A System for Experimental Research in Distributed Survivability Architectures[1]

John Knight, Robert Schutt, Kevin Sullivan
University of Virginia Department of Computer Science
Thornton Hall, Charlottesville VA 22903
Phone: (804) 982-2216, Fax: (804) 982-2214
E-mail: (knight | rschutt | sullivan)@Virginia.EDU

Abstract: The survivability of critical infrastructures is not assured today because of their reliance on complex, vulnerable information systems. Survivability enhancement will require changes to system architectures. Experimental systems research in this area is complicated by the private ownership, criticality, and complexity of infrastructure systems. A key research method is therefore to explore and evaluate novel architectural concepts by prototyping them within dynamic models of infrastructure systems. We present a toolkit to support the construction of such models and the prototype-based evaluation of novel architectural idioms. The toolkit captures common features of infrastructure systems that challenge the state of the art in survivability management. Our models and prototype constructs run on workstations running Windows 2000 and function as distributed systems of programmable, message-passing processes with reflective capabilities. We have built infrastructure models with up to 20,000 processing nodes, and we have evaluated several novel architectures, including one based on distributed hierarchical control.

Keywords: Survivability, testbeds, distributed systems, development environments.

1 Introduction

The survivability of critical, information-intensive infrastructure systems, such as electric power generation and control, banking and financial systems, telecommunications, and air traffic control systems, has emerged as a major national concern [16, 22]. By survivability, we mean the ability of such systems to continue to provide acceptable levels of service under predefined adverse circumstances [11]. *Acceptable* service is defined in terms of the aggregated impact of service disturbance to users who depend on the system. Such service might be a degraded form of normal service or an alternate service, differing in whole or in part from normal service, as determined by application experts and policy makers [20]. *Adverse* circumstances might be widespread environmental damage, major equipment failures, coordinated security attacks, and so on. In this case, survivability can be thought of as requiring fault tolerance with very specific and usually elaborate requirements for continued service. Novel forms of error detection, damage assessment, and so on are specific to the context of a complex distributed system.

Our concern is with survivability in the face of disruptions to the vulnerable *distributed information systems* that automate and operate the infrastructures. Tremendous efficiency improvements and service enhancements have been made in recent years in many infrastructure systems by the introduction of sophisticated information systems. In some cases these changes are very visible, e.g., the worldwide access to funds provided by ATM machines. In other cases they are not, e.g., just-in-time delivery of manufactur-

1. Revised August 2000.

ing materials by freight rail. The high-level problem is that enterprises, including civil and military organizations, are put at risk by vulnerabilities of critical infrastructure systems to disruption of their information systems in a world of increasingly complex information systems and increasingly capable adversaries.

Our research goal is to design survivable systems and to enhance the survivability of existing systems against threats that pose risks to the provision of acceptable levels of service. Achieving this goal in practice will require major investments in research, development, and re-engineering. We view survivability as a property that is made attainable, at least in part, by a system's architecture. Thus, we seek to explore, develop and evaluate innovative architectural techniques for achieving survivability in complex new and existing systems.

The survivability enhancement of existing systems is complicated by their size and complexity and thus their resistance to change. We therefore seek survivability enhancement techniques that can be imposed largely transparently on existing systems. The ability to do this is constrained, of course—often heavily—by the particular properties of any given system.

Experimentation is a crucial element of research in this area. It is vital, for example, to the early evaluation of novel survivability techniques. Unfortunately, several factors present serious impediments to experimentation. First, infrastructure systems are privately owned. Second, they are, by definition, critical from both business and societal perspectives. It is inconceivable that their operators would permit experimentation on them. Third, infrastructures are enormous in physical scale, cost and complexity, which makes it infeasible to replicate them in the laboratory.

We have thus adopted an approach based on the use of *operational models* of infrastructure systems as testbeds for developing and evaluating prototype architectural mechanisms for survivability. In this paper we describe a system that permits a wide range of representative models of critical infrastructure systems to be built rapidly and made the subject of experimentation. Provision is made within the system for creation, manipulation, and observation of models of infrastructure systems as well as the introduction of architectural elements designed to enhance survivability. The ability to develop and analyze models and prototype survivability mechanisms rapidly is an important aspect of the work because we wish to explore a range of infrastructure applications and a variety of architectural concepts efficiently.

The rest of this paper is organized as follows. Section 2 discusses the basic modeling concepts. Section 3 presents key aspects of infrastructure applications. Section 4 discusses the requirements for a modeling system. Section 5 presents our system in detail. Section 6 summarizes an example model that we have been built using the modeling system. We discuss related work in Section 7. Section 8 presents our concluding remarks and plans for future work.

2 Operational Models

Given the impediments to direct experimentation with real infrastructures, we have adopted an experimental approach based on operational models. By an *operational model*, we mean a simplified version of the real system that executes as the real system does—as a true concurrent system—but that provides only a restricted form of its functionality. The goal for a given model is to have a simplified laboratory version of the associated infrastructure system that is made manageable by implementing only relevant application functionality and implementing only essential characteristics of the underlying target architecture. Most infrastructure applications, for example, are distributed and this is an essential characteristic, although the particular protocols used in the underlying network are not in most cases. Thus for our purposes an operational model needs to be truly concurrent but it need not use any specific network protocol.

Building operational models of critical infrastructure information systems presents two significant challenges: (1) modeling the critical and no other aspects of infrastructure systems with sufficient accuracy and completeness; and (2) facilitating inclusion in the model of relevant architectural mechanisms to be developed or evaluated. For purposes of experimentation, an operational model has to represent relevant functional and architectural features of a given system, as well as its operational environment, including a dynamic model of vulnerabilities, internal failures and external threats. Once such a model is built, mechanisms must be present to allow prototypes of architectural survivability mechanisms to be introduced. Both models and architectural supplements must be instrumented for collection of data needed to analyze and evaluate survivability mechanisms.

The *prima facie* validity of conclusions derived from studies based on models and prototypes depends on the extent to which they capture the relevant properties of modeled situations. Validation of such results by other means is essential. Our results are not validated to the degree required to warrant adoption into real infrastructure systems. Rather, at this early stage of scientific inquiry into the critical area of infrastructure survivability, our work is at the level of basic experimental systems research, in both the technical and methodological dimensions.

3 Relevant Features of Critical Infrastructures

The design of the modeling system is based on two sets of issues. The first centers on key, relevant features of critical infrastructure domains determined through our in-depth study of several critical infrastructures systems [10]. The second centers on the need to enable experimental architectural research in largely transparent monitoring and control. In this section, we summarize the key domain properties; in the next section, support for experimental research.

The features of the infrastructure domain that we selected for explicit representation in our models were derived from the research questions with which we are concerned. An important aspect of this selection is that the open research problems are posed by precisely those properties that distinguish infrastructure systems from other, more familiar, systems for which high levels of dependability are required. The properties with which we are most concerned are as follows:

- *System Size*
  The information systems supporting critical infrastructure applications are very large, both geographically and in terms of numbers and complexity of computing and network elements. For example, the banking system of the United States includes thousands of nodes in many business situations. Infrastructure systems are also widely distributed, in large part because they must deliver service streams over large, sometimes even worldwide, geographic regions.
- *Hierarchic Structure*
  Many of these systems are structured hierarchically, although "short circuits" are sometimes present for performance reasons. The United States financial payment system, for example, can reasonably be viewed as roughly tree-structured with the Federal Reserve System at the root of the tree. In reality, the Federal Reserve competes with other financial institutions to provide many important services, such as check clearing, so a forest is perhaps a more accurate description of the payment system. When interdependencies among infrastructures are taken into account (e.g., the dependence of banking on electric power and *vice versa*), the structure of the infrastructure system-of-systems is seen as a complex nested hypergraph.
- *Serial Functionality*
  At the highest level many of these systems operate as loosely coupled subsystems, each implementing a function that provides only *part* of the overall service. The complete service is only obtained if sev-

eral subsystems operate correctly in the correct sequence. For instance, the U.S. payment system operates through computations at local branch banks, centralized money-center banks, and very centralized clearing organizations, including the Federal Reserve Bank. In most cases, at least one bank at each level has to operate to clear a check.

- *COTS and Legacy Components*
  The information systems underlying critical infrastructures are and will continue to be built from commercial off-the-shelf components, including standard hardware, operating systems, network protocols, database management systems, job control mechanisms, programming environments, and so on. These systems also include custom-built software, much of it of a "legacy" nature. That is, the software has grown and changed over many years, has a degraded structure, and is thus hard to understand and costly to change.
- *Multiple Administrative Domains*
  Many infrastructure information systems span multiple administrative domains. For example, the U.S. payment system is an extremely complex system of systems for managing transfers of value and commitments to transfer value among financial and other institutions. The institutions include not only local and money-center banks as already noted, but credit card issuing and clearing organizations, check clearing organizations, loan organizations, financial and other exchanges, and so on. Moreover, in traditional physical infrastructures, such as transportation, computing systems are becoming integrated across corporate boundaries to achieve previously impossible supply chain efficiencies.
- *Availability*
  The requirements for availability in infrastructure systems are considerable as would be expected; but availability is a more complex issue than it might appear. In some cases, availability requirements vary with function. It is important in power generation, for example, to maintain power supply if possible, but it is not necessary to maintain an optimal generation profile, and some customers can tolerate some interruptions. In other cases, availability requirements vary with the type of application node. In the banking system, for example, there are many branch banks but very few banks permitted to transfer value between organizations. Thus the availability of service at a local bank is relatively unimportant but it is essential for banks conducting value transfer. Finally, availability requirements vary among infrastructures, and they certainly vary among customers, and even over time. Longer-term power outages are more critical to hospitals than to homes, and in winter than in summer.
- *Complex Operational Environments*
  The operating environments of critical infrastructures are of unprecedented complexity. They carry risks of natural, accidental, and malicious disruptions; sometimes highly variable loads; varying levels of criticality of service; and so forth. For example, freight rail service is especially critical in October—harvest time. Moreover, operational environments are now believed to have potential to exhibit previously unrealized behaviors such as widespread, coordinated information attacks. Cascading failures of infrastructure systems, in which the failure of one node leads to the failure of connected nodes, and so on, are also a real concern.

4 Modeling Requirements to Support Architectural Research

The general requirements for a modeling system derive from the two challenge areas identified earlier: modeling of infrastructure applications and modeling of architectural supplements. In addition, of course, support for experimentation has to be provided to allow for model control and display of results. The details of the requirements derive from the features of critical infrastructures discussed in the previous section, the architectural research goals, and the experiments that are to be performed. In this section, we outline the requirements that the system is designed to satisfy.

4.1 Infrastructure Application Modeling

The requirements for modeling the application infrastructure break down into four categories:

- *Application target architecture.*
  Support must be provided to model distributed systems with large numbers of nodes. Heterogeneous nodes need to be supported along with arbitrary application network topologies. Since infrastructure applications typically make provision for availability, it must be possible to model all forms of redundancy. Finally, the model must allow rapid change of any element of the target architecture so as to permit a wide variety of systems to be modeled.
- *Application functionality.*
  Crucial aspects of application functionality must be modeled. This includes typical processing that takes place on a single node and serial functionality where a series of nodes operate on a data stream to provide a single service to the user. In the infrastructure applications that we have analyzed, a great deal of functionality is associated with manipulation of databases and generation of associated reports. This does not mean, however, that the system needs to provide some form of elaborate database support. In fact, it is such details that have to be abstracted away to produce useful models of tractable size. Database issues associated with a single node are not relevant for our modeling purposes except in so far as they affect system-wide issues.
- *Application operating environment.*
  All relevant input sources and output sinks have to be modeled. Input sources include all types of user requests as well as all forms of application data. Output sinks include displays, reports, application data, etc.
- *Application failures.*
  All relevant types of failure have to be modeled along with all relevant failure parameters. Types of failure include hardware failure, software failure, operator mistakes, environmental trauma (e.g., hurricanes), security penetrations, and so on. Of particular importance are failures that affect large parts of the system such as coordinated security penetrations and local failures that cause cascading effects through the network. Failure parameters of interest include timing, scope, duration, extent, and so on.

4.2 Architectural Supplement Modeling

The architectural concepts of particular interest to us focus on monitoring and control [20]. This requires that sufficient state information be available to permit system-wide errors to be detected and system-wide damage assessment to be undertaken. In addition, we wish to implement continued service for applications in which many nodes cooperate to provide functionality, and in which continued service therefore requires the manipulation of the system as a whole. These general notions lead to the following three basic requirements:

- *Application information.*
  It must be possible to acquire information about both running and corrupted application elements. This requires the ability for applications to supply prescribed information and the ability to acquire application information by observation. The acquisition of application information should be by means similar to those that might be used in a production implementation of the architectural ideas being explored. A key aspect of this requirement is that the facilities be transparent to the application to the extent possible. Thus architectural technologies related to the idea of "transparent wrappers" are of particular significance.
- *Application enhancement.*
  It is necessary to be able to introduce enhanced functionality over and above the required application functionality. Enhanced functionality might have to be added at the network link, computing node, subsystem, or complete system levels. Again, this requirement has to be met by the modeling system in

the same manner as will occur in a complete application.

- *Reconfiguration.*
  Reconfiguration of application architectures is a significant aspect of the way in which continued service is likely to be provided in infrastructure systems recovering from a failure. The modeling system obviously has to support this in as general and flexible a manner as possible.

4.3 Support for Experimentation

Since the purpose of the modeling system is to enable a variety of experiments to be performed, support for experimentation has to be provided, specifically:

- *Model control.*
  Typical models that might be used for architectural research will involve large numbers of nodes and such models will be executed on diverse physical platforms. Provision must be made to control models in terms of mapping models to platforms, model initialization, induction of failures, and so on. These facilities are especially important for complex models that operate on target systems with large numbers of computers, some of which might be geographically remote.
- *Data acquisition and display.*
  Data capture and the display of raw and processed data is essential. But it is difficult with large operational models of the type we require. The difficulty arises from the volume and the asynchronous nature of the data that might be generated in an experiment. A typical model will almost certainly involve thousands of application nodes, each of which might generate a data stream such as statistics on local network message traffic. For purposes of analysis, it might be necessary to process this data centrally and have ordering information so as to be able to predict recovery actions.

5 The Modeling System

5.1 Overview

The system we have developed to meet the various requirements outlined above is called RAPTOR. For purposes of experimentation, the RAPTOR system provides the user with an efficient, easily manipulated operational model of a distributed application with extensive control, monitoring, and display facilities. Figure 1 provides an overview of the system.

A RAPTOR model is specified by defining the desired *topology* and the desired *application functionality.* From the topology, the model is created using services from the modeling system's support libraries and using application software provided by the model builder. *Vulnerabilities* to which the model should be subject are defined and controlled by a user-defined vulnerability specification. During the execution of a model, *symptoms* can be injected into the model to indicate any event of interest to the user. Events might include security penetrations, hardware failures, etc. Any *data* of interest to the user can be collected and made available to a separate process (possibly on a remote computer) for *display* and analysis. Finally, since multiple independent models can be defined from separate topology specifications, complex systems of *interdependent* critical networks can be modeled (see Figure 2).

5.2 Building Blocks for Models

The basic semantics of a model is a set of concurrent message-passing entities that we refer to as *virtual message processors.* Figure 3 depicts the general structure of a virtual message processor. A virtual message processor is provided with a queue of incoming messages that it can read and process as it chooses. Usually, these messages are routed from the input queue to programmable message interpreters. Any new messages generated as a result of interpreting received messages are sent immediately although their

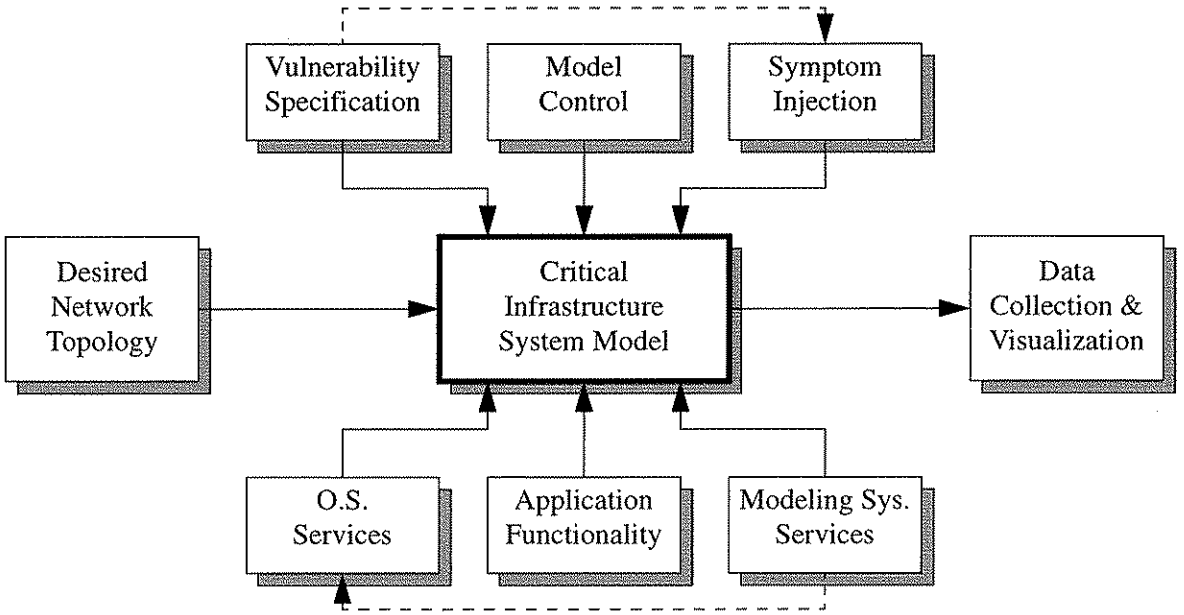

Figure 1. The RAPTOR modeling system architecture

arrival times at their destinations can be controlled. Messages can be generated asynchronously also if needed based on, for example, a timer event.

Within the modeling system, a network node is modeled as a set of one or more virtual message processors each of which is executed by a separate OS-level thread. A typical simple node can be modeled with a single virtual message processor and hence a single thread. More complex nodes can be modeled as a collection of threads thereby allowing such nodes to exhibit concurrent internal behavior. The use of threads for modeling nodes allows the model of a single node to be itself concurrent thereby permitting issues such as synchronization errors and race conditions to be modeled realistically.

Node-to-node communication is modeled by message passing between threads. Messages are passed through memory and so message passing is very efficient. Any thread can send a message to any other thread subject to restrictions imposed by a model's topology (see below). In order to permit models in which resource contention might occur, the input message queue for a virtual message processor can be

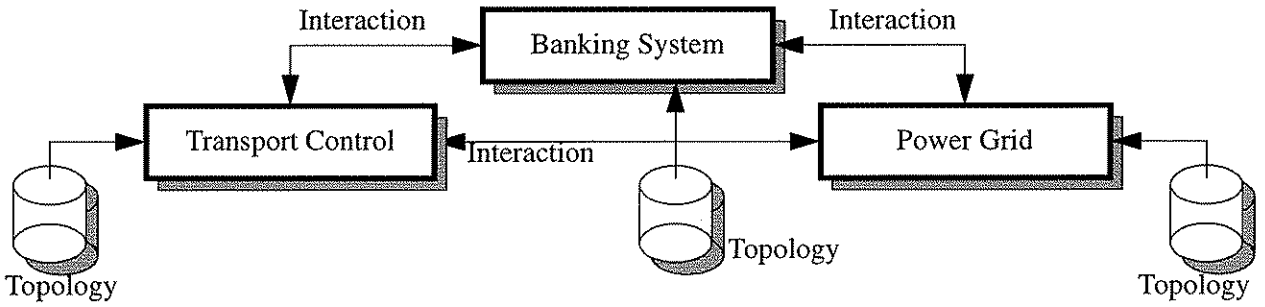

Figure 2. A Model of Multiple Interacting Infrastructure Systems

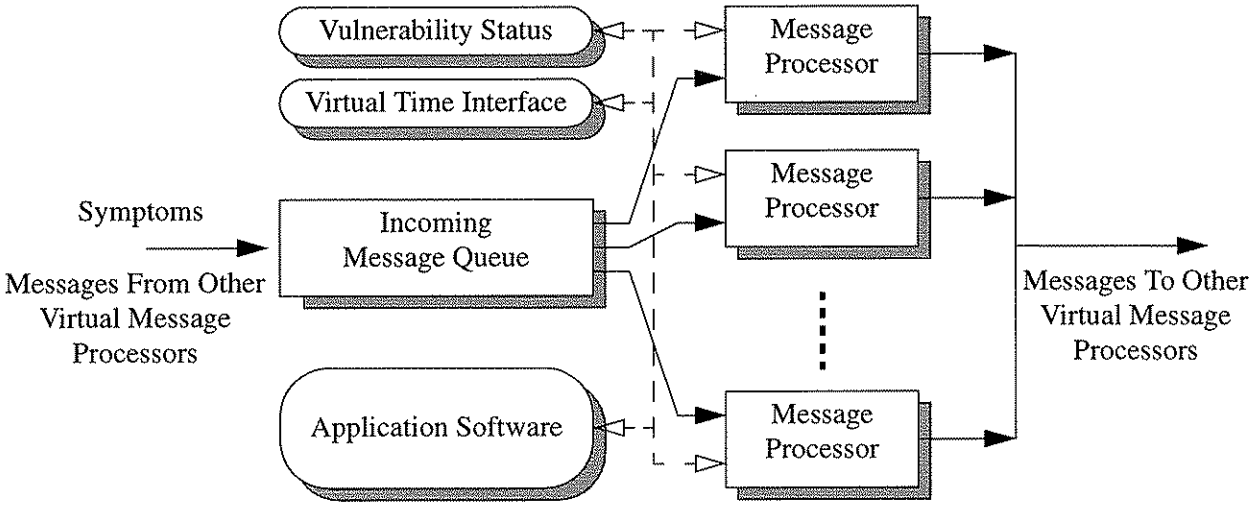

Figure 3. RAPTOR Virtual Message Processor

given a maximum size. If the input message queue is full when a message arrives, the attempt to send the message to that virtual message processor fails. Message transmission times can also be specified to allow transmission delays to be modeled (see section 5.5).

5.3 Model Topologies

The description of the model to be run is defined in a topology file. The topology defines precisely what nodes are to be created, what software each will run, and what the connectivity is to be. By using this approach to the definition of models, different models can be created quickly and modifications to models can be effected easily.

The topology specification defines the required virtual message processors that are to be run so it is a simple matter for the user to arrange for a node in the model to consist of any desired number of virtual message processors. The specification of connectivity is unidirectional, i.e., it states that one virtual message processor may send messages to another. By combining this specification with the virtual message processor specification, nodes of any complexity can be specified. An example is shown in Figure 4. In this example, the large circles represent nodes, the smaller grey circles represent virtual message processors, and the arrows represent communications links. The model has three nodes each of which has a different number of virtual message processors and where the communication topology is very specific.

5.4 Modeling the Application

In modeling critical infrastructure systems, it is essential to model the *application functionality* that is necessary for the problem being studied but to omit all unnecessary detail. The range of functionality is considerable, and this presents a challenge in the design of a modeling system. The approach taken in RAPTOR is to allow the functionality of the different nodes in a system to be defined in a high-level language (C++). The advantage of this approach is that it permits any form of functionality to be expressed and any required level of detail achieved. The effort involved is commensurate with the results, however, in that building a model in this context requires some programming effort.

5.5 Modeling Time

A comprehensive notion of *virtual time* is supported in the RAPTOR modeling system. A clock is main-

tained by the system that is incremented when all virtual message processors are either idle because they are blocked waiting for a message or have blocked themselves waiting for time to advance.

The virtual time mechanism permits the modeling of both computational and communication delays throughout any given model. Computational delay is modeled by individual virtual message processors determining that they have completed as much work as they should in a single time interval. In this way, virtual message processors can operate at any relative speed and they can adjust their speed if necessary because self-enforced blocking on time can be conditional.

Communication delay is modeled by defining a delay between the sending of a message and the time when it should arrive. Thus, a virtual message processor can send a message and indicate that its delivery should be delayed by t time units from the time it is sent.

This time mechanism allows studies of performance including throughput, distributions of delays, delays associated with specific events, and so on.

5.6 Modeling Hazards, Threats and Vulnerabilities

Vulnerabilities are circumstances that can lead to the failure of a system. A security vulnerability, for example, is an aspect of a system that an adversary might exploit to harm a system or to steal information. Simple but unexpected vulnerabilities have been the route by which many serious virus attacks have propagated. Vulnerabilities are both common and very complex in critical infrastructure systems. In almost all cases, either the existence of a vulnerability is unknown or the vulnerability is not considered likely to be attacked. This aspect of the problem makes modeling vulnerabilities and their effects at the level of a single node essentially impossible.

The RAPTOR modeling system supports the introduction of known vulnerabilities into nodes as properties that the nodes have. This is done by requiring that the virtual message processor code maintain a data structure for all the vulnerabilities in the model with a parameter reflecting that virtual message processors susceptibility to each vulnerability. The parameter might be absolute (yes or no) or it might be probabilistic for each vulnerability. This mechanism permits, for example, statistical analysis of the effects of vulnerabilities that have certain distributions across the network. As an example, almost all of the nodes in a

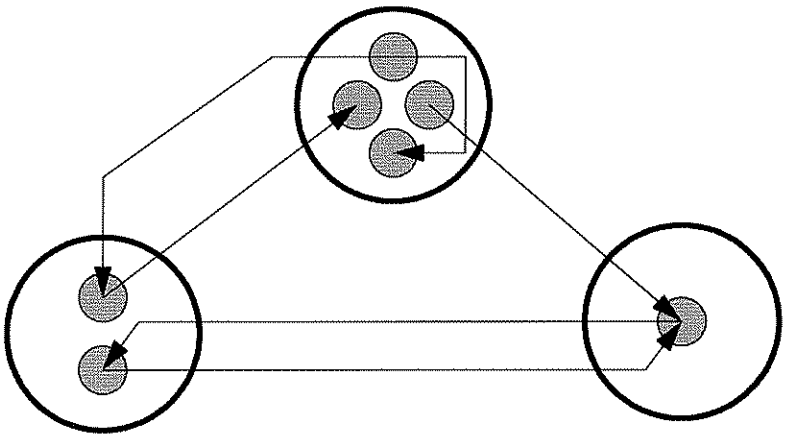

Figure 4. Example Simple Network of Complex Nodes

model might be defined not to be susceptible to a security attack (such as password guessing) but a small fraction, say 0.001%, defined to be susceptible.

Symptom injection is the way in which events such as security attacks and hardware failures are modeled, and the RAPTOR modeling toolset provides a set of fault types, occurrence rates, and durations. Working with the clock that is recording virtual time, the modeling system dispatches symptoms to select nodes or links in any prescribed sequence that is of interest to the researcher. This mechanism is implemented using the message-passing facilities so nodes merely receive a message at a specific virtual time announcing that an event (such as a hardware failure) has occurred. Associated application semantics (such as cessation of all activity within the node) is then effected by the node's "application" software.

Combined with the vulnerability mechanism, symptom injection allows the researcher to model complex fault and attack scenarios. For example, sequences of symptoms can be injected over time (either a short time or a long time) and these symptoms can be taken to represent coordinated security incidents. Similarly, a mixture of faults can be modeled including incidents such as a terrorist attack followed by a coordinated security attack. Each of these has to be programmed as a separate vulnerability and separate symptom sequences but the requisite programming is simple and permits great flexibility.

5.7 Modeling Survivability Mechanisms

Virtual message processors provide network addressing and message transmission at the model level to support required communication structures. They are not, however, limited to being used as elements of an infrastructure model. Thus arbitrary architectural extensions can be modeled using virtual message processors as the building blocks. For example, a sophisticated control-system architecture can be effected simply by merely defining the requisite topology (what nodes there are and to which application-system nodes they will be connected) and by defining the control-system functionality just as the application functionality is defined.

Shell and wrapper architectures are trivial to implement in a RAPTOR model since virtual message processors are responsible for processing messages they receive but there is no system-prescribed processing. Thus a shell can be effected by merely arranging for all incoming or outgoing messages, or both, to be processed by a second set of message processors in additional to those used for basic application functionality within a given virtual message processor.

5.8 Observing Models

Being able to *visualize* what is going on in a critical infrastructure system is important if humans are to be involved in decision making yet the technology for visualizing large networks is not adequate for the very large networks that are at the heart of critical infrastructure systems. The RAPTOR modeling system permits data acquisition of any form and at any time because data acquisition can be programmed as part of the application software. RAPTOR provides a display framework to permit the development of displays suitable for any particular model. The framework provides a collection point for data together with a link to a remote process that can be programmed to include any display features that the model requires. Since the display is remote, it can be executed by a different computer thereby minimizing impact on model execution.

6 Evaluation

The versatility of the virtual message processor construct has provided us the ability to build infrastructure models and to evaluate experimental survivability mechanisms rapidly and easily. We illustrate the use of the modeling system with an example. We have used the RAPTOR system described in this paper to build

several models of a critical application from the banking system, namely the U.S. financial payment system. Our selection of this particular application for modeling was based on our more detailed understanding of the domain—it is arbitrary and of no particular significance. For the example, we describe the largest and most comprehensive model that we have built to date.

The research goal for this particular model is to explore a survivability architecture in which a supplementary hierarchical control system is used to react to widespread failures of several types [3, 14, 20]. This case addresses three critical research issues: (1) the feasibility of distributed, hierarchical control as a survivability architecture; (2) the control algorithms required to respond to failures of different types; and (3) mechanisms that could be employed to specify survivability control policies for the control system.

In addition to these research questions, the development of this model was also used to evaluate the modeling system itself. Clearly issues such as performance, utility, flexibility, and ease of use were of concern.

6.1 Model Architecture

The overall network structure in this model is a tree. This is typical of the way banks are connected for payment purposes, but our topology is strictly hypothetical. The model includes three types of application node. The first is a *branch* bank providing customer service. Such nodes appear as leaf nodes in the tree. The second type of node is a *money center* bank—essentially the primary information center for a single commercial bank. A money-center bank appears as an intermediate-level node in the tree, and has a set of local banks as children. The third type of node represents the *Federal Reserve System* and is the root of the tree. Money-center banks are connected to the Federal Reserve System.

Needless to say, the information system that effects payment in the U.S. is a very large network, and we could not model this exact scale although the model is of the right order. Our current model is composed of almost 10,000 application nodes with almost 100 money-center banks, each of which has 100 associated branches.

The model of the Federal Reserve System includes a primary server and two geographically remote warm spares that permanently mirror the data held by the primary server. They are able to provide service to the remainder of the network if the primary fails, but, in order to do so, the money-center banks must reroute payment requests and wait for service to be initialized. This model is representative of the availability mechanisms actually used by the Federal Reserve System.

6.2 Application Functionality

The application functionality we have implemented in the model includes check processing and large electronic funds transfers. Each payment demand includes typical routing information—source bank, source account number, destination bank, and destination account number as well as the payment amount. User's bank accounts are held at the branch banks and it is there that all payment requests are made. A load generator (another virtual message processor) creates random sequences of payment demands that take the form of either a "check" or an EFT request.

As in the real payment system, payment demands below a certain threshold value are grouped together so that funds transfers between money-center banks are handled in bulk by the Federal Reserve System at scheduled settlement times. Bulk funds received by a money-center bank have to be dispersed through the bank's own network so that the correct value reaches each destination account. This part of the application models the processing of paper checks. Transfers of funds where the value exceeds the threshold value are effected individually and upon receipt of the demand. This aspect of the application models large EFT request processing. The two-tier approach to payment processing is representative of the overall structure

of the real payment system.

6.3 Architectural Supplement

We have described the control systems architecture with which we are experimenting in detail elsewhere [20]. It is a (typically) distributed system that is separate from the application system which senses the state of the application and reconfigures it in the face of adverse circumstances by sending it reconfiguration commands. Sensing the state of application nodes and transmitting commands for reconfiguration takes place conceptually via protection shells that surround application nodes. To permit reconfiguration to be tailored to different semantic levels in the application network topology, a control system typically operates hierarchically with lower levels supplying summary information to upper levels to optimize control decisions.

The function of the control system is to implement *survivability policies*. A survivability policy describes a non-local event of concern (such as a failure of more than a certain number of application nodes), and the recovery commands and their sequence that are have to be sent to application nodes for that particular event. The actual continued service is implemented by the application—it is its invocation that is the responsibility of the control system.

The control system used in the evaluation activity has ten nodes that are connected to groups of application nodes. Each of three of the control-system nodes is connected to one of the Federal Reserve processing nodes together with roughly ten of the money-center banks. The control system observes the application network and determines action based on error detection and error recovery specifications that have been included in the control algorithm.

For purposes of experimentation, the current model implements four responses that can be used for a variety of faults that might arise. These responses are designed to demonstrate and evaluate key aspects of the modeling system and the control-based survivability architecture:

- *Federal Reserve Redirection*
  This response requires that the entire payment system switch to the use of a warm spare in the Federal Reserve System.
- *Node Isolation*
  This response requires that a node whose intrusion-detection system is triggered be isolated and ignored by the remainder of the network.
- *System-Wide Key Replacement*
  This response requires that the entire payment system switch cryptographic keys and account passwords.
- *Comprehensive Shutdown*
  This response requires that the entire payment system be shut down.

In a specific scenario that has been used for evaluation, the banking system is first attacked by a terrorist who "bombs" the primary Federal-Reserve server. This results in a switch to the warm spare for continued service. The terrorist bomb is followed by a coordinated security attack in which a series of money-center and branch banks are attacked one after the other. As each attack is detected, the control system directs that the associated node be ignored. After five attacks have been observed, the control system directs a system-wide change of cryptographic keys and account passwords. At that point, communication with nodes that had been attacked and were being ignored is restored. Finally, when a total of ten attacks have taken place, the control system orders the entire payment system to be shutdown.

None of the key services required by transaction processing systems (such as two-phase commit protocols) are provided by the modeling system, nor are they intended to be. The modeling of continued service that is being developed in this example is at the level of system and application management. We are abstracting away important issues such as consistent recovery in distributed heterogeneous systems. Our focus is, instead, on monitoring and control in large distributed systems. We assume that lower level details are provided by the application. They could be added explicitly as part of the application functionality in a model built for a different research goal.

The control system does implement a two-phase commit protocol in this particular modeling exercise and uses it to ensure that consistent decisions are made about what control-system node is to do what regarding recovery.

6.4 Model Implementation

The topology of this model is defined entirely in the topology specification. Application nodes are virtual message processors and the application's communications system is implemented by links between virtual message processors. The control system architecture model is also built with virtual message processors.

The application functionality is implemented by small sections of C++ source providing message interpretation in the application nodes. The functionality implied by the redundancy model for the Federal Reserve System is achieved with a trivial amount of programming within the application functionality.

6.5 Results

Our results to date are in three areas: (1) the utility of the system; (2) the performance of the system; and (3) the feasibility of hierarchic control of network survivability using the control system paradigm. In the first area, the modeling system supported well and in all respects development of the model that we have discussed. Building of the model was easy. Its specification is short. The facilities of the model, especially the pattern of use of components, met all of our demands. We were able to build several versions of the model quickly (in a few days) and incrementally.

To date we have assessed the runtime performance of the system (as opposed to its support for model construction) informally and in only a single area—the runtime performance of a physical computer. On a Pentium-based machine with 64 Mbytes of main memory and a typical disk configuration, acceptable performance is obtained with up to about 10,000 virtual message processors running concurrently processing messages associated with the payment-system model.

Finally, the model described here incorporates a preliminary hierarchic control system that is designed to provide significant survivability enhancement. Although no performance quantification has been undertaken, the model demonstrates the feasibility of network-wide state assessment and damage assessment coupled with a hierarchic approach to state restoration, and continued service. The latter is especially important since survivability of large distributed applications will almost certainly require the following two activities to cope with major failures:

- Significant reconfiguration of the network's topology (state restoration) where different elements of the reconfiguration are coordinated yet tailored to different circumstances throughout the network.
- Substantially different alternative or reduced applications (continued service) at different locations based again on the different circumstances throughout the network.

Using a data collection facility integrated into the model, we measured "transactions" successfully com-

pleted per unit time. Here a transaction is a retail payment, either a "check" or an "EFT" order. Running the model with the survivability enhancement disabled, the transaction rate dropped almost to zero as soon as the Federal Reserve server was bombed. With the survivability mechanism in place the system maintained a reasonable rate of transaction processing with dips occurring as each trauma hit the system.

7 Related Work

The use of models to explore systems that do not admit direct or comprehensive manipulation is not new. Models are used widely, for example, as tools for computer architecture design and development [12, 18, 19, 23] to permit design trade-offs to be studied that would be infeasible by any other means. Simulation is also used, of course, to provide environments that are not otherwise readily available.

There are several existing frameworks and/or toolkits available that support the development of distributed systems. For example, Java's Remote Method Invocation (RMI) [21] supports a distributed object model which abstracts the communication interface to the level of an object method invocation. For our purposes, however, RMI is not close enough to the semantics of infrastructure applications (message passing) to permit the simple development of models.

The Common Object Request Broker Architecture (CORBA) [17] is a conceptual "software bus" that allows applications to communicate with one another, regardless of who designed them, the platform they are running on, the language they are written in, and where they are executing. The emphasis in CORBA is a distributed system paradigm that promotes interoperability. Again, however, the software that supports CORBA cannot be used easily to build the operational needs that we require.

Much research has addressed the issue of fault tolerance in distributed applications. The Isis toolkit [5], for example, provides a set of mechanisms to support reliable communication in a process group. Other examples include the work on replication of Fabre et al. [9] and transaction models by Chelliah et al. [6]. Existing techniques do not deal with modeling in the sense used with the system that we describe, including transparent insertion of architectural elements, and control and measurement of experimental survivability models. Nor do existing techniques deal with the multitude of system-wide issues that arise in critical infrastructure applications.

Modeling and characterizing of distributed systems have been studied extensively. Andrews discusses process types and process interaction in distributed systems [1]. Nikolaidou et al. describe a Distributed System Simulator (DSS) [15]. The DSS is an integrated environment for performance evaluation of distributed systems. A distributed system is viewed as a combination of a distributed application and a network infrastructure. The DSS permits analysis of the behaviors of the network infrastructure under conditions imposed by the defined distributed application and estimates performance parameters. Though related, our system differs in that the objective is to investigate vulnerabilities of new and existing infrastructure systems and to study ways to deal with them, rather than addressing traditional performance issues.

8 Conclusions

Dealing with the survivability issues posed by modern critical information systems in infrastructure applications presents many challenges. The systems are large, usually depend upon COTS components, contain extensive legacy code, and must meet multiple diverse dependability requirements. The need for improved survivability is increasing as more and larger systems are deployed, as society becomes more dependent on critical infrastructures, and as some threats (such as the possibility of national-scale malicious attacks) become more likely and their perpetrators more sophisticated.

A serious impediment to research in this field is the difficulty of experimentation. Real systems cannot be the subject of experiment for the most part because real systems are, by definition, critical and their operators cannot risk the possibility of damage during experimentation. We have described a modeling system that begins to address this problem by supporting the development and evaluation of operational models of infrastructure applications and survivability mechanisms.

The system that we have described allows us to develop executable models very quickly. Preliminary results based on an example model have shown that the system meets the basic requirements set forth, and that model runtime performance is adequate for our experimental purposes. The role of the system in our research is to permit experimentation with architectural concepts that support survivability. The model that we have described demonstrated a hierarchic control mechanism providing error detection, damage assessment, and facilities for continued provision of service that can be tailored to the needs of different parts of a specific application. Of course, we have not established the utility of such a control system for real infrastructure survivability. Nevertheless, our modeling system and research method provide a basis for undertaking the basic experimental research needed to begin to evaluate such concepts.

Acknowledgments

This effort sponsored in part by the Defense Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-1-0314. The U.S. Government is authorized to reproduce and distribute reprints for governmental purpose notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Rome Laboratory or the U.S. Government. Kevin Sullivan was also supported by the National Science Foundation under grants CCR-9502029, CCR-9506779, CCR-9804078.

References

1. G. R. Andrews, Paradigms for Process Interaction in Distributed Programs, *ACM Computing Surveys*, Vol. 23, No. 1, March 1991, pp. 49-90.
2. R. L. Bagrodia and C.-C. Shen, MIDAS: Integrated Design and Simulation of Distributed Systems. *IEEE Transactions on Software Engineering*, Vol. 17, No. 10, 1991, pp. 1042-1058.
3. R. Bateson, *Introduction to Control System Technology*, (6th ed.), Upper Saddle River, NJ: Prentice Hall, (1998).
4. M. A. Bauer, R. B. Bunt, A. El Rayess, P. J. Finnigan, T. Kunz, H. L. Lutfiyya, A. D. Marshall, P. Martin, G. M. Oster, W. Powley, J. Rolia, D. Taylor, and M. Woodside, Services Supporting Management of Distributed Applications and Systems, *IBM Systems Journal*, 36, 4, (1997), 508-526.
5. K. P. Birman and R. van Renesse, *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, Los Alamitos, CA, 1994.
6. M. Chelliah and M. Ahamad, *Multi-Model Fault-Tolerant Programming in Distributed Object-Based Systems*. Technical Report, GIT-CC93/72. College of Computing, Georgia Institute of Technology.
7. Defense Modeling & Simulation Office (DMSO), Introduction to the High Level Architecture. Simulation Interoperability Workshops, September 8-12, 1997. See http://hla.dmso.mil/hla.
8. R. J. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. Longstaff, and N. R. Mead, *Survivable Network Systems: An Emerging Discipline*, Technical Report CMU/SEI-97-TR-013, Software Engineering Institute, Carnegie Mellon University, (November 1997).
9. J.-C. Fabre, V. Nicomette, T. Perennou, R. J. Stroud, and Z. Wu, *Implementing Fault Tolerant Applications using*

Reflective Object-Oriented Programming. Technical Report, LAAS-CNRS, LAAS-CNRS#94156, March 1995.

10. J. C. Knight, M. C. Elder, J. Flinn, and P. Marx, *Summaries of Three Critical Infrastructure Applications*, Technical Report CS-97-27, Department of Computer Science, University of Virginia, Charlottesville, VA 22903 (December 1997).

11. J.C. Knight, R. W. Lubinsky, J. McHugh, and K. J. Sullivan, *Architectural Approaches to Information Survivability*, Technical Report CS-97-25, Department of Computer Science, University of Virginia, Charlottesville, VA 22903 (September 1997).

12. S. F. Lundstrom and M. J. Flynn, *Design of Testbed and Emulation Tools*, Technical Report, CSL-86-309, Computer Systems Laboratory, Stanford University, Sept. 1986.

13. K. Marzullo, R. Cooper, M. D. Wood, and K. P. Birman, Tools for Distributed Application Management, *IEEE Computer*, (August 1991), 42-51.

14. R. Murray-Smith and T. A. Johansen (eds.), *Multiple Model Approaches to Modeling and Control*, Taylor & Francis: London, UK, (1997).

15. M. Nikolaidou, D. Anagnostopoulos, and P. Georgiadis, *Modeling and Simulation of Distributed Systems*, Technical Report TR97-0011, Department of Information, University of Athens, Greece.

16. Office of the Undersecretary of Defense for Acquisition & Technology, Report of the Defense Science Board Task Force on Information Warfare-Defense (IW-D), (November 1996).

17. A. Pope, *The COBRA Reference Guide: Understanding the Common Object Request Broker Architecture*, Addison Wesley, 1998.

18. M. Rosenblum, S. A. Herrod, E. Witchet, and A. Gupta, Complete Computer Simulation: The SimOS Approach, *IEEE Parallel and Distributed Technology*, Fall 1995.

19. C. Ruemmler and J. Wilkes, An Introduction to Disk Drive Modeling. *IEEE Computer*, March 1994, pp. 17-28.

20. K.J. Sullivan, J.C. Knight, X. Du and S. Geist, Information survivability: a control systems perspective, *Proceedings of the 1999 International Conference on Software Engineering*, Los Angeles, May 1999.

21. Sun Microsystems, *Remote Method Invocation Specification*. 1997. http://java.sun.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc.html.

22. United States Government Printing Office (GPO), No. 040-000-00699-1, *Protecting America's Infrastructures: Report of the Presidential Commission on Critical Infrastructure Protection* (October 1997).

23. J. E. Veenstra and R. J. Fowler, MINT: A Front End for Efficient Simulation of Shared Memory Multiprocessors, *Proceedings of the Second International Workshop on Modeling, analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 1994, pp. 201-207.