exposed to the model developer. OMNET++ is a discrete event simulator, intended for simulating models of “distributed or parallel systems” (Varga 2001). Although OMNET++ contains a number of features that make simulating such systems convenient, it can, in theory, be used as a general-purpose simulation environment: the sending and receiving of messages and self-messages (including the ability to cancel and reschedule those messages); the organization of models into “simple modules,” which have no simulatable submodules, and “compound modules,” which can be nested up to an arbitrary depth, yet which must always ultimately decompose into one or more simple modules; and the Finite State

Machine mechanism provide virtually all that one needs to build models in OMNET++ according to roughly the same principles employed during model construction in the DEVS-Suite environment (Varga 2005). It should be noted, however, that OMNET++ does not enforce the same set of constraints which DEVS-Suite applies regarding model behavior and composition. This stands in stark contrast with the IT Guru environment, where the fundamental nature of the op_runsim simulation executable is largely hidden from the user; even if the model developer were allowed to code new models for IT Guru, there would be no way to verify that the behavior implemented by the simulator would match the developer's expectations.

5.1 Model Development with INET

The production version of OMNET++ 4.0, recently released at the time of this writing, includes an IDE based on Eclipse, which adds a degree of automation to the development process. However, the research conducted for this thesis is based on version 3.3p1, which was the most recent stable version during the majority of the time in which the research was conducted. As the IDE is a recent addition in version 4.0, it will not be discussed in this thesis. Regardless of whether or not the IDE is used, creating network system models in OMNET++ – and, by extension, creating models based on the INET Framework – remains primarily a process of editing source code and configuration files.

5.1.1 Network Topology Construction

The topology of a network in OMNET++ is expressed via NED files (text files with .ned extensions), which employ the eponymous NED language. NED is a reasonably simple language (the extended BNF grammar for NED provided in the user manual accompanying OMNET++ 3.3p1 spans only 155 lines, in spite of liberal use of white space) used to declare the modules (models) that will be used and how those modules are composed, as well as the connections that are made between pairs of gates belonging to individual modules. Certain mechanisms exist which help improve the flexibility of NED topologies, such as the "like" keyword (explained in (Varga 1998)) which effectively allows for a limited form of polymorphism among modules. NED also provides analogues for a few popular control flow statements, such as for loops and if statements. Flexibility is further increased by the fact that OMNET++ allows NED files to be loaded dynamically when the simulator is launched, which means that the network topology can be reconfigured without needing to recompile the source code for any of the modules, a feature which is typically not automatically provided by environments such as DEVS-SUITE. OMNET++ includes a tool called GNED (the functionality of which has since been incorporated into the OMNET++ IDE), which provides a graphical environment for constructing NED files.

5.1.2 Node and Link Configuration

In addition to defining the topology of a network (or of an individual module), NED files can be used to provide values for various parameters used by individual modules or connections, or even to declare new parameters that can receive values from other NED files or from external .ini configuration files. The syntax of .ini files allows for the use of wildcards in identifying modules, so that a given parameter value can be assigned simultaneously to multiple modules. These parameter values, once supplied by a NED or .ini file, can then be used to adjust the topology of a module (where appropriate), or can be referenced by the code contained within the module. Modules themselves are individually categorized as either simple models – which are implemented using subclasses of the cSimple class defined in the OMNET++ source code – or as compound modules, which are described entirely within NED files, without relying on user-defined code. The roles of simple and compound modules can be likened to the Atomic and Coupled Models defined by DEVS-Suite, where Coupled Models exits primarily to encapsulate and form connections between the individual models which they contain. Parameters of INET modules, as with parameters of OMNET++ modules in general, are used both to adjust the topology and composition of the modules, and to influence the behaviors of the modules and those of their sub-modules. Some parameters are available which correspond to configuration settings for real-world protocols and hardware devices, but, in general, the parameters built into the INET modules appear to permit less of this form of customization than the various attributes found in IT Guru’s prebuilt models. Configuration of connections between nodes is particularly limited, since INET merely employs the three connection parameters – propagation delay, data rate, and bit error rate – which OMNET++ itself supports by default. As a result, no explicit type-system is directly enforced by the environment with regards to links and interfaces; nor can propagation delays be implicitly defined in terms of the distance between a pair of nodes.

Due to the nature of compound modules in OMNET++, prebuilt models of nodes in the INET Framework are inherently quite simple in and of themselves, consisting of just a handful of compound and simple modules, the connections made between them, and values for some of their parameters. NED files can also be used to construct entirely new nodes (in the form of new compound modules) using preexisting modules. However, a much broader form of node "configuration" can be achieved by coding new simple modules in C++, writing NED files to accompany those modules (and to depict any new compound modules that should encapsulate those simple modules), and then incorporating the resulting modules into a new or existing node module. One must bear in mind, however, that any new modules which are developed will have to be carefully designed and tested. Not only will the modules themselves have to be individually verified and validated, but they must also be V&V-ed in terms of their interactions with the various components of the INET Framework.

5.1.3 Modeling Applications and Traffic Loads

The INET Framework presents a significantly different environment for developing models of software applications and network traffic than that of IT Guru. Instead of confining the user to a specific interface for constructing models of applications, with a fixed set of possible conditions, interactions, and conventions for describing application behavior, INET allows application behavior to be modeled in any manner which is consistent with the tenets of the OMNET++ simulation environment, meaning that virtually any application behavior which can be expressed in terms of C++ and OMNET's discrete-event paradigm can be depicted. INET provides a handful of convenience classes and methods that can be used to provide application models with an interface that is roughly comparable to a typical API for interacting with a transport-layer protocol. This can make application models even more straightforward to produce, since the often-familiar perspective of developing a networked application can be adopted when designing an application's model and characterizing its behavior. However, this perspective can also invite a great deal of misunderstanding if the model developer loses sight of the fact that the implementation underlying the interface is nothing more than a model, whose behavior must inevitably be understood as being fundamentally distinct from that exhibited by the real-world system which it is intended to represent.

5.2 Impact on Verification and Validation

As has been mentioned repeatedly, the use of open source models in the INET Framework allows the developer to inspect every facet of the prebuilt modules; if one so desires, it would be technically possible to verify and validate the entire INET Framework, in addition to any network system models which one composes from the elements of that framework. One can even go so far as to test the underlying OMNET++ environment. This is obviously a significant difference from IT Guru, but the ability to enact this form of extensive V&V does not mean that doing so is inherently the best policy. The documentation generated by doxygen (Heesch 2009) for the INET Framework, as well as the documentation which accompanies the framework itself, indicates that, as of the October 20, 2006 release, the framework includes over 150 modules, over 540 C++ classes, structs, unions, and interfaces, and a total of more than 4,300 member functions. While testing the INET Framework would arguably be less costly and time-consuming than developing and testing a similar model library from scratch, it is by no means a small feat to undertake. Even without testing the framework directly, however, the developer at least has access to a means of obtaining the information that they require regarding the elements of the framework. Questions or concerns regarding the structure and behavior of an individual prebuilt module can be answered directly and authoritatively by referencing that module's source code. Through careful inspection of the source code, one can reconstruct the detailed relationships between the various modules, allowing one to predict the range of behaviors that a model will exhibit during simulation. As a result, the Verification process is free from many of the constraints imposed by IT Guru.

In addition to simply being open sourced, the OMNET++ environment provides a number of features which are particularly helpful during testing and V&V. The most prominent of these is undoubtedly the use of a graphical representation of the simulation, which is constructed using Tcl/Tk (Tcl Developer Site n.d.). The concept of visually representing simulation model behavior is hardly unique to OMNET++; DEVS-Suite allows the modeler to view animations depicting messages exchanged between pairs of models while the simulation is in progress, and similar animations can also be viewed in IT Guru – although, in the case of IT Guru, these animations must be viewed after-the-fact, once the simulation has completed. However, in addition to illustrating the events that take place within the system, OMNET++'s simulation GUI allows the user to inspect the state of each model as the simulation is being executed. By default, parameter values and a handful of other attributes are automatically exposed to the user, but certain macros and function calls can be inserted into a module's code to allow the user to view and (at the model developer's discretion) even change the values of a wide range of variables found in the code of the module. In effect, one can, to a limited extent, debug a simulation model without the use of third-party tools. Having such a strong visual representation of the model's structure and behavior can be tremendously useful in assessing the consequences of particular design decisions that have been made during a model's development, and it provides a very intuitive means of both understanding how a model functions, and demonstrating that functionality to others. However, it is equally important to remember that not every aspect of a model will be portrayed by this means; interesting (and sometimes unintended) behaviors can take place as a small part of a single simulation event, and can concern variables of a module which were not included in that module's GUI representation, yet there is no real way for such behaviors to be captured by the graphical representation which is provided. Visualization can also lead to misunderstandings regarding the underlying models. For instance, as has been mentioned before, there is no inherent relationship between the apparent distance between two nodes in an INET simulation model and the propagation delay between those two nodes; similar confusion can also arise from the optional independence between propagation delays and distances in IT Guru. This conflicts with the seemingly reasonable expectation which a user may have that the propagation delays between nodes should always depend on the distances between those nodes. Another area of concern is that poor placement of modules can cause some modules to be covered up by others, making it appear as if the hidden modules are not actually present.

Not everything about the INET Framework is necessarily advantageous during verfication and validation. While it is technically possible to decouple individual modules and their various components to an extent that could not be achieved in IT Guru, many of INET's modules are, at present, quite tightly coupled. As an example of this, consider the StandardHost compound module, a prebuilt module in INET which describes an individual host in a computer network. In Fig. 5.1 (next page), a screenshot of the topology of StandardHost (taken from a simulation of the built-in "FlatNet" example network) is shown;

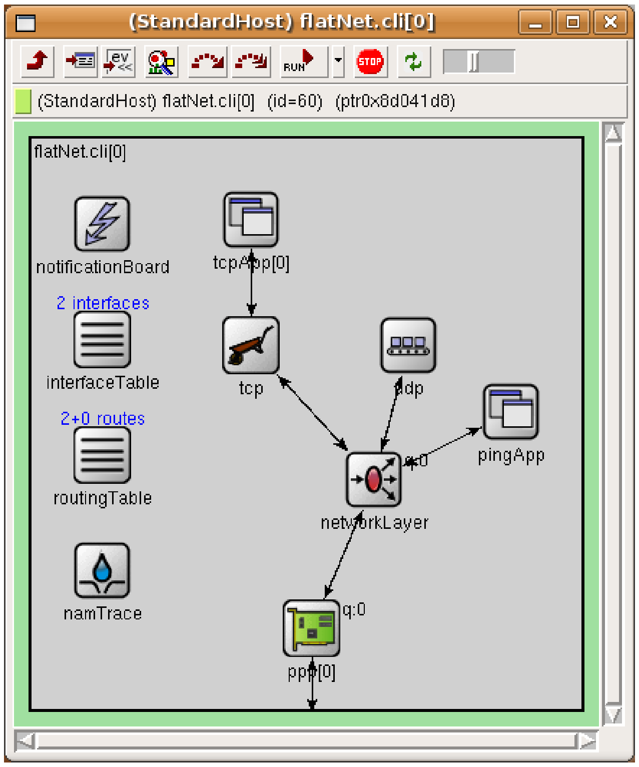

Fig. 5.1. StandardHost Module in INET's Prebuilt FlatNet Network Model

inspecting the NED file confirms that the only connections explicitly made by StandardHost are a pair of connections (one in each direction) between each TCP application or UDP application and the appropriate transport protocol module, between networkLayer and each of tcp, udp, and pingApp, between each ppp or eth interface and the networkLayer, and between each interface and the surrounding StandardHost module itself. However, during execution, one witnesses animations which explicitly demonstrate that the seemingly-isolated notificationBoard and routingTable are interacting with the other modules in StandardHost. One can also observe from this diagram that most of the hardware components of a node are not being explicitly modeled as part of the INET Framework – for instance, consider the fact that the StandardHost model does not include a module representing the node's cpu. This situation may still be considered advantageous in comparison to what one encounters when working with a tool such as IT Guru, where the hardware/software composition of the individual models is simply altogether unknown. However, it still makes it difficult to determine how data gathered from components of a given real-world system should be compared to the data obtained from an INET model, as well as to verify the model and its underlying design as a representation of one’s conceptual model of the underlying hardware and software components of the intended system.

6 EXAMPLE MODELS

As part of Operational Testing and Evaluation (OT&E) of the Net-Centric Enterprise Services (NCES) (NCES n.d. a) collaboration tools (NCES n.d. b) acquired by the Defense Information Systems Agency (DISA) (DISA n.d) in 2006 and 2007, Modeling and Simulation was used to attempt to predict the capability of the two collaboration services – termed “Button One” and “Button Two” – to meet certain performance criteria once they had been widely deployed. One product of this effort was a series of computer network system models developed in IT Guru which depicted some of the traffic sent between relatively small groups of end-users and the servers where the collaboration services would be hosted. The network system models were split into two groups, depicting what were termed the “typical case” and “worst case” scenarios of each particular set of interactions. Each such pair of “cases” was generally distinguished by the volume of background traffic (traffic not generated as part of the collaboration service) that was depicted, as well as the number of active users and – in some cases – the usage patterns of each individual user. The topology of the network system models and parameterizations of the individual nodes were informed by data gathered from the existing real-world network where the services would be deployed, and the models used to represent the collaboration software were configured based on data regarding the real-world services. To address the specific requirements which had been formulated for the collaboration services, the text-chat, video conferencing, and voice-chat functionality of each service were made the primary focuses of the modeling efforts. A team from ASU participated in efforts to perform independent verification and validation (IV&V) of the resulting simulation models. Due to the inaccessibility of detailed information regarding the components underlying the IT Guru models, it was determined that a series of models would be developed using both IT Guru and INET/OMNET++, in order to compare the models and their simulation results and to build the team’s confidence in their understanding of IT Guru and its components.

Each comparison involved a pair of models developed in each tool that sought to depict the same hypothetical computer network system. Each of these hypothetical systems was intended to represent a simplified version of a computer network system described by one of the original NCES-developed collaboration models, using a simplified topology and reduced traffic load. For each pair of models, a comparison was made between a particular set of high-level statistics obtained from simulations of each of those models. These comparisons sought to determine whether or not the two models yielded similar aggregate behaviors with respect to the chosen statistics. While such a comparison could help provide a limited degree of confidence in the results from IT Guru even if INET were a closed-source tool, it is made far more useful due to the open-source nature of INET. Since we are able to directly investigate the inner-workings of each INET module, we can determine with much more certainty whether or not the INET simulation model we have composed is consistent with our understanding of the system we are attempting to model. By demonstrating that simulations of the IT Guru model exhibit aggregate behavior comparable to that observed in simulations of a model which we have a high level of confidence in, we can much more reliably establish whether or not our IT Guru model seems to represent the system we had intended to model. There are, of course, many obvious limitations to this approach, several of which will be discussed in section 6.4.

6.1 Original IT Guru Model

The example IT Guru computer network system model which will be presented was derived from a highly simplified and scaled-down version of one of the models of the NCES collaboration services. The specific model the example is based on depicts a “Server_Farm” subnet, located in the continental US, where the Button Two services are being hosted, and a “JTF_HQTRS” (Joint Task Force headquarters) subnet, in eastern Africa, where the group of users of interest in this particular model are located (see Fig. 6.1, next page). A router in each subnet – “Jtf_rtr” in JTF_HQTRS and “Server_farm_rtr” in Server_Farm – is connected by a satellite link to a node termed the “Core_Cloud”; Core_Cloud serves to abstract away the details of the network infrastructure that lies between the two subnets. The somewhat unreliable nature of the infrastructure between the two subnets is modeled by having the Core_Cloud stochastically discard a fixed percentage of the packets that it receives, rather than forwarding those packets on to their destinations. In the simplified version of this model, all background traffic has been eliminated, and the “foreground” traffic load has been reduced to just that which corresponds to the voice-chat service. As a result, several of the nodes in the model no longer generate or receive application traffic; these nodes have been removed from the simplified model. Additionally, the number of users in the JTF has been reduced to a total of 20, split evenly between two subnets designed to encompass the participants in each collaboration meeting.

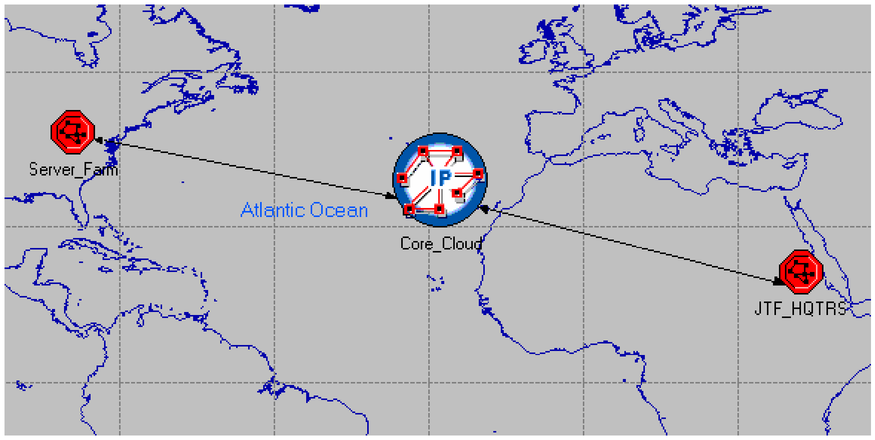

Fig. 6.1. Top-level Topology of the IT Guru Voice Chat Model

Fig. 6.2, below, depicts the logical topology of one of the meeting rooms in the simplified version of the model. Every subnet in the simplified model, besides the top-level “network” that was shown in Fig. 6.1, employs its own “logical” coordinate system, as described in section 4.1.1. One participant in each meeting, termed the “mod” (for “moderator”), generates compressed audio data which it sends to the server in the U.S., and the server then sends the audio traffic – after some amount of processing – to the remaining participants (named “rec1,” “rec2,” …, “rec9” for “receiver 1,” “receiver 2,” …, “receiver 9”) in the

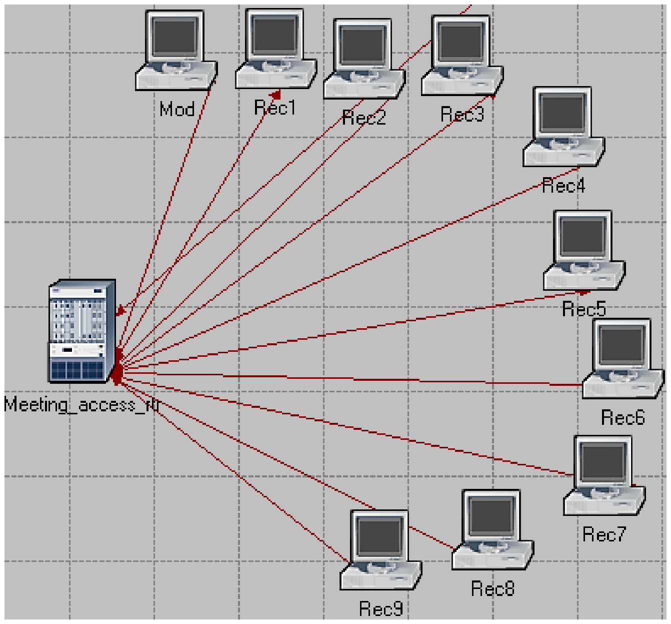

Fig. 6.2. Logical Topology of a Meeting Room in Simplified IT Guru Voice Chat Model

meeting. The server also sends a fixed-sized packet to the mod in response to each packet of audio data that it receives. All three of these streams of packets are transported via TCP. In the model, each of these three streams of packets is represented by a separate two-participant Application, which consists of a group of scripts or distributions describing the amount of application-layer data which each participant generates. Each Application sends application-layer data almost exclusively in one direction. This approach partially decouples the behaviors of the various participants, and has the advantage of making it relatively easy to characterize, test, and verify the individual Application models. However, it also prevents the model developer from representing many of the potential interactions between pairs of nodes in the real-world system, and it can lead to behaviors – such as audio data being sent from the server before the data has been generated by the mod, or even audio data being sent from the server while the mod is disconnected, disabled, or missing altogether – that are noticeably inconsistent with those exhibited by the real-world system. One also finds the method of implementing packet loss in the Core_Cloud to be potentially inconsistent with the real-world system: the probability of a given packet being dropped is defined as a fixed constant, which does not produce behavior the reflects the clustering of packet loss events that might be observed in a real world system as a result of congestion, hardware failures, or similar developments. The advantages in performance and development time that such decisions can bring must be weighed against the importance of the characteristics of the real-world system which the resulting model fails to capture, in order to decide which approach is more appropriate.

As part of this process, it is vital that one considers if the simplified simulation model is sufficient to address the requirements for the M&S effort. In the example model, for instance, the expected amount of traffic produced directly by each application model during a simulation will generally be roughly consistent with the amount of traffic produced by each application in the real-world system, but the actual sequences of interactions between application models will generally not conform to the behaviors witnessed in the real-world system. While this obviously makes the model a poor candidate if the objective is to analyze those sequences directly, problems can arise even when aggregate values of statistics regarding the system are considered. This is most readily apparent when the aggregation measures the maximum or minimum value of a statistic, but behaviors which emerge when particular constraints are or are not captured can also be significant enough to noticeably affect values derived through other forms of aggregation. Consider what happens when the Core_Cloud in this simulation model drops a packet of audio data which was sent from the mod to the server. The model in its present form will indeed be impacted by such an incident, in that the flow of packets between the mod and the server will be at least temporarily altered (although the resulting changes in the availability of bandwidth, queuing space, and other resources along the route between these nodes can impact the rest of the model to some degree). If the model enforced the “proper” ordering of interactions between the various applications, however, during this retransmission process (which lasts for a fairly significant amount of time, due to the considerable propagation delay of the satellite links used in the model), *none* of the subsequent audio data would be sent from the server to the recs in the corresponding meeting (since the TCP protocol in the server will not have delivered this data to the application residing on the server). Due to the configuration of TCP that is used in the model, the server is likely to have received a great deal of subsequent audio data from the mod once the missing data arrives. When the missing data finally makes it through, the server would attempt to send all of its available data to the receivers. Depending on the implementation and configuration of TCP at each of the hosts, and the availability of resources across the various nodes and links between the server and the recs, this could lead to a state of temporary congestion within the network, producing a number of effects (including retransmission of packets which are dropped due to the congestion) that are lost by the example model.

6.2 Model Replicated in INET

Once the IT Guru model had been formed, an attempt was made to develop a further-simplified version of this model using the INET Framework. To facilitate testing, this initial version of the model contained just a single moderator, a single receiver, the Jtr_rtr and Core_Cloud nodes, and “Fgnd_audio_server” – the audio server node. As with the original IT Guru model, the TCP protocol is used for all application-layer data generated as part of the voice-chat application. Fig. 6.3 (next page) depicts this first version of the model that was developed.

Other than discarding packets, the functionality of the original Core_Cloud node is quite similar to a typical router. For this reason, the decision was made to develop a new type of node in INET to represent the Core_Cloud, based largely on the design of INET’s prebuilt Router compound module. A new simple

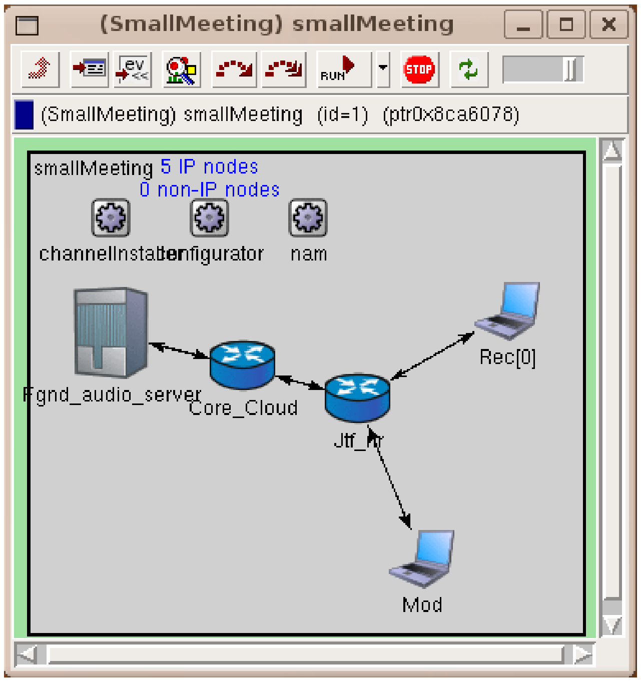

Fig. 6.3. Simplified INET Voice Chat Model

module, the PacketDiscarder, was also created; PacketDiscarder has the exceedingly simple behavior of randomly dropping a fixed percentage of the messages it receives, while forwarding all remaining messages through the appropriate output gates. The resulting model of Core_Cloud, whose type was given the name "LossyRouter," is depicted in Fig. 6.4 (next page).

Modeling the various hosts in the network was much more complicated. Since it was not feasible to definitively determine the relationships between the various node models and the utilities where the Profiles and Applications of the IT Guru model were declared, it was decided that the individual applications would be modeled exclusively as submodules of the nodes where they were being run. This decision, in itself, represents a significant simplification of the structure of the original IT Guru model; further simplification is achieved by decoupling the server and client halves of each Application, creating a total of six unique application types. Even though these choices may have increased the number of components being modeled, the resulting model proved much easier to visualize, describe, and test. Fig. 6.5 (next page) depicts the most complex of the three node models, that of the Fgnd_audio_server. Note that a module representing a cpu (as well as an accompanying "scheduler") has been added, so that contention for

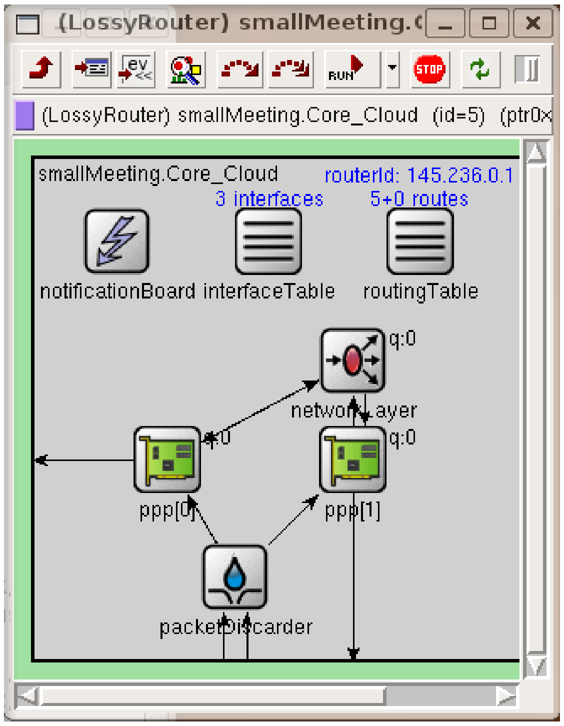

Fig. 6.4. Core_Cloud Model

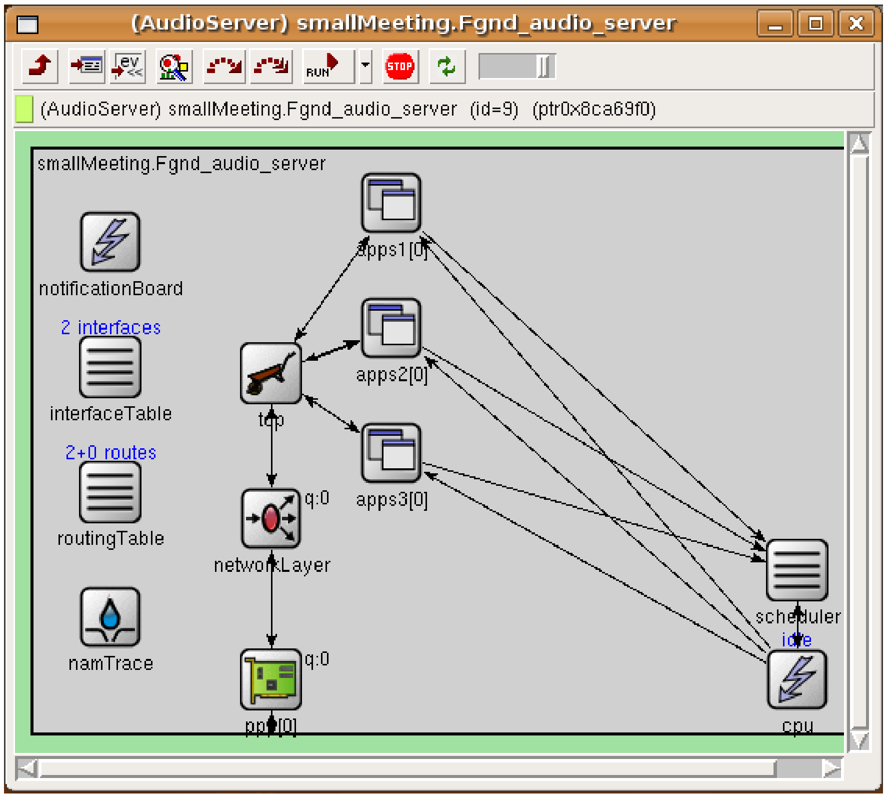
(AudioServer) smallMeeting.Fgnd_audio_server
(AudioServer) smallMeeting.Fgnd_audio_server (id=9) (ptr0x8ca69f0)
smallMeeting.Fgnd_audio_server
notificationBoard
2 interfaces
interfaceTable
2+0 routes
routingTable
namTrace
apps1[0]
apps2[0]
apps3[0]
tcp
q:0
networkLayer
q:0
ppp[0]
scheduler
idle
cpu

Fig. 6.5. Fgnd_audio_server Model

processing resources between the individual application instances can be modeled. A noticeable shortcoming of this design, particularly with respect to the original IT Guru model, is that the remaining components of the node, which belong to the existing INET Framework, do not employ this cpu, so the applications do not directly contend for resources against packet routing activities performed by the node.

After the components of the model depicted in Fig. 6.3 had been created and individually verified and – to the extent possible – validated, V&V of the network system model as a whole can commence. Once this process has been completed, we attempt to expand the model to the form depicted in Fig. 6.6 (next page). Note that, in order to permit even this simple degree of scalability, extra work must be performed during the design and V&V of the various modules, to ensure that they can be employed in a variety of network topologies and under a range of traffic conditions. In particular, the complexity of modeling software applications in the Fgnd_audio_server was noticeably increased by the decision to allow the server to service an arbitrary number of clients. If the module were not originally designed to permit this functionality, a number of changes would have to have been made to the module – which would have to be reevaluated following the modifications as part of V&V – before the network system model could be expanded. Expanding the model is by no means the end of the process; in general, it will be necessary to perform additional V&V on the expanded model as well, to determine if the assumptions and decisions that were made when constructing the modules are still applicable, and if the expanded model itself is an acceptable representation of the intended system.

It should be mentioned at this point that the “real-world” or “hypothetical” system depicted by a model can be an ambiguous and potentially misleading concept. In the case of the INET model shown in Fig. 6.6, the corresponding system could potentially be understood as any number of things, including the original collaboration system which the original IT Guru models were intended to depict, the example IT Guru model itself, or a hypothetical computer network which is described by the example IT Guru model. It is critical that a model developer have a clear understanding of the system which is being modeled, as well as the information which that model must be able to convey – similar to the requirements for a

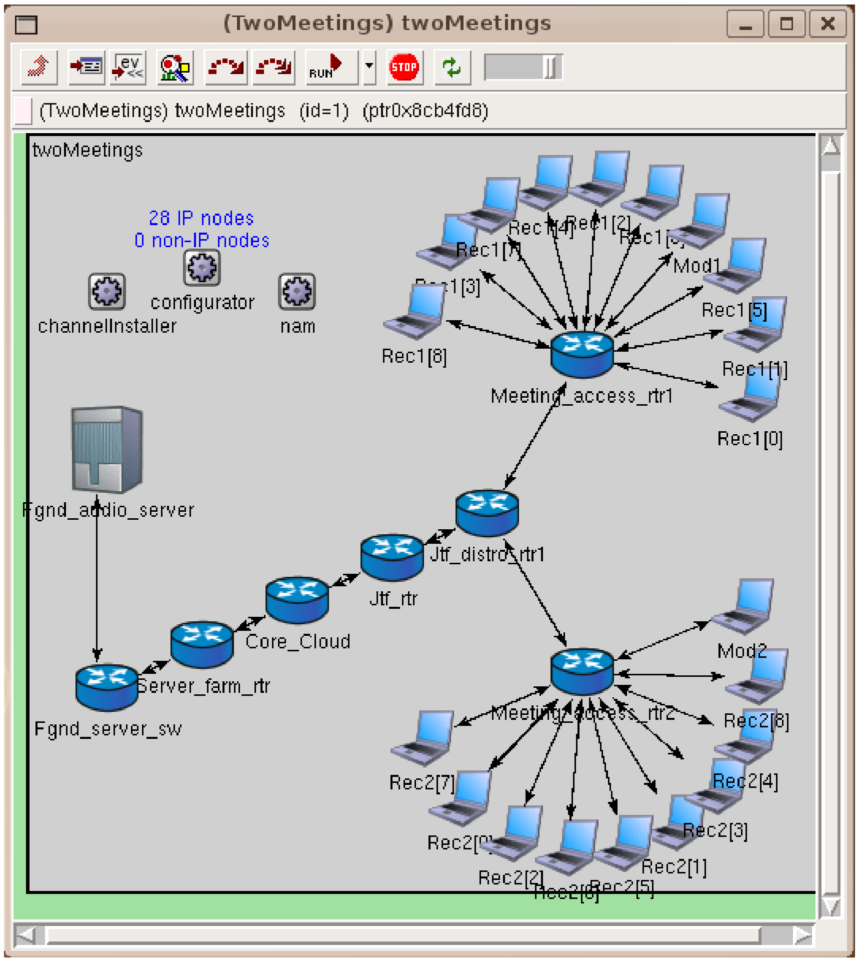

Fig. 6.6. Expanded INET Voice Chat Model

traditional software product – before beginning to design the system. In producing the example INET model, it was decided that the model should specifically aim to depict the network that was being described by the example IT Guru model. This decision influenced a number of choices made during the design of the model, as well as the scope and objectives of the verification and validation process.

6.3 V&V of the Example Models

The restrictive nature of IT Guru, and the unusual origin of the example INET model, both influenced the procedures which were employed during verification and validation of the example models. The goal of development of the INET model in particular was not to directly model a real-world system, but to establish a basis of comparison with the example IT Guru model, and to assess our team's understanding of that model. As a result, a primary consideration during verification and validation was to attempt to characterize the relationship between the two models and between their associated simulation results.

Predictable, verification of the IT Guru model was heavily constrained by the inaccessibility of knowledge regarding the components used in the model. We were unable to directly conclude if the model would capture the range of behaviors that we were expecting, if the implemented model represented the structure envisioned in our conceptual model or our design, or if the model would be able to address our requirements. We were instead forced to confine our efforts to verifying that the visible portion of the model configuration was consistent with our design, and that the behaviors which we assumed would be exhibited by the prebuilt components should be sufficient to fulfill our requirements for the model. It was the need to investigate these assumptions, and to provide some degree of assurance regarding their correctness, which motivated the development of the INET model. Validation was also inhibited by the lack of time and resources available during the IV&V efforts; it was not deemed practical to conduct new tests on a real-world system to gather data that could be applied specifically to the scaled-down model that we had developed. Instead, the validation of the scaled-down IT Guru model with respect to the hypothetical system under consideration relied in part on the relationship between the scaled-down model and the original model from which it was derived, a model which had already undergone V&V at the hands of the team originally tasked with modeling the collaboration services. Direct validation of the example IT Guru simulation model was instead primarily confined to assessing the “reasonableness” of the behaviors which the model exhibited – a far less rigorous process – together with a comparison against the behaviors produced by the INET model.

Verification of the design and implementation of the INET example model could be conducted with far more rigor. Reviewing the code of the pre-built modules, and of the handful of modules which we developed ourselves, we were able to establish to our own satisfaction that the implemented model should properly represent our conceptual model and design, and address our requirements. Validation was noticeably more cumbersome than in the case of IT Guru, due to the need to pre-process simulation data before it could be imported into a third-party tool for analysis. However, we were also freed from having to rely exclusively on the built-in statistics supplied by IT Guru. By inserting appropriate calls to the statistic recording functions built into OMNET++, we were able to define and collect new statistics that captured relationships between events that were beyond the scope of the statistics built into IT Guru.

As part of validation, we collected certain statistics with respect to each model and compared the results. One of these statistics relates to one of the performance requirements applied to the original collaboration systems, which is defined in terms of the average delay of audio data. Due to the decoupled nature of the individual applications, and the fact that the time at which each segment of audio data is generated is partially randomized, it is effectively impossible to measure the end-to-end (moderator-to-receiver) delay associated with any given segment of audio data directly, since one cannot consistently establish a meaningful and deterministic relationship between each individual segment of audio data originating at a moderator with sets of corresponding segments of audio data originating at the server. However, it is still possible to estimate the effective average delay of those segments, by simply adding together the average delay experienced by the segments of application-layer data sent from the moderators to the server, the average delay experienced by the segments of application-layer data sent from the server to the receivers, and the expected value of the average delay incurred from “processing” of audio packets at the server. Other statistics we compared included the observed packet loss rate at the Core_Cloud node, and the average rate at which application-layer traffic was sent and received at the server. Table 6.1 (next page) presents the results of this comparison for the example models, and demonstrates that while the models do exhibit differences in behavior – due in part to the specific algorithms and seed values used to generate pseudorandom numbers in each simulation environment – the effects of these differences on the specific measurement of interest (average end-to-end audio delay) is largely negligible in this case. To a rather limited degree, one can view the validation of the example INET model against the example IT Guru model as a validation of the INET model against the original real-world system via association with the IT Guru model.

Of course, it is quite conceivable that the similarity in the measured average end-to-end audio delay is due in large part to the simplicity of the two example models, and to the specific configurations which were applied to each model. The network models presented in the examples contain a total of only

Table 6.1. Simulation Results for Example Models

Measurement	IT Guru Example Model	INET Example Model
Average end-to-end delay of audio segments	0.73 seconds	0.73 seconds
Observed packet loss rate in Core_Cloud	0.00503	0.00492
Average rate of application layer traffic received by server	10,485 bytes/second	10,951 bytes/second
Average rate of application layer traffic sent by server	66,427 bytes/second	69,371 bytes/second

28 nodes, with every node besides the server participating in only one to two application instances. There is a single, unambiguous shortest path between each pair of nodes, and the average load placed on each link falls far short of the bandwidth available to that link. As a result of these and other facets of the configurations used, the above comparison does not address how the models, under different configurations, might respond to events or circumstances such as queue overflows, long term congestion, IP addressing conflicts, the existence of multiple shortest paths between a pair of nodes, dynamic changes in the topology of the network, software or hardware failures, or traffic from other sources. In effect, the validation performed is only effective for the single pair of configurations being considered. This is, in many ways, similar to what one observes when validating a model against a real-world system: the validation, in and of itself, can only provide information regarding the behaviors of the system and model under the exact circumstances in which the data used for validation were gathered. Nevertheless, the comparison was still sufficient to convince our team that, in the case of the exact models which we had created, the aggregate behaviors of the two models were similar enough for us to declare with a sufficient degree of confidence that we understood the noticeable effects of the relevant mechanisms in place within the example IT Guru model. While this does not mean that we were able to directly obtain the same level of confidence regarding our understanding of the original IT Guru models, or of the complete workings of IT Guru as a whole, it still provided a degree of assurance that had been otherwise unavailable.

6.4 Limitations of Comparisons between Example Models

As was just noted, comparisons such as those demonstrated in section 6.3 do not directly contribute any definitive knowledge regarding models or model configurations other than those used in the comparison itself. Without understanding the IT Guru model components directly, one cannot expect to determine how their behaviors will change as model topologies and configurations are altered, or as new components are introduced into the model. The comparisons that have been demonstrated also intentionally avoid analyzing the low-level behaviors of the two models directly. The decision to compare only high-level statistics was made for several reasons, including the relative amount of time needed to compare more detailed statistics, as well as the fact that the original collaboration models were likewise concerned primarily with aggregate behaviors. However, one overriding factor in this decision was that low-level statistics proved both difficult to interpret and – in the case of IT Guru Academic Edition – difficult to obtain. While IT Guru's UI made the collection of such statistics fairly easy, it significantly impaired the ability to extract and analyze those statistics on a sufficiently large scale. More troubling, however, was the difficulty in determining precisely what each statistic represented, a problem stemming in part from the lack of information regarding the internal structures and behaviors of the IT Guru mode components.

Of course, a pair of simulation models do not necessarily have to exhibit the same low-level behaviors in order to produce comparable aggregate results. In the case of the example IT Guru and INET models, for instance, one finds that while their simulations yielded very similar values for the average end-to-end audio segment delay (which was depicted in Table 6.1), there are noticeable differences in the delays of their individual segments of audio data. Fig. 6.7 and Fig. 6.8 (next page) illustrate the one-way delays (rounded to the nearest millisecond) of individual segments of audio data, recorded from 40 simulation runs of each example model, where each run lasted for 30 logical minutes; this produced a total of roughly 9 million data points for each model. It should be noted that these one-way delays differ from the end-to-end delays discussed in section 6.3, as the natures of the models prohibit measuring the end-to-end delay directly. Fig. 6.7 presents – in ascending order – the durations of the longest single one-way audio segment delay observed in each simulation run. Fig. 6.8 demonstrates how many times each length of delay is observed across the 40 simulation runs of each model, after rounding the duration of each delay to the nearest one-tenth of a second; the vertical axis (representing the number of times the given length of delay is observed) has been logarithmically scaled.

From Fig. 6.7, it is readily apparent that there seems to be a substantial difference in the maximum delays produced by each model. In fact, the average maximum delay recorded for the IT Guru simulation runs, roughly 4.37 seconds, is 56% greater than the average maximum delay recorded for INET, roughly

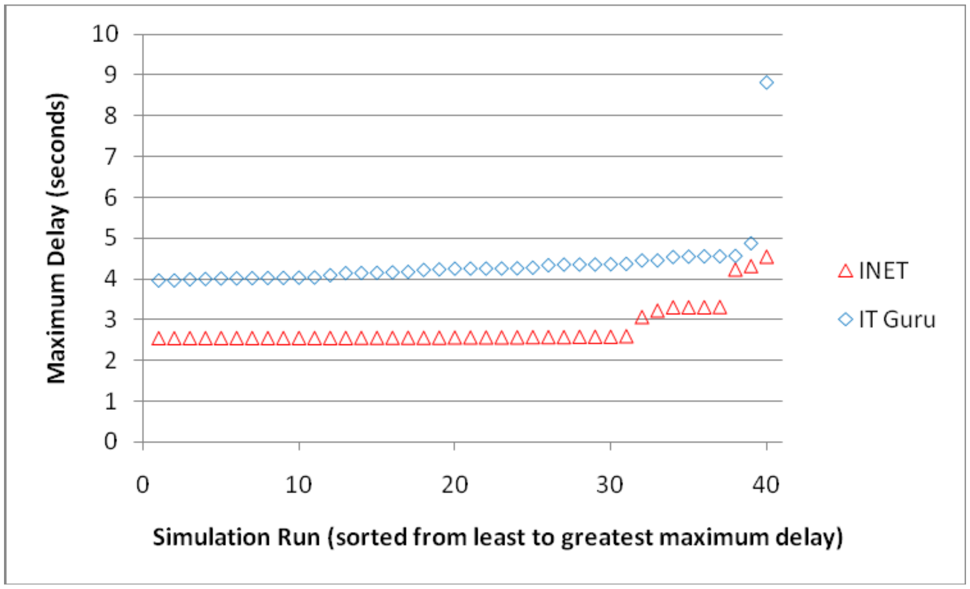

Fig. 6.7. Longest One-Way Audio Segment Delay Observed in Each Simulation Run

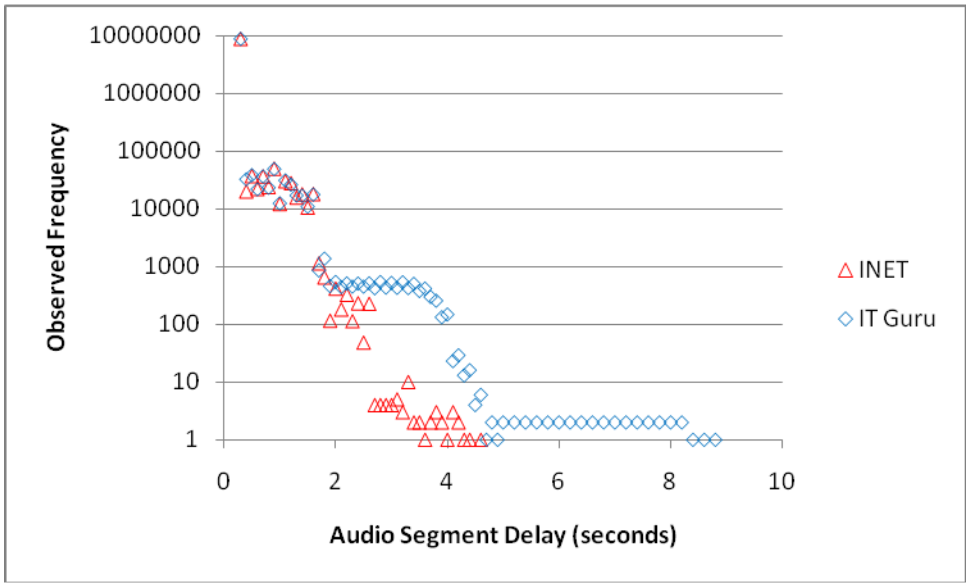

Fig. 6.8. Observed One-Way Audio Segment Delays

2.81 seconds. Of the 40 INET simulation runs, only 3 have maximum delay values that fall within the range of those observed for their 40 IT Guru counterparts. Fig. 6.8 demonstrates that delays lasting longer than roughly 1.6 seconds are produced at very different rates by the two models. However, one can also observe that the distributions of delays lasting 1.6 seconds or less are quite similar between the two models; this is particularly evident in Fig. 6.9 (below), which contains an enlarged view of the region of Fig. 6.8 spanning [0s, 1.6s] on the horizontal axis, and [10,000, 10,000,000] on the vertical axis. It is also important to realize that the short delays depicted in Fig. 6.9 (lasting from roughly 0.3seconds to 1.6 seconds) constitute the vast majority of the segment delays from Fig. 6.8 (recall again that both figures are logarithmically scaled).

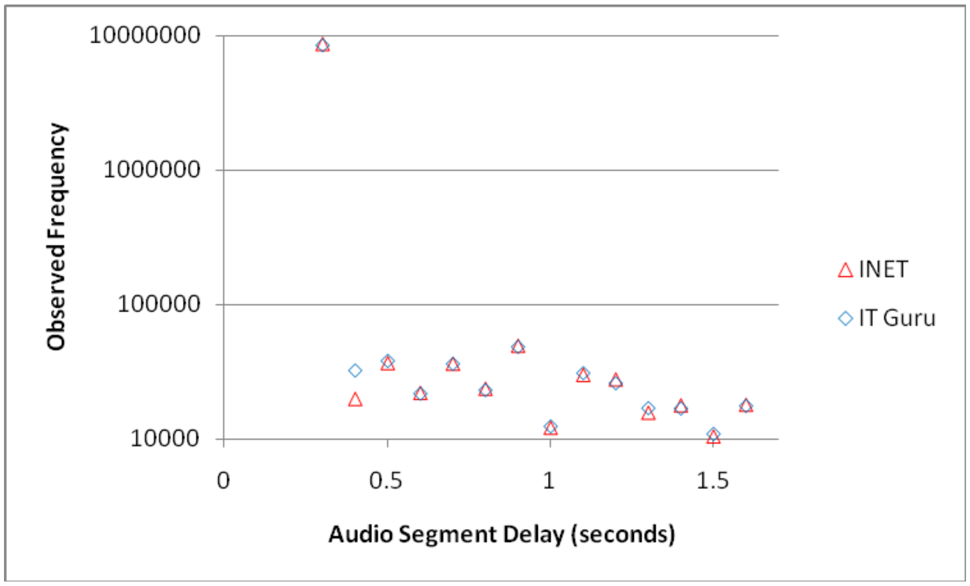

Fig. 6.9. Observed One-Way Audio Segment Delays (Enlarged)

One can conceive of many possible explanations for these observations; one obvious consideration is the slightly lower packet loss rate observed in the INET model, as was demonstrated in Table 6.1. However, it is unlikely that the relatively small difference in loss rate explains the dramatic difference in the number of delays with durations of roughly 2.7 seconds to 4.0 seconds; there were only 47 such delays observed during the INET simulations, while the IT Guru simulations produced 5,396 of them. It is also possible that the difference derives from the models of the TCP protocol used by each tool, and specifically how those models respond to retransmission timeouts. Unfortunately, since sufficient knowledge regarding the IT Guru model components is unavailable, one is realistically only able to speculate on the actual causes of these discrepancies. One must keep in mind, however, that the two example models did produce similar simulation results for each of the statistics of interest presented in Table 6.1. Ultimately, the degree to which the two models are considered "equivalent" depends entirely on the criteria used to evaluate that equivalence. Based on the findings detailed in this section, it appears as if IT Guru and INET may exhibit comparable behaviors on a scale such as that considered in Table 6.1, for models of systems similar to the hypothetical system underlying the two example models. It also seems fairly reasonable to expect the two tools to exhibit noticeably different behaviors on a scale such as that considered in Fig. 6.7 and Fig. 6.8, for models of systems similar to the aforementioned hypothetical system. In fact, due to the complexity of the systems which computer network M&S tools seek to describe, it seems extremely unlikely that any two such tools, developed independently, will produce completely identical simulated behaviors from a pair of models under all possible circumstances.

7 EVALUATION FRAMEWORK FOR DOMAIN-SPECIFIC M&S TOOLS

Due to the variety of specific advantages and liabilities which a tool can confer, the range of objectives which a modeling and simulation project may have, and the diversity of systems which one may seek to model, it is fairly clear that the decision to employ a particular M&S tool in one's project must be made in view of both the unique nature and requirements of the project, and the characteristics of the tool. The thoroughness and stringency of V&V required, the time and hardware resources available for simulation, the resources available for model design and implementation, and the scope and scale of the system being modeled can vary widely from one project to the next, and the relative worth of a tool such as IT Guru or INET depends heavily on how much one requires the specific benefits which the tool provides, and how well one can accommodate the limitations which accompany the tool. As a consequence, it can be argued that a flexible and extendable evaluation framework, which can be readily adapted to the specific needs, concerns, and priorities of the model developer, is likely to be the most applicable and appropriate means of assessing such tools. In the remainder of this section, a set of features and characteristics which a tool may possess that could heavily influence model development and the processes of verification and validation are outlined, and one possible method of evaluating these features for a pair of tools is described and demonstrated. The intent of the resulting "evaluation framework" is to allow a model developer to consider and assess certain general aspects of a tool which may significantly affect their project, so that they can then select the tool which appears best suited to address their own unique requirements. The set of features that will be described is by no means exhaustive, and is primarily intended to provide a basis for tool evaluation. By incorporating additional features into the evaluation framework, one can extend it to account for considerations that are not explicitly addressed by its present form, such as simulation run time, hardware requirements, and licensing terms.

7.1 Set of Features

Each of the following features or characteristics – referred to as f_1, f_2, …, f_6 – can be seen to provide some benefit to the model developer, by expanding the range of systems, relationships, and behaviors which can be readily described by the tool, improving the ability to obtain confidence in the model and its simulation results, or lessening the requirements for model development in terms of manpower, technical skill, time, or other resources. A description is also given of how each feature or characteristic relates to IT Guru and INET.

7.1.1 f_1: Requires Minimal Software Engineering Expertise

A tool such as the INET Framework places an added burden on the model development team by requiring a substantial degree of software engineering and programming maturity in order to properly design and implement simulation models. Model development using such tools is also encumbered by the time and manpower needed to design, code, test, and debug the required software elements. In contrast, a tool such as IT Guru does not require software engineering expertise on the part of the model development team; it is entirely possible for an individual with little to no programming experience to design and implement a model of a computer network system using IT Guru. By reducing both the technical skill and range of activities necessary for model development, a tool possessing this characteristic can both simplify the development process and reduce the personnel requirements.

7.1.2 f_2: Does not Prevent Acquisition of Knowledge Regarding Implemented Model

This characteristic is readily apparent in the case of INET; because the complete source code for the INET Framework and OMNET++ simulator are immediately available to the model developer, INET places no artificial constraints on the ability of the model developer to acquire knowledge regarding a model implemented using the tool. As has been noted before, this is extremely valuable during verification, as it allows the developer to obtain firsthand a definitive answer to virtually any question regarding the structure or behavior of a particular module, or to reconstruct or reason about the behaviors of groups of modules working in concert. This allows one to readily verify not only the implementation itself, but also to verify that the implemented design can address one’s requirements and conform to one’s conceptual model of the system. In contrast, a tool such as IT Guru, by heavily constraining the ability to obtain knowledge regarding the structure and behavior of its pre-built components – components which must, nevertheless, be used when modeling a computer network system – prevents the developer from understanding the structure and behavior of their implemented model design. This inherently limits the ability to make definitive assertions regarding the implemented model, which greatly restricts the potential thoroughness and reliability of verification.

7.1.3 f_3: Domain Knowledge Incorporated into Tool

Due to the array of complex and often rigidly defined behaviors exhibited by components of real-world computer network systems, the inclusion of pre-defined logic within the simulation tool to describe the effects of many of these behaviors can be a tremendous asset to the model developer. Both IT Guru and INET describe and enforce a number of behaviors and constraints dictated by common protocols including IPv4, TCP, UDP, and ICMP, and in doing so, they can help free the developer from the burden of researching, coding, and testing their own implementations of those same behaviors and constraints. Due to the amount of scrutiny undergone by these tools and the amount of time spent on their development, the quality and reliability of the code is also likely to be superior to what would be achieved if the model developer were to devise their own implementation of the logic described by the various protocols. INET and IT Guru further demonstrate and enforce certain relationships between elements of computer network systems, and each provides a set of “tried and true” abstractions for use in describing a computer network system. The domain knowledge incorporated into such tools can greatly reduce the time, manpower, and other resources required during model development, and can potentially improve the quality of both the final simulation model and the underlying design of that model.

However, it should be remarked that the availability of this pre-programmed domain knowledge does *not* entirely remove the need for domain knowledge on the part of the model developer. A degree of domain knowledge is obviously required simply to identify the system to be modeled and define the requirements for the modeling and simulation effort. However, when working with IT Guru or INET, further domain knowledge is called for when attempting to implement and configure the simulation model. Both IT Guru and INET support a number of configuration settings inspired by real-world systems that can cause the behavior of the simulation model to dramatically deviate from that of the system being modeled if they are not configured properly, and significant domain expertise may be needed in order to identify these settings. For example, the decision to enable or disable the use of Nagle’s algorithm – which is enabled by default in INET, and disabled by default in IT Guru – has a noticeable impact on the behavior of the two models presented in section 6, causing the average audio segment delay to more than double when it is enabled. This is due in part to the fact that the round trip time between the meeting room participants and the server is considerably longer than the time which elapses between the creation of two consecutive segments of audio data at a sending application, as well as to the value of the Maximum Segment Size configured for TCP on the various nodes, which is very large relative to the amount of traffic generated by each application, effectively preventing the transmission of data in most situations until either an acknowledgment of all outstanding data has been received or a timeout has occurred. Without a sufficient understanding of what Nagle’s algorithm describes and how it might affect the system, it would be very difficult for the model developer to anticipate and account for such behaviors. This is particularly crucial when a setting or characteristic found in the real-world system is either not supported or not well-advertised by the M&S tool.

7.1.4 f_4: Structured Model Design and Implementation Process

In working with both INET and IT Guru, one can observe that the somewhat rigidly segmented design and implementation process exhibited by IT Guru, while detrimental to flexibility, can be quite helpful in terms of productivity. By providing a defined and repeatable process with accompanying abstractions for describing and modeling a computer network system, IT Guru allows a model developer who is familiar with the tool and highly knowledgeable of the domain and system being modeled to design and construct a simulation model at a very rapid and relatively predictable pace. The framework in place for creating the network topology, describing and configuring nodes and applications, and associating applications and background traffic loads with individual nodes allow the model developer to view each modeled computer network system from the same general perspective, while the commands and wizards provided by the GUI help to speed up many of the activities involved in model implementation. In contrast, the INET Framework provides relatively little guidance on how models should be designed and constructed. While examples of individual nodes, applications, and networks are provided, one does not find a range of utilities and tutorials to assist with model development comparable to those provided by IT Guru. INET allows the model developer to make more decisions when designing a model than IT Guru, but, by the same token, it also *requires* the model developer to make more decisions. It is relatively more difficult to define a common perspective and domain-specific workflow that remains applicable to multiple modeling projects in INET, while the predictability and speed of model development can be seen to be constrained by the frequent need to rely on programming and testing of new modules or of modifications to existing ones.

7.1.5 f_5: Supports Fully User-Defined Statistics

In order to validate a simulation model as a representation of a given system, and to determine if the simulation model properly captures and represents the particular characteristics, behaviors, and aspects of the system which are of interest, it can be necessary to investigate specific and potentially complex relationships among the individual components of the model and the behaviors they exhibit. IT Guru provides a large selection of predefined statistics which can be used to obtain data on the behavior of a simulation, but it does not allow the model developer to construct statistics based on arbitrarily complex relationships. In the case of the example models presented in section 6, one might be interested in determining the maximum number of times that a single segment of audio data had to be retransmitted, measuring the average size of packets discarded by the Core_Cloud, or even in identifying the single longest period of simulation time for which every observed segment of audio data received by the server can be associated with a corresponding set of segments of processed audio data sent from the server, and then directly computing the average end-to-end audio delay over this span of time based on the resulting associations. Such measurements are quite feasible to obtain using INET, as the procedures for computing them can be incorporated directly into the code of the relevant modules, while they are all but impossible to obtain using IT Guru's pre-defined statistics. It is not strictly impossible to reproduce such statistics for IT Guru models, but doing so requires configuring the tool to log the relevant events that take place during simulation, and then using a third-party tool to compute the final value based on the logged data. Allowing the user to define such statistics directly within the tool, as in the case of INET, can help validation of the model to proceed more swiftly and dependably.

7.1.6 f_6: Does not Confine the Range of Domains which can be Modeled

Use of the INET Framework does not inherently prevent the incorporation of elements from domains besides computer networking into a model. OMNET++, while not necessarily defined in terms of a rigid formalism such as that which underlies DEVS-Suite, can still be seen to provide the necessary capabilities to function as a general-purpose discrete event simulation environment, as it nevertheless supports many of the essential capabilities of DEVS-Suite and its predecessors, including the use of primitive components which change state in response to external events (“messages) or internal events (“self-messages”), non-primitive components which create links between their contained components, a “typing” system applied to individual “ports” / “gates” that enforces a defined directionality for the flow of messages, and the use of a general purpose programming language for describing the behaviors undertaken by components in response to events. In general, any element of a system which can be defined in terms of the formalism supported by OMNET++ can therefore be incorporated into a model employing the INET framework. The availability of source code for the models and simulator means that it is also possible, at least in theory, to extend an INET simulation to support HLA, although the effort required to do so may render such an approach as impractical. IT Guru, in contrast, cannot be made to support HLA without the purchase of an additional module, and does not directly permit the development of user-defined components describing elements from other domains. As a result, an IT Guru model of a computer network system cannot be easily incorporated into a model which contains elements from other domains, which severely inhibits the potential expressiveness of models. Systems which need to be characterized in terms of detailed behaviors of users, or of external events originating in domains such as retail, finance, weather, or public utilities, can be difficult or even impossible to model using IT Guru. A tool which allows modeling to expand beyond a single domain, either by introducing new components into an existing model, or by employing the simulation model created by the tool as a federate in a larger federation, will permit modeling of a much wider range of multi-domain systems. If a tool does not support such a capability, then the model developer will have to very carefully consider whether or not the tool is even capable of describing the relevant aspects of the particular system under consideration.

7.2 Evaluation of IT Guru and INET

The immediate purpose of the list of features described in section 7.1 is to illustrate some of the issues which have to be kept in mind when evaluating a domain-specific M&S tool, regardless of the particular tool or domain under consideration. However, these features can also be used as part of a

Table 7.1. Scores Assigned to IT Guru and INET

Feature / Characteristic	IT Guru	INET
f_1: Requires Minimal Software Engineering Expertise	3	1
f_2: Does not Prevent Acquisition of Knowledge Regarding Implemented Model	1	3
f_3: Domain Knowledge Incorporated into Tool	3	3
f_4: Structured Model Design and Implementation Process	3	2
f_5: Supports Fully User-Defined Statistics	2	3
f_6: Does not Confine the Range of Domains which can be Modeled	1	2

quantitative, albeit subjective, assessment of two or more tools in a common domain. One method which could be used in achieving this is to assign each tool a score for each feature using a numeric scale such as the set of integers from 1 to 3, where the value assigned represents how well the tool demonstrates the beneficial aspects of the feature in question. A comparison of the resulting scores can be used to characterize how each tool is likely to impact the model development process, and to gauge how effective a tool is likely to be in achieving one's objectives. Once scores are assigned, a number of techniques can be used to reach a decision regarding which tool to use, such as removing from consideration all tools which fail to achieve a sufficient score for all of the features deemed necessary to one's project, and/or assigning weights to each feature and comparing the weighted total scores of each tool.

To provide an example of an assignment of scores, a brief evaluation of IT Guru and INET will be carried out. Each of these tools has already been described in terms of the features under consideration in section 7.1, so the reasoning behind the scores which are assigned will not be expanded upon in great detail. Table 7.1 (above) presents the scores which were assigned using a scale of integer values from 1 to 3.

A few of these scores are nevertheless worthy of at least some explanation. In evaluating f_3, the difference in domain knowledge supplied by the two tools was not deemed significant enough to warrant any distinction in scores. In the case of f_4, it was decided that the degree of support which INET does provide in terms of examples, documentation, and pre-defined abstractions is still sufficient to lend a noticeable amount of structure to the model development process, even though it is still noticeably less structured than development using IT Guru. For f_5, it was decided that the flexibility of the pre-defined statistics and the ability to record sequences of simulation events helped to partially make up for the lack of direct support for user-defined statistics in IT Guru. Finally, IT Guru received the minimum score for f_6 due to the fact that it both prevents the user from modeling elements of other domains, and does not support

HLA when used as a standalone project, while a single point was deducted from INET due to the difficulty involved in producing an HLA compliant simulation model. Each of these decisions is clearly subjective in nature, and the scores assigned in Table 7.1, as with the features considered, should not be considered final or authoritative. Instead, they represent the outcome of one attempt to analyze the general strengths and weaknesses of two particular modeling and simulation tools.

However, an analysis of how the scores are distributed nevertheless illustrates several key points which can be made regarding these two specific tools. One finds that INET scores better than IT Guru for the two specific features which have been identified in their descriptions as primarily influencing the processes of verification and validation, f_2 and f_5. In contrast, IT Guru scores as well as or better than INET in three remaining features which impact the resources and time required for model design and implementation, f_1, f_3, *and* f_4. Finally, judging from f_6, INET appears capable of modeling a wider range of multi-domain problems, thanks to the open nature of the OMNET++ simulation environment.

7.3 Limitations of a Domain-Neutral Evaluation Framework

In developing the framework of features presented in section 7.1, a conscious effort was made to select individual features which can be easily applied to tools in other modeling and simulation domains. This allows the framework to be used for a variety of other purposes, and also serves to restrict the scope of features and characteristics considered for inclusion into the framework. However, it also results in a framework which is unable, in its present form, to capture a range of very specific issues pertaining to the domain of computer networks, or to computer-based systems in general. One prominent example is the issue of distinguishing between the hardware and software components of a computer based system. As was previously mentioned in sections 4 and 5, neither IT Guru nor the INET Framework provides the model developer with a clear breakdown of the individual model components into analogues of hardware and software elements found in a real-world system. The decision to bundle software and hardware elements into a single primitive component is one of many simplifications that can serve to reduce the complexity of a model and improve the performance of simulations. However, this decision also represents a largely irrevocable break from the perspective of computers as primarily comprised of software components running atop hardware components. It is quite difficult to translate a conceptual model of a computer system formulated as a collection of individual hardware and software elements into a design that can be readily implemented using IT Guru or INET. It is also difficult to identify and account for the relationships and behaviors which are lost along the way. Problems can easily arise during verification and validation if one is unable to establish relationships between the various model components and the individual elements of the real-world system or of one's conceptual model of that system. One way in which this specific concern may be addressed is to explicitly model the computer-based system as a combination of individual hardware and software components. One tool which takes this approach is DEVS/DOC (Hu and Sarjoughian 2005). While DEVS/DOC does not contain the sort of extensive domain knowledge found in a tool like IT Guru or INET, it does seek to capture the concept of distinct layers of hardware and software components, and the relationships between those components, within the modeled system. The tightness of couplings between hardware and software components described by a tool is only one of many domain-specific concerns that deserve to be addressed when performing an evaluation of a tool. While the framework that is provided does not deal with such domain-specific features directly, it is still possible to add such features to the existing framework.

Along with being component-based, the two tools which were investigated are also quite clearly designed to describe a network from a system-level perspective, rather than focusing directly and exclusively on the hardware components of a single node, on an individual protocol, or on some other specific aspect of a computer network system. This is particularly apparent in the case of IT Guru, where it is effectively impossible to construct a useful model which does not contain an individual component representing a combination of both hardware and software elements of a real-world system. When one considers modeling a specific segment of computer network, or modeling another form of computer-based system entirely, one is likely to find that such a domain lies outside the range of systems easily addressed by the specialized IT Guru and INET tools. While it is believed that the six features which have been discussed may remain relevant when discussing tools designed specifically for modeling protocols, hardware, or other domains relating to computer-based systems, it is entirely possible that a different class of specific or general concerns will arise when modeling within such a domain. One may find, for instance, that the precise ordering of individual events, which may not be deemed critical when assessing the aggregate behavior of a very large system, will be crucial when testing the sequence of actions dictated by an individual protocol. It should also be understood that the type of comparison demonstrated in section 7.2 is not intended to be made between pairs of tools designed to address different domains.

8 CONCLUSION

A wealth of characteristics and criteria can be considered when attempting to evaluate a modeling and simulation tool, particularly with respect to verification and validation. While the evaluation described by this work deals directly with only two specific tools for computer network M&S, the general approach it demonstrates can equally be applied to numerous other tools in virtually any domain. Although defining an evaluation framework does not remove the subjectivity involved in defining the merits and limitations of a tool, or in selecting the best tool to use for a given project, it does provide a way of focusing one’s investigation. By identifying characteristics of a tool which one deems to be relevant, and analyzing those characteristics which one’s colleagues deem to be worthy of consideration, one can seek to devise and plan a structured approach towards evaluating M&S tools. It is hoped that the concepts and perspectives illustrated by the previously introduced framework will be able to serve as a useful springboard for the creation of such approaches in a variety of different domains.

8.1 Future Work

The framework and approach demonstrated in this thesis can be expanded upon in several ways. An examination of additional tools designed for modeling and simulating computer network systems, including tools that may not be entirely component-based, is likely to reveal additional features and characteristics that were taken for granted during the comparison between IT Guru and INET, and provide an expanded perspective that permits a fairer and more thorough evaluation of the various tools. Even without discussing additional tools, it is still possible to identify domain-specific questions and issues that remain to be addressed. The relationship between hardware and software components is one such issue that has already been raised several times throughout this thesis, and the ability of a tool to capture such relationships and convey their effects during simulation is an essential consideration when attempting to evaluate the performance or scalability of a networked application or service through M&S. One can also attempt to adapt the approach for use in evaluating M&S tools specialized towards other segment of the general domain of computer-based systems. One is likely to find that very different priorities and requirements are evidenced during development and V&V of simulation models describing microprocessor architectures, operating system kernels, social networking communities, failure rates of hardware components, or the spread of malware. Another avenue of expansion is to conduct a comparable analysis within a domain that does not primarily involve computer-based systems, to determine if the specific features identified remain relevant, or if the overall approach of analyzing tools in terms of such general characteristics is even applicable in the case of the alternative domain in question. A long-term objective of future research may be the development of a truly domain-neutral – or at least domain-adaptive – technique for evaluating M&S tools, although the general paradigm demonstrated by this thesis of collecting empirical observations regarding specific tools would require the use of a great deal of care in distinguishing between relatively universal aspects and functions of tools which arise directly from either the generic model development process or from commonalities of most modern software tools, and the relatively non-universal aspects and functions of tools that arise from the particular domain, modeling formalism, execution environment, or intended audience of the individual tool.

REFERENCES

Arizona Center for Integrative Modeling and Simulation (ACIMS). DEVS-Suite. http://www.acims.arizona.edu/SOFTWARE/software.shtml#DS (accessed April 2009).

Balci, O. (1998). Verification, Validation, and Accreditation. *Proceedings of the 30th Winter Simulation Conference*, pp. 41-48. Los Alamitos, CA.

Bhargavan, K, C. A. Gunter, M. Kim, I. Lee, D. Obradovic, O. Sokolsky, and M. Viswanathan (2002, February). Verisim: Formal Analysis of Network Simulations. *IEEE Transactions on Software Engineering 28(2)*, pp. 129-145. IEEE. http://cs.uiuc.edu/cgunter/dist/BhargavanGKLOSV02.pdf (accessed April 2009).

Dahmann, J., R. M. Fujimoto, and R. M. Weatherly (1997, December). The Department of Defense High Level Architecture. *Proceedings of the 1997 Winter Simulation Conference*. http://www.cc.gatech.edu/computing/pads/thesiss.html (accessed April 2009).

Floyd, S., and V. Paxson (2001, August). Difficulties in Simulating the Internet. *IEEE/ACM Transactions on Networking 9(4)*, pp. 392-403, IEEE/ACM.

Free Software Foundation. The GNU General Public License. http://www.gnu.org/copyleft/gpl.html (accessed April 2009).

Freigassner, R., H. S. Sarjoughian, H. Praehofer, and B. P. Zeigler (2001). A Systems Approach to a Verification and Validation Methodology within the FEDEP Six-Step-Process. *2001 European Simulation Interoperability Workshop Papers and Presentations*. http://www.sisostds.org/index.php?tg=fileman&idx=get&id=2&gr=Y&path=Simulation+Interoperability+Workshops%2F2001+EURO+SIW%2F2001+EURO+SIW+Papers+and+Presentations&file=01E-SIW-085.pdf (accessed April 2009).

Gibbs, J. D. and H. S. Sarjoughian (2009, March). Assessing the Impact of a Modeling Tool and its Support for Verification and Validation. Unpublished.

Harmon, S. Y. and S. M. Youngblood (2005, October). A Proposed Model for Simulation Validation Process Maturity. *Journal of Defense Modeling and Simulation 2(4)*, pp. 179-190.

Heesch, D. (2009, March). Doxygen. http://www.stack.nl/~dimitri/doxygen/ (accessed April 2009).

Holzmann,G. J. (1991). *Design and Validation of Computer Protocols*. Englewood Cliffs, NJ: Prentice Hall.

Hu, W. and H. S. Sarjoughian (2005). Discrete Event Simulation of Network Systems Using Distributed Object Computing. *Proceedings of the 2005 International Symposium on Performance Evaluation and Telecommunication Systems*, pp. 884-893. San Diego: Society for Modeling and Simulation International.

OMNET++ Community Site (n.d. a). INET Framework Documentation and Tools. http://www.inet.omnetpp.org/ (accessed April 2009).

OMNET++ Community Site (n.d. b). OMNET++ Community Site. http://www.omnetpp.org (accessed April 2009).

OPNET Technologies (n.d. a). IT Guru Academic Edition. http://www.opnet.com/university_program/itguru_academic_edition/ (accessed April 2009).

OPNET Technologies (n.d. b). Network Planning. http://www.opnet.com/solutions/network_planning_operations/itguru_network_planner/index.html (accessed April 2009).

OPNET Technologies (n.d. c). Network Simulation. http://www.opnet.com/solutions/network_rd/modeler.html (accessed April 2009).

OPNET Technologies (n.d. d). University Program Products. http://www.opnet.com/university_program/university_program_products/index.html (accessed April 2009).

Sarjoughian, H. S. and K. Shaukat (2008, May). A Comparative Study of DEVS and ns-2 Modeling Approaches. Internal Report. Department of Computer Science and Engineering, Arizona State University, Tempe, AZ.

Sargent, R. G. (2004). Validation and Verification of Simulation Models. *Proceedings of the 36th Winter Simulation Conference, volume 1*, pp. 13-24. Los Alamitos, CA.

Simulcraft. Simulcraft, Inc. http://www.omnest.com/ (accessed April 2009).

Tcl Developer Site. http://www.tcl.tk/ (accessed April 2009).

The Network Simulator – ns-2. http://isi.edu/nsnam/ns/ (accessed April 2009).

United States Department of Defense (US DoD) (2008, January). *Department of Defense Standard Practice: Documentation of Verification, Validation, and Accreditation (VV&A) for Models and Simulations*, 3.

United States Department of Defense, Defense Information Systems Agency (DISA). DISA – For The Public. http://www.disa.mil/audience/public.html (accessed April 2009).

United States Department of Defense, Modeling & Simulation Coordination Office (MSCO) (2006). *VV&A Recommended Practices Guide, Build 3.0*. Available: http://vva.msco.mil/.

United States Department of Defense, Net-Centric Enterprise Services (NCES) (n.d. a). Net-Centric Enterprise Services (NCES). http://www.disa.mil/nces/ (accessed April 2009).

United States Department of Defense, Net-Centric Enterprise Services (NCES) (n.d. b). NCES – Collaboration. http://www.disa.mil/nces/product_lines/collaboration.html (accessed April 2009).

University of California, Los Angeles, Mobile Systems Laboratory (MSL). About GloMoSim. http://pcl.cs.ucla.edu/projects/glomosim/ (accessed April 2009).

Varga, A. (1998). Parameterized Topologies for Simulation Programs. *Proceedings of the 1998 Western Multiconference on Simulation*.

Varga, A. (2001). The OMNET++ Discrete Event Simulation System. *Proceedings of the 2001 European Simulation Multiconference*.

Varga, A. (2005, March). *OMNET++ Discrete Event Simulation System Version 3.2 User Manual*.

Wong, R. M., C. L. Dalmadge, and A. M. Fiedler (2007, January). Managing eBusiness Continuity: Formulating an Appropriate Strategy to Manage System Scalability. *Hawaii International Conference on System Sciences – 40, Proceedings*. IEEE.

Zeigler, B. P., H. Praehofer, and T. G. Kim (2000). Theory of Modeling and Simulation, 2nd ed. San Diego, CA: Academic Press.