Information Survivability Control Systems

Kevin Sullivan, John C. Knight, Xing Du, and Steve Geist
University of Virginia, Department of Computer Science
Thornton Hall, Charlottesville, VA 22903, USA
Tel. (804) 982-2206; FAX: (804) 982-2214; URL: www.cs.virginia.edu
E-mail: {sullivan, knight, xd2a, smg9c}@cs.virginia.edu

ABSTRACT

We address the dependence of critical infrastructures—including electric power, telecommunications, finance and transportation—on vulnerable information systems. Our approach is based on the notion of control. We envision distributed, hierarchical, adaptive, multiple model, discrete-state control systems to monitor infrastructure information systems and respond to disruptions (e.g., security attacks) by changing operating modes and design configurations to minimize loss of utility. Controlling legacy information systems presents some significant challenges. To explore and evaluate our approach, we have developed a toolkit for building distributed dynamic models of infrastructure information systems. We used this toolkit to build a model of a simple subset of the United States payment system and a control system for this model information system.

Keywords
Infrastructure survivability, control, architecture economics

1. INTRODUCTION

The survivability of critical infrastructure systems, such as electric power distribution, telecommunications, freight rail and banking, has become a major concern of the United States government, and will garner increasing concern from private industry [5, 15, 18]. The intuitive notion of survivability is clear: we want infrastructure systems that continue to provide acceptable service levels to customers in the face of disturbances, natural, accidental or malicious.

It is now believed that the reliance of infrastructure systems on fragile information systems puts our infrastructures at risk of catastrophic failure. Threats arise from reliance on commercial (COTS) components of unquantified reliability and security, legacy software that defies comprehension and evolution, wide-area distribution, strict performance requirements, complex networks, and openings to outside manipulation through networks and outsourcing of design.

The vulnerability of critical infrastructure systems to failures in underlying information systems creates new challenges for software engineering research, as well as research across many related disciplines, including systems engineering, computer security, real-time systems, and so forth. In software engineering, architectural support for survivability emerges as a top research priority.

Defensive architectural design, e.g., computer security and disaster recovery planning, is an aspect of a comprehensive architectural approach to infrastructure survivability. However, when defenses fail to prevent disturbances, then reaction will be necessary to minimize the loss of utility provided by an infrastructure. In this paper, we address the reactive element of survivability. For example, in reacting to a coordinated security attack, computers hosting critical databases might be disconnected from a network. Reconfiguration will involve changes in operating modes, module implementations and interconnection structure.

We describe our approach, which is based on a control systems perspective and our previous work [10]. Control theory [1] provides a vocabulary for reasoning about how to keep systems operating as desired, and for structuring information-based mechanisms to effect such control.

The characteristics of the systems that we seek to control imply novel control systems. Infrastructure systems and their information systems are large and distributed; so control must be decentralized. System-wide monitoring, reporting, and control implies some centralization, and so a hierarchical structure. Controlled systems change (e.g., as hardware fails and as they evolve over time), so a control system must be adaptive. Finally, we seek to control not physical but information systems, whose behaviors are described not by differential equations but by discrete state transitions; so we need discrete state control systems. These considerations take us beyond canonical control theory. We recognize that our appeal to control theory begins at the metaphorical level. Strong theorems, e.g., on stability, robust control, etc, are unlikely to hold in our context.

Section 2 presents the rationale for a control approach. Section 3 introduces a toolkit for building dynamic models of infrastructure information and systems. Section 4 presents a simple model of one information system; and Section 5, its controller. We summarize insights in Section 6 and related work in 7. Section 8 presents our conclusions.

2. SURVIVABILITY CONTROL SYSTEMS

An infrastructure provides to its customers a *service stream over time*. For example, the electric grid provides a stream of electricity to each home and business. Such a stream has a *value* or *utility* to a customer depending on its particular needs. Provision of energy to homes is more valuable in winter than summer, for example. At some level, there is an aggregate *value added* that depends on the reliance of customers on service and the criticality of each customer in a broader context, as defined by societal or business policy.

The direct consequence of a disruption of an infrastructure system is a reduction in the service stream to customers. The key consequence is a *loss of value over time*. The value lost to a given customer depends on its reliance on the service, and the kind and duration of disruption. Individual loss sums to an aggregate loss. Survivability means that this aggregate loss is minimized and judged acceptable under defined adverse circumstances.

Society now finds itself in a doubly dangerous situation in which infrastructure survivability is not ensured (assuming reasonable definitions of *acceptable* and knowledge of failure modes that have become possible, neither of which, however, might ever have been made explicit). Massive computerization has enabled efficiencies through tightened coupling. Just-in-time delivery of automotive parts by rail has enabled dramatic inventory reductions; but manufacturers are now more reliant on a highly reliable stream of timely deliveries. The cost of interruptions grows more rapidly in time now than before computerization. At the same time, increasing reliance on computers increases vulnerability to disruption. The problem is to devise approaches to *infrastructure information systems design and evolution* that simultaneously enable the efficiencies that computers make possible while ensuring that the costs of service stream interruptions remain acceptable in the face of disruptions to underlying information systems.

2.1 Control system perspective of survivability

In this paper we present a *control systems* approach to infrastructure survivability in the information age. The information systems that run our infrastructures appear to be vulnerable to disruption—by natural disaster, accident, mismanagement, design error, malicious attack, etc. In the face of such disruptions, actions must be taken to ensure that a managed infrastructure system continues to meet its survivability requirements, specified in terms of acceptable reductions in value added under defined circumstances.

When disrupted, an information system must be adjusted so as to continue to provide information services on which the acceptable provision of infrastructure service depends. Adjustment will involve reconfiguration of the information system. To be reconfigured, an information system must be reconfigurable. System reconfigurability can occur at many levels, including operating parameters, module implementations, and physical devices. We refer to the set of all possible *configurations* as a *design space*.

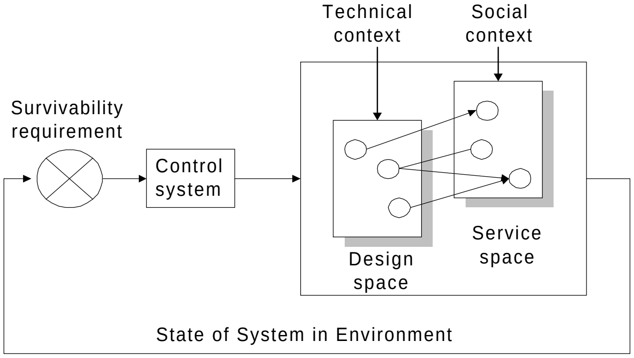

Figure 1. Survivability control system concept

The traditional approach to managing complex systems, such as avionics platforms, to ensure continued acceptable provision of service is to use a control system. We envision the use of control to manage *information systems* based on data from infrastructures, their information systems, and their operational environments. In essence, such a control system is responsible for choosing a configuration at each point in time based on current conditions to minimize the loss of aggregate value, with assurances that under defined circumstances loss of value will not exceed defined limits.

As illustrated in Figure 1, a given design configuration supports some level of infrastructure service. A given level of service is related to value through context as discussed above (is it winter?). The design space determines the extent to which a control system can manage loss of value in the face of disturbances. An information system for a *survivable* infrastructure system must have a configuration enabling service provision that meets value requirements for each defined hazard or circumstance to which the infrastructure and its information system is subject.

The notion of using control systems to manage information systems is not entirely new, as we discuss in Section 7. Our contributions are: (1) interpreting survivability as a large-scale control and optimization problem; (2) deriving basic characteristics of control systems from properties of infrastructure information systems; (3) presentation of an experimental systems framework and methodology for research on infrastructure information system survivability; and (4) insights arising from the control perspective.

2.2 Hierarchical adaptive control

We frame *decentralized, hierarchical, discrete-state, adaptive control* as an architectural style for survivable infrastructure information systems. A control system manipulates a controlled system on the basis of sensor data from the controlled system, predictions of its behavior, and other such information (including, in *stochastic control,* estimates of probabilities of future states of nature) to maintain acceptable levels of system operation. Examples are familiar to every engineer.

A *decentralized* control system is one in which parts of the control system control parts of the underlying system autonomously. An *adaptive* control system is one that can continue providing control in the face of changes to the controlled system and to the control system. For example, an adaptive control system for an avionics application can ensure that an aircraft remains under control even if it loses part of a wing and some sensors in an air engagement. A *hierarchical* control system is one in which control actions are determined at a number of levels in a hierarchical system, with low-level control system elements influencing and being influenced by higher levels of control. Tactical decisions might be made close to individual components in a controlled system, while strategic decisions are made at a higher level based on aggregated global system state.

In our formulation, the controlled system is the information system automating an infrastructure system. In freight rail, for example, the physical system comprises rails, cars and locomotives. This infrastructure is controlled by a complex information system that manages train assembly, dispatch, scheduling, motion control, billing and so on to meet performance, safety, business and other objectives. We envision superimposing a survivability control system atop such an information system. Among a wide variety of survivability properties achieved by this structure, such a control system would implement intrusion monitoring and response; system-wide fault tolerance; and controlled service degradation under adverse conditions.

2.3 Why hierarchical and adaptive?

The need for a hierarchical structure is implied by the size and distribution of infrastructure information systems. It is implausible, for example, to have a single computing node monitoring the entire United States banking system. Each major bank would have a local control system interacting through abstract interfaces with higher-level (e.g., Federal Reserve) and lower-level (e.g., branch) control systems.

A hierarchical structure is natural to support scalability through local control and the passing of aggregated status information up and down a hierarchy. Such information flows will be needed in practice to implement system-wide reconfiguration policies with acceptable performance. Such a structure enables local control nodes to implement policies based on local information and aggregated global state passed from above. In addition to performance, hierarchy enables abstraction and complexity control in control system implementation. Details of local application nodes are abstracted by local control nodes. Higher level control nodes are specified and implemented in terms of the observable and controllable aspects of control nodes at the next level down the control hierarchy. Hierarchy is also intended to foster evolvability of the control policies. Such evolvability will be critical to effective “learning” by a system over time, and as the underlying information and infrastructure systems evolve.

A disciplined approach to the modular design of the control system will also be critical in building *adaptive* control systems that can tolerate the loss, addition, or modification of control and controlled nodes. The control theoretic notion of *multiple-model adaptive control*—in which the control system views the controlled system as being in one of a number of possible distinct operating regimes in which distinct control rules apply—offers especially attractive prospects. Our dynamic modeling toolkit, which we describe next, provides a monitoring capability that is used to connect control nodes in such a way as to ensure that nodes within the control system have a model of both the controlled and control system.

3. A TOOLKIT FOR EXPERIMENTAL SYSTEMS

A serious impediment to research on infrastructure survivability is that researchers can neither experiment with nor even measure infrastructures or their information systems because they generally have little or no direct access to them. Our approach is to build dynamic models of these systems, and to explore, develop and evaluate our control systems approach in the context of these dynamic models. The idea of building infrastructure simulations is not a new idea: it is done routinely in the electric power industry, for example; and such simulations are typically used as subjects of and elements in control systems [27]. We know of no other work using simulations of infrastructure information systems to serve as subjects of control systems for infrastructure survivability research.

In this section, we describe briefly our toolkit for building dynamic models of infrastructure systems and their superimposed control systems. The basic building block in this toolkit is what (for historical reasons) we call the *L-node*. L-nodes provide a flexible mechanism for building distributed dynamic models and control systems.

An L-node is a multi-threaded message dispatching process that can be programmed to support an object-oriented design style. L-nodes communicate by passing *Message* objects, which are structured, application-level messages that are serialized for network communication. An L-node is implemented as a Windows NT process. A dynamic model of an infrastructure system is implemented as a set of communicating L-nodes. In the next section, we describe a highly simplified dynamic model of the United States payment system that we built in this style. Each L-node in such a system has an address represented as an integer. At the application level, L-nodes send messages to each other using these addresses. At the network level, the TCP network protocol is used to transmit such messages.

A key property of the design of our toolkit is that a new L-node can be inserted transparently between any two nodes in a system. (At present, each application L-node has exactly one such associated L-node.) Messages passed between the two nodes are then routed through these interposed shell [10] or mediator [20, 21, 23] nodes. Because messages are represented as application-level semantic objects, shell nodes can process communications between application nodes at the application level and so can reason about the system state and behavior in its own terms (as opposed to having to reconstitute application semantics from low-level network traffic, for example). The ability that this structure gives us to build model systems that transparently reflect upon and modify their own operation models the kind of approach that we would want for real survivable infrastructure information systems.

We use this "shell" mechanism in a variety of ways in our dynamic models. Modeling the transparent wrapping and monitoring of legacy systems is one such use. Other uses are merely implementation details within our dynamic models. We use shell processes to receive messages from L-nodes that are programmed to simulate failures of and attacks upon application nodes, for example.

We implemented this "modeling middleware" on Pentium-based machines running Windows NT and using TCP/IP for communication. The programming language environment is Visual C++ 5.0 and its Microsoft Foundation Class (MFC). We have experimented with running programs built using this toolkit on an "infrastructure" consisting of about a dozen machines distributed across the United States and connected via the Internet. The machines were in Charlottesville (VA), Portland (OR), Tucson (AZ), and Pittsburgh (PA).

4. A MODEL OF THE U.S. PAYMENT SYSTEM

In this section, we describe as an example a distributed dynamic model of the United States payment system [11] together with a variety of malicious attacks to which it might be subjected. Our model is grossly simplified in relation to the real banking system, of course, but it captures some essential function and architecture. In the next section, we describe how we use this system as a testbed for survivability research.

Our "payment system" implements the payment activities of a three-level hierarchical banking system with branch banks as leaves, money-center banks in the middle, and the Federal Reserve at the root of a tree. Depositing "checks" at a branch bank results in requests for transfers of funds among accounts. When a check with a source-account number, destination-account number, and amount is deposited in a branch bank, the check is handled internally at the branch bank if both accounts are within that branch. If not, the check is passed up to the money center. If the source-account number of the check is at a branch bank that is connected to the money center in question, then the check deposit request is routed there. If not, then the check must be routed through the Federal Reserve. Checks for small amounts are aggregated at the money centers for processing through the Federal Reserve in a batch clearing process. Large checks are handled individually as they are deposited. The Federal Reserve transfers funds as necessary. When a check reaches the branch bank holding the source account, the check either clears or bounces and the status is routed back through the system accordingly. The money-center banks maintain balances in their accounts at the Federal Reserve Bank to allow the necessary funds transfers. This model is based on our domain study of the banking system [11], and models the payment reasonably well at a gross level.

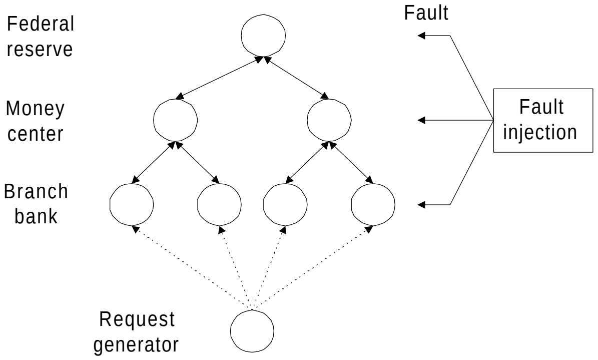

Figure 2. Payment system model and ancillary nodes

Figure 2 illustrates our experimental infrastructure model in a form simplified slightly for presentation. The actual model comprises 11 application nodes: one Federal Reserve Bank node, three money-center bank nodes, and seven branch-bank nodes. Each runs on its own computer. In addition, a request-generator node simulates bank customers depositing checks. The request generator sends check deposit requests randomly to branch banks at a specified frequency. Another node is responsible for injecting faults into the system, in this case simulating attacks on banks. Currently, the simulated intruder can launch several kinds of attacks:

- penetration of a single node,
- simultaneous penetration of several nodes either within the same bank company or on several companies, and
- penetration of several nodes within a specified time interval either within the same bank company or across several companies.

The intent of the first type of attack is to model a simple hacker scenario. The intent of the other two is to model coordinated attacks in which either one organization is the target or several are simultaneous targets.

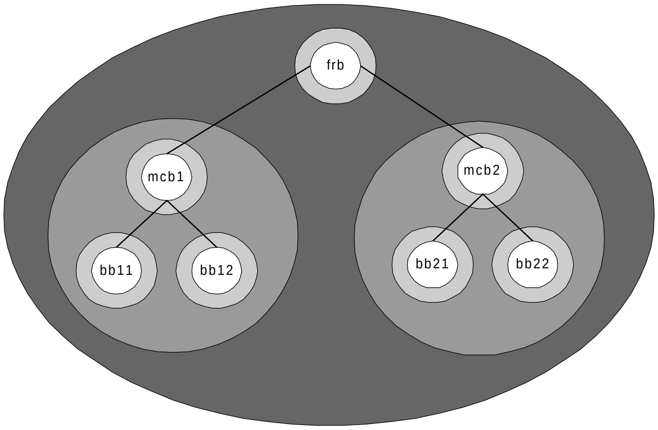

Figure 3. Superimposed hierarchical control system

5. A SURVIVABILITY CONTROL SYSTEM

To explore, develop, and evaluate control-system based survivability architectures, we have designed and implemented a prototype control system to manage our dynamic model of the banking system when it is under attack. The prototype system includes the payment system model and a control system that reconfigures the payment system in response to several types of attack.

5.1 System structure

Figure 3 illustrates the structure of the payment system and superimposed hierarchical control system, omitting the fault injection and application load generation nodes. The payment system (application) nodes are white. The bb_{ij} are branch banks. The mcb_i are money-center banks. And *frb* models the Federal Reserve Bank. Elements of the control system are depicted as circles and ovals in gray. Successively higher levels of control appear in successively darker shades. The scope of control of each level of the control system is indicated by the nesting in the diagram. Each of the model application and control system nodes is implemented as an L-node.

In our system, each bank including branch banks, money-center banks, and the Federal Reserve has a local control node to enforce policies for that bank. These nodes detect and report potential intrusions into the bank's information system and monitor communication traffic, both incoming and outgoing, for the node. Each money-center bank has a control node whose scope is the money-center bank and subordinate branch banks. This higher control level manages the system rooted at and including the money center bank. This higher-level control node communicates with subordinate control nodes, accepting reports from them and passing aggregate system-level information to them. Finally, the system has a control node whose scope is the Federal Reserve's local control node and the control nodes of the money-center banks.

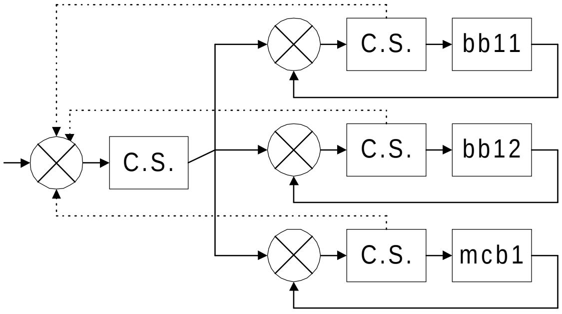

Figure 4. One level of hierarchy in the control system

In addition to communicating with higher and lower level control nodes, each control node provides a user interface at the bank at that control node's level in the hierarchy. This monitoring and control interface reports status to human management, and provides for human-initiated control actions in addition to automated control actions.

5.2 Hierarchical, distributed, multiple model control

Figure 4 gives a more detailed view of the hierarchical nature of the control system. The control system building block is the *control component*. The control system is decomposed into several layers of control components. Each controls an application node or a set of control components—a *controlled component*. The dotted lines indicate feedback from lower level to higher level controls.

A given control policy will perform well under a limited range of operating conditions. For example, an efficient but fragile configuration might be controlled under one policy, but under a different one in a less efficient but more secure configuration (e.g., in which new nodes are prohibited from entering the system).

This observation leads us to the notion of multiple-model control [14]. In traditional control theory, multiple-model control is used to partition non-linear systems into piece-wise linear systems, with each piece subject to a different analyzable control policy. To explore and illustrate this idea in the context of information-systems control, we have decomposed the operating range of our dynamic model into four regimes based on the kind of attacks with which we are concerned. We use this factor as the variable to characterize the operating regimes:

- No-attack. The system is running in its normal state.
- Single-attack. The system is experiencing scattered attacks on individual nodes. The nodes may be branch, money-center, or Federal Reserve Bank nodes.
- Regional-attack. There are coordinated attacks on multiple nodes belonging to the same company.
- Widespread-attack. The system is experiencing coordinated attacks on multiple bank nodes across several companies.

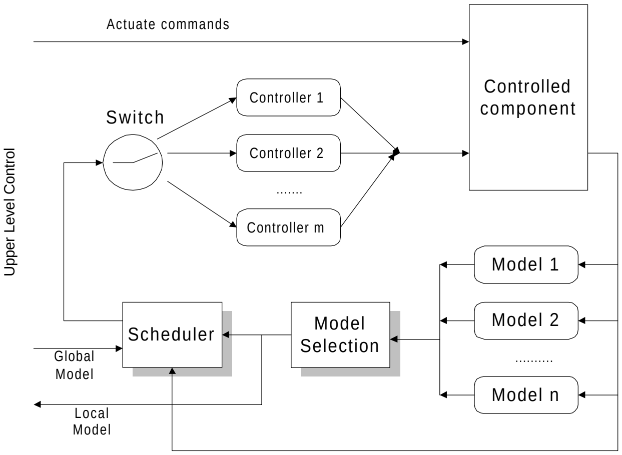

Figure 5. Structure of a control component.

For purposes of exploration, we adopt the simplistic view that loss of value is minimized by shutting down banks that are under attack, rather than letting them operate with a risk of corrupting the banking system. In the no-attack regime, controllers monitor payment system nodes. In this regime, the control system must be efficient and affect the payment system minimally. The goal is to maintain close to full service under a single attack, and to secure trusted nodes under regional or coordinated attack. The control actions for these regimes are specified as different control policies.

Multiple-model control is implemented using multiple control components running at various levels of the control hierarchy. The structure of a single control component is shown in Figure 5. A control component is composed of a set of models of the controlled system, a model selector, a set of controllers each implementing a control policy, and a controller scheduler. The controlled-component behaviors are compared against multiple models by the model selector, which determines the regime (model) in which the controlled component is currently operating. Using the selected model and other controlled component information, the controller scheduler chooses a suitable controller (control policy) to control the component.

A control component reports local model information both up and down the control hierarchy. Higher level control nodes collect information about nodes within their scope of control, form models, and propagate them to subordinates. The scheduler in each control component selects a local controller based on the global model it has, the local model of the system it controls, and information on other controlled components. The selected local controller remains active until the scheduler replaces it.

5.3 Model and policy representation

At present, our control system nodes are based on finite state machines (FSMs) with abstract-data-type interfaces. One potential advantage of this choice is that it will preserve a degree of analyzability—e.g., using model checking—not feasible with a richer computational model. However, analysis itself remains as future work. If we find that a richer computational model is needed (e.g., abstract data types or adaptive agents), encapsulation of implementations behind interfaces will ease the transition.

Each local control node is in one of the four operating regimes described above: one in which there is no attack; one in which there is a local attack on its controlled node but no other attacks elsewhere, to the best of its knowledge; one in which the money center bank to which it belongs is under attack (the Federal Reserve belongs to no such bank); and one in which the whole banking system is under attack.

Our current design is intended to permit exploration of the issues in passing information up and down the control hierarchy to enable proper switching among these modes, i.e., selection of a model of the banking system and an associated local control rule. In practice, a local control system would behave differently in different modes: perhaps reporting attacks with less provocation in the face of attacks on other nodes of the same money center bank, or even disconnecting itself from the network under severe threat, e.g., under coordinated attack on the whole system.

In our implementation, the local control nodes receive the information from parent nodes needed to drive switching among models; however, we have not yet designed control rules that use this information in a realistic way. Our local control nodes use a single policy. Each has a “sensor” logically inserted into the local banking application. In our case, the sensor models an intrusion detection system running within a banking organization (branch, money center, or Federal Reserve) that signals whether that bank is under a security attack or not. Our sensor sends *under-local-attack* and *not-under-local-attack* notifications. At a detailed implementation level, these events are sent in response to directions from the “fault injection node” in our model, which simulates effects of intruder behaviors.

The local control nodes implement a policy that responds to a *under-local-attack* status by switching the controlled bank node to an *off-line* mode, modeling a bank closure. In this mode, the banking node buffers checking requests but does not process them. When *not-under-local-attack* is detected, the control node puts the bank back in operation.

In addition to reconfiguring bank nodes, local control nodes forward *under-attack* and *not-under-attack* notifications to their parents. If two or more subordinate control nodes report *under-attack*, then the parent control node concludes that its domain is under coordinated attack, so it sends *bank-under-attack* to its subordinate control nodes, and *under-attack* to its parent. Thus, for example, if a branch bank and its money center bank both report local attacks, then the money center control node reports to the local control nodes for both the money center bank and for both branch banks that the bank as a whole is under attack.

Similarly, if the top-level control node sees two or more banks *under-attack* (e.g., the Federal Reserve and a money center bank), it sends *coordinated-attack* to its subordinate nodes, which then forward this event to their subordinates. Forwarding of *bank-attack* and *coordinate-attack* messages enable switching of models and control rules at all levels.

5.4 Implementation status

We have completed implementation of our banking system model using the L-node toolkit, and have implemented a simple control policy in a single L-node node. We are now implementing the distributed control system. Each L-node implements either application-specific (banking) or control-system-specific code.

6. INSIGHTS FROM EXPLORATORY WORK

Work to date has provided insights into research methods for information survivability, the design of applications for survivability, and the design of survivability control systems. We now discuss a number of these insights.

6.1 Application design for survivability

For an information system to be subject to control, so as to ensure continued provision of the information services on which an enterprise depends, it appears that the information system must be designed for reconfiguration. That is, the application must provide a sufficiently rich design space to provide scope for a control system to reconfigure it to handle specified adverse conditions. The need for design flexibility is reminiscent of Parnas's notion of design for ease of extension and contraction [16]. Our situation is different in at least two ways. First, survivability demands run-time not just design-time reconfiguration. Second, it demands flexibility to respond to adverse conditions related to information systems operation, not just for market segmentation or incremental delivery.

How best to determine and specify requirements for such flexibility is an open research question in our opinion. The problem appears complex. It requires an understanding of the impact on customers of service stream interruptions, how information system failures can cause interruptions, and how hazards to information systems lead to failures. Moreover, the costs of such flexibility have to be balanced against benefits, the latter of which, like insurance policies, are contingent on the flexibility being needed at some time.

6.2 Flexibility requirements analysis and specification

Information systems that run infrastructures systems should be amenable to reconfiguration under all kinds of plausible adversity. Unfortunately, an information system that has not been designed for flexibility in a specific dimension is unlikely to be flexible. The extent to which existing infrastructures were so designed in the dimensions needed for control in the face of emerging hazards and threats is unclear. Although some flexibility is obviously present, e.g., often for standard fault tolerance or disaster recovery, the ability of these systems to handle the novel and emerging threats is questionable. Some operational systems clearly were not designed or tested for such flexibility. In the future, we envision a systematic approach to the design of infrastructure information systems that integrates mechanisms which: (a) mask certain disruptive events (such as hardware failure); (b) limit certain events (such as security violations); and (c) provide design alternatives to allow controlled reconfiguration.

6.3 Subjecting legacy systems to novel forms of control

While analysis and specification of flexibility requirements appear to present significant challenges, implementing the requirements presents additional difficulties. One especially difficult problem is presented by legacy infrastructure information systems. Legacy software systems are an essential part of most infrastructures. The problem is two-fold. First, these systems were presumably not designed to have the kinds of flexibility needed in the face of novel threats. In our domain analysis of several applications we have observed such cases. Second, these systems are generally old, complex, and structurally degraded, and thus hard and costly to change—often infeasibly so because they are under tight monetary and intellectual capital-budgeting constraints. What can we do with legacy systems whose design space is poor and that cannot easily be changed?

One partial answer appears to lie in transparent extension of the design space of existing systems. To make the point concrete, consider our banking dynamic model. Our original banking nodes had operations permitting the nodes to be either on-line or not, but the nodes had no function for buffering requests during periods of suspended operation. We achieved transparent extension of the space of operating modes using the shell structure provided by our L-node mechanism. In particular, by "wrapping" the bare banking nodes behind transparent wrappers that added a buffering function, we enriched the design space enough for our control system to meet its objectives.

In a sense, then, our recipe for survivability hardening of existing legacy infrastructure information systems is first to extend (and perhaps also restrict) their design spaces using a wrapping technique; then subject the modified systems to survivability control.[1] We have demonstrated this approach in the context of a simple dynamic model. We have not proven the approach for real infrastructure systems; but wrapping is a well known and widely used technique for encapsulating and extending legacy systems. We have formulated and provided a proof of concept for a principled approach to an extremely complex problem.

---

[1] See a sister submission to this conference for related work on behavioral extension and restriction—M. Marchukov and K.J. Sullivan, *Reconciling Behavioral Mismatch through Component Restriction.*

6.4 Security of the control system

Adding complexity to a complex system in an attempt to make it better often makes it worse. This principle applies to our approach very clearly. A design that inserts into a critical system a control system able to manipulate it in dramatic ways presents an obvious risk: the control system becomes a rich target for a potential adversary.

Securing the control system thus becomes a key objective. A particularly interesting issue is that sensors that report information on the controlled system to the control system, on which the decisions of the control system are based, often run on the same platforms as the controlled system. If those platforms are vulnerable to attack, then so are the sensors. By spoofing sensor data, an adversary could mislead the control system into taking an action that serves the objectives of the adversary.

This observation has led our research team to focus on a little studied security problem: running trusted code on untrusted platforms, a problem dual to the "Java security problem" of running untrusted code on trusted platforms. We believe that, in general, there is no solution to the problem we have formulated, but that means can be used to raise the cost to spoof to a discouraging level. In practice, a broad range of security and other measures would be taken to provide defense in depth of such a control system.

6.5 Control structure determined by information flows

One of the things that we learned when taking the control systems perspective is that the information that has to be passed within a control system depends in large part on the control rules to be enforced. A policy declaring a global bank holiday if any bank is attacked requires the propagation only of a Boolean value indicating whether any bank is under attack, for example; while our richer policy requires richer flows. Thus, there is likely no one architecture for survivability control. Rather, we envision an *architectural style* for survivability control based on concepts and structures from the intellectual discipline of control theory.

6.6 Need to reason about relative dynamics

Another observation is that the dynamics of a control system have to be sufficiently faster than those of the controlled system in order for time-sensitive control rules (survivability policies) to be enforced. For example, a policy might require that a subtree be spliced out of the network before a disturbance within that subtree can propagate to other parts of the application system. Functional properties are not enough; real-time control appears likely to emerge as an important issue.

7. RELATED WORK

Control theory [1] provides a mature way of thinking about and designing information flows and feedback to maintain complex systems under desired behavioral conditions over time. For traditionally engineered systems, control theory provides a rich and beautiful set of modeling and analysis methods based on advanced mathematical analysis. At present we have in control theory a metaphor that can guide us to a novel software architectural style and to a deeper understanding of the nature of the important but inchoate concept of information survivability.

The simple control system that we presented implements a *static optimization* scheme: a precomputed policy that defines the action to take under specified circumstances. Control theory suggests an appeal to the idea of *stochastic optimal control*, with a control system using a probabilistic model of possible future conditions to choose an action that yields best *expected* results. Management of uncertainty appears to be a key problem for infrastructure suvivability. However, it is too early to know whether stochastic control has a significant role. One problem is that policy-makers might not be willing to permit probabilistic control rules.

A second problem is that it might be (and seems quite likely to be) difficult to formulate an explicit objective function (*cost*) for a control system to minimize in the tremendously complex and policy-dependent domain of infrastructure protection. Nevertheless, the metaphor seems to lead to interesting structuring techniques and to useful albeit still imprecise problem formulations.

We will thus continue to pursue connections between software design for survivable infrastructures and the clean but not always directly applicable concepts of control. In one related project, we are applying concepts from options theory—an application of stochastic optimal control, and of optimal stopping theory in particular—to reason about the value of flexibility in software products and processes [24]. In a second thrust, we appeal to the concept of economic optimization under uncertainty to reason about the nature of software evolvability [22].

The application of control systems concepts in software design is not new. Jehuda and Israeli [9] propose a control system for dynamically adapting a software configuration to accommodate varying runtime circumstances impacting on real-time performance. In contrast to our work, which leaves the objective function as a qualitative notion, Jehuda and Israeli use explicit optimization. In CHAOS [7], real time systems are adapted with the use of an entity-relation database modeling system structure. Control systems ideas have been used in distributed application management. Meta [13] is an architecture and a tool that uses a non-hierarchical control system to optimize performance in fault-tolerant distributed systems using Isis. Distributed application management (e.g., [2, 25]) employs services supporting the dynamic management of distributed applications. Network management uses control concepts to manage networks and their running software [3, 4].

However, the major objective in such work is to monitor and improve application or network performance in traditional dimensions, e.g., runtime efficiency. By contrast, our use of control is targeted at enhancing the survivability of controlled applications. Many of the control-based ideas that have been developed by others promise to contribute to our work on survivability control.

When considered from the perspective of survivability, the techniques developed in the areas of reliability, availability and security can contribute to system survivability but they are not sufficient. Techniques for achieving reliability, for example, assume different failure models and are aimed at different target applications. Similarly, security techniques are used to harden a system but typically do not provide any solution when the system is compromised.

Intrusion detection provides a way to monitor and control the abnormal behaviors of a system. EMERALD [17] introduces an approach to network surveillance, attack isolation, and automated response. It uses distributed, independently tunable surveillance and response monitors as the building blocks, and combines signature analysis with statistical profiling to provide localized protection. A recursive framework is proposed for coordinating the dissemination of analyses from the distributed monitors to provide a global detection and response capability. We address disturbances not limited to security.

GrIDS [19] is a graph based large network intrusion detection system. It collects data about computer activity and network traffic, and aggregates this information into activity graphs which reveal the causal structure of network activity. This is an intrusion detection system. No response mechanism is discussed. The graph based detection mechanism could perhaps be used in our architecture.

The Dynamic, Cooperating Boundary Controllers program [26] is developing a capability to allow traditionally static and standalone network boundary controllers (e.g. filtering routers and firewalls) to work cooperatively to protect networks. The capability is achieved through the use of an Intruder Detection and Isolation Protocol (IDIP). The work attempts to address the network intrusion problem only.

Hiltunen and Schlichting propose a model for adaptive systems [8] that respond to changes in three phases: change detection, agreement, and action. It is used for performance and fault-tolerance. Goldberg et al. discuss adaptive fault-resistant systems and present some examples [6]. Our approach provides a way to embed adaptation in the system through multiple model control. Different control policies may be adaptively used for different operating regimes.

8. CONCLUSIONS

Dealing with the fragility of critical information systems is a significant problem that must be addressed if disruptions to our everyday activities are to be prevented. That disruptions can occur is well illustrated by the many incidents that have already been reported.

Societal exposure to information systems is increasing as new applications (such as electronic commerce) are developed, as existing applications incorporate information systems to improve their efficiency, and as existing applications move from expensive closed private networks to less-expensive open Internet-based communication. The threats are also increasing. On the horizon is the prospect that critical information systems will become the targets of terrorist groups and even unfriendly foreign governments.

Dealing with disruptions that occur, no matter what the cause, requires diagnostic and corrective actions to be taken. In almost all cases, minimizing the loss of aggregate value to users and ensuring that it remains within a range required to safeguard the public interest is achieved only by taking a system-wide view.

We claim that one formalism that shows promise to aid in reasoning about this problem in infrastructure information systems is hierarchic adaptive control. In this paper, we have presented the architectural notion of survivability control systems. We have described some of the details of this architecture and illustrated the approach using a simple example derived from the banking domain. The implausibility of experimenting with actual infrastructures led us to a research methodology based on dynamic models as platforms on which to build and evaluate architectures, with room for expansion through the use of richer models.

Developing highly survivable critical information systems is not going to come about as the result of any single advance. These systems pose many challenges that will require innovation in a number of areas if they are to be addressed adequately. The control-system architectural perspective is a general framework for dealing with part of the problem.

ACKNOWLEDGEMENTS

Effort sponsored by DARPA and Rome Laboratory under grant number F30602-96-1-0314 and by the NSF under grants CCR-9502029, CCR-9506779 and CCR-9804078.

REFERENCES

1. R. Bateson, Introduction to Control System Technology, (6th ed.), Upper Saddle River, NJ: Prentice Hall, (1998).
2. 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.
3. B. Boardman, Network Management Solutions Lack Clear Leader, *Network Computing*, (August 15, 1998), 54-67.
4. Computer Associates, Enterprise Management

Strategy: Managing the New Enterprise, White paper, http://www.cai.com/products/unicent/whitepap.htm, (1996).

5. 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).

6. J. Goldberg, L. Gong, I. Greenberg, R. Clark, E. D. Jensen, K. Kim, and D. Wells, Adaptive Fault-Resistant Systems, Technical Report, SRI, (1994).

7. P. Gopinath, R. Ramnath, and K. Schwan, Data base Design for Real-Time Adaptations, *Journal of Systems and Software*, 17, (1992), 155-167.

8. M. A. Hiltunen and R. D. Schlichting, Adaptive Distributed and Fault-Tolerant Systems, *International Journal of Computer Systems and Engineering*, 11, 5, (1995), 125-133.

9. J. Jehuda and A. Israeli, Automated Meta-Control for Adaptive Real-Time Software, *Real-Time Systems*, 14, (1998), 107-134.

10. 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).

11. 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).

12. R. W. Lubinsky. Distributed Model Development System: Description and Examples, Unpublished Internal Report, Department of Computer Science, University of Virginia, (May 1998).

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

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

15. 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).

16. D. Parnas, “Designing software for ease of extension and contraction,” *IEEE Transactions on Software Engineering, SE-5,* no. 2, March 1979, pp. 128—138.

17. P. A. Porras and P. Neumann, EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances, *1997 National Information Systems Security Conference*, (October 7-10, 1997), Baltimore, Maryland. (Proceedings on the CD-ROM.)

18. President’s Commission on Critical Infrastructure Protection, Critical Foundations: Protecting America’s Infrastructures, United States Government Printing Office (GPO), No. 040-000-00699-1.

19. S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle, GrIDS - A Graph Based Intrusion Detection System for Large Networks, *Proceedings of The 19th National Information Systems Security Conference.* (October 22-25, 1996), Baltimore, MD, USA, 361-370.

20. K. Sullivan, Mediators: Easing the Design and Evolution of Integrated Systems. Ph.D Dissertation, Department of Computer Science and Engineering, University of Washington, TR UW-CSE-94-08-01, (August 1994).

21. K. Sullivan, I. J. Kalet, and D. Notkin, Evaluating the Mediator Methods: Prism as a Case Study. *IEEE Transactions on Software Engineering*. 22, 8, (1996). 563-579.

22. K. Sullivan, The Phenomenology of Software Evolution, *International Software Evolution Workshop*, Kyoto, Japan, (1998).

23. K. Sullivan, P. Shaw, and S. Geist, Mediators in Architectural Hardening of Critical Infrastructure Applications, *The 3rd International Software Architecture Workshop.* (1998).

24. K. Sullivan, S. Jha and P. Chalasani, Software Design Decisions as Real Options, *IEEE Transactions on Software Engineering*. To appear.

25. Tivoli Systems, Tivoli and Application Management, White paper, http://www.tivoli.com/o_products/html/body_map_wp.html, (1998).

26. UC Davis and Boeing Co., Intrusion Detection and Isolation Protocol (IDIP). http://www.cs.ucdavis.edu/projects/idip.html.

27. A.J. Wood and B.F. Wollenberg, *Power Generation, Operation and Control*. New York: John Wiley & Sons, 1996 (2nd ed.)