Active Trust Management for Autonomous Adaptive Survivable Systems

Howard Shrobe, Jon Doyle, and Peter Szolovits

Massachusetts Institute of Technology
Artificial Intelligence Laboratory and Laboratory for Computer Science

This is the main text of a proposal written in January 1999 and submitted in January 2000 to the Defense Advanced Research Projects Agency in response to BAA #00-15 "Information Assurance and Survivability (IA&S) of the Next Generation Information Infrastructure (NGII)"

Contents

1 Innovative Claims

The traditional approaches to building survivable systems assume a framework of absolute trust requiring a provably impenetrable and incorruptible Trusted Computing Base (TCB). Unfortunately, we don't have TCB's, and experience suggests that we never will.

We, therefore, must instead concentrate on architecting software systems to provide useful services in an imperfect environment in which any resource may have been compromised to some extent.

We believe that such systems can be built by restructuring the ways in which systems organize and perform computations. In particular,

1. Such systems will estimate to what degree and for what purposes a computer (or other computational resource) may be *trusted*, as this influences decisions about what tasks should be assigned to them, what contingencies should be provided for, and how much effort to spend watching over them.

2. Making this estimate will in turn depend on having a model of the possible ways in which a computational resource may be *compromised*.

3. This in turn will depend on having in place a system for long-term *monitoring and analysis* of the computational infrastructure which can detect patterns of activity indicative of successful attacks leading to compromise. Such a system will be capable of assimilating information from a variety of sources including both self-checking observation points within the application itself and *intrusion detection* systems.

4. The application systems will be capable of *self-monitoring and diagnosis* and capable of *adaptation* to best achieve its purposes with the available infrastructure.

5. This, in turn, depends on the ability of the application, monitoring, and control system to engage in *rational decision making* about what resources they should use in order to achieve the best ratio of expected benefit to risk.

Our claim is simple but revolutionary: "Survivable systems must make careful judgments about the trustworthiness of their computational environment and make rational decisions about strategy and resource allocation."

2 Technical Rationale

2.1 A Scenario

> Within the MIT Artificial Intelligence Laboratory an ensemble of computers runs a Visual Surveillance and Monitoring application. On January 12, 2001 several of the machines experience unusual traffic from outside the lab. Intrusion Detection systems report that several password scans were observed. Fortunately, after about 3 days of varying levels of such activity, things seem to return to normal; for another 3 weeks no unusual activity is noticed. However, at that time, one of the machines (named Harding) which is crucial to the application begins to experience unusually high load averages and the application components which run on this machine begin to receive less than the expected quality of service. The load average, degradation of service, the consumption of disk space and the amount of traffic to and from unknown outside machines continue to increase to annoying levels. Then they level off. On March 2, a second machine in the ensemble (Grant) crashes; fortunately, the application has been written in a way which allows it to adapt to unusual circumstances. The system considers whether it should migrate the computations which would normally have run on Grant to Harding; however, these computations are critical to the application. The system decides that in spite of the odd circumstances noticed on Harding earlier, it is a reasonable choice.

Did the system make a good choice? It turns out it did. The system needed to run those computations somewhere; even though Harding was loaded more heavily than expected, it still represented the best pool of available computational resources, other machines were even more heavily loaded with other critical computations of the application. But what about all the unusual activity that had been noticed on Harding? It turns out that what had, in fact, transpired is that hackers had gained access to Harding by correctly guessing a password; using this they had set up a public FTP site containing among other things pirated software and erotic imagery. They had not, in fact, gained root access. There was, therefore, no worry that the critical computations migrated to Harding would experience any further compromise. (Note: the adaptive system in this story is fictional, the compromised computers reflect an amalgam of several real incidents).

Let's suppose instead that (1) the application was being run to protect a US embassy in Africa during a period of international tension (2) that we had observed a variety of information attacks being aimed at Harding earlier on (3) that at least some of these attacks are of a type known to be occasionally effective in gaining root access to a machine like Harding and that (4) they are followed by a period of no anomalous behavior other than a periodic low volume communication with an unknown outside host. When Grant crashes, should Harding be used as the backup? In this case, the answer might well be the opposite; for it is quite possible that an intruder has gained root access to Harding; it is also possible that the intent of the intrusion is malicious and political. It is less likely, but still possible, that the periodic communication with the unknown outside host is an attempt to contact an outside control source for a "go signal" that will initiate serious spoofing of the application. Under these circumstance, it is wiser to shift the computations to a different machine in the ensemble even though it is considerably more overloaded than Harding.

What can we learn from these examples?

1. It is crucial to estimate to what degree and for what purposes a computer (or other computational resource) may be *trusted*, as this influences decisions about what tasks should be assigned to them, what contingencies should be provided for, and how much effort to spend watching over them.

2. Making this estimate depends in turn on having a model of the possible ways in which a computational resource may be *compromised*.

3. This in turn depends on having in place a system for long term *monitoring and analysis* of the computational infrastructure which can detect patterns of activity such as "a period of attacks followed by quiescence followed by increasing degradation of service". Such a system must be capable of assimilating information from a variety of sources including both self-checking observation points within the application itself and *intrusion detection* systems.

4. The application itself must be capable of *self-monitoring and diagnosis* and capable of *adaptation* so that it can best achieve its purposes with the available infrastructure.

5. This, in turn, depends on the ability of the application, monitoring, and control systems to engage in *rational decision making* about what resources they should use in order to achieve the best relation of expected benefit to risk.

Systems that can do the above things can be resilient in the face of concerted information attacks. They can carry on despite non-malicious intrusions; that is they can figure out when compromises which might be present within the infrastructure can't actually hurt them.

Our claim is simple but revolutionary: "Survivable systems make careful judgments about the trustworthiness of their computational environment and make rational resource allocation decisions accordingly."

The claim is deceptively simple: To make it real one needs to develop serious representations of the types of compromises, of the trustworthiness of a resource, and of the goals and purposes of the computational modules within an application. One also needs to build monitoring, analysis and trend detection tools and adaptive computational architectures.

Finally, one needs to find a way to make the required rational decision making computationally tractable. None of this is easy, but we have ideas and ongoing projects addressing each of these issues. The claim is also revolutionary: we note that with the single exception of the term *intrusion detection*, none of the key terms in our summary above are ordinarily talked about in the context of information survivability.

We propose to develop the science and engineering principles of building practical, resilient systems and to demonstrate them in modest applications in a testbed environment.

2.2 Trust in Survivable Systems: An Overview

> "Only the paranoid survive" - Andy Grove (everything is suspect)
> "Love your mother, but cut the cards" - folk wisdom (trust management is crucial)
> "Trust but Verify" - Ronald Reagan (always monitor)

2.2.1 Trust and rational decision making should supplant traditional notions of protection as the core concepts of Survivability

Traditional approaches to building survivable systems assume a framework of absolute trust. In this view, survivable systems require a provably impenetrable and incorruptible Trusted Computing Base (TCB). Unfortunately, we don't have TCB's, and experience suggests that we never will.

Instead, we will need to develop systems that can survive in an imperfect environment in which any resource may have been compromised to some extent. We believe that such systems can be built by restructuring the ways in which systems organize and perform computations. The central thrust of this approach is a radically different viewpoint of the trust relationships that a software system must bear to the computational resources it needs.

The traditional TCB-based approach takes a binary view of trust; computational resources either merit trust or not, and non-trusted resources should not be used. The traditional view also considers trustworthiness as a nearly static property of a resource: trust lost is never regained, short of major system reconstruction. Consequently, these systems wire decisions about how and where to perform computations into the code, making these decisions difficult to understand, and preventing the system from adapting to a changing runtime environment.

We agree with this viewpoint on the crucial role of the assessment and management of trust, but reject the assumptions about the binary, static nature of trust relationships as poor approximations to real-life computing situations. We instead base our approach on a different, more realistic set of assumptions:

1. All computational resources must be considered suspect to some degree, but the degree of trust that should be accorded to a computational resource is not static, absolute,

or known with full certainty. In particular, the degree of trustworthiness may change with further compromises or efforts at amelioration, in ways that can only be estimated on the basis of continuing experience. The system must thus continuously and actively monitor the computational environment at runtime to gather evidence about trustworthiness and to update its trust assessments.

2. Exploiting assessments of trustworthiness requires structuring computations into layers of abstract services, with many distinct instantiations of each service. These specific instantiations of a service may vary in terms of the fidelity of the answers that they provide, the conditions under which they are appropriate, and the computational resources they require. But since the resources required by each possible instantiation have varying degrees of trustworthiness, each different way of rendering the service also has a specific risk associated with it.

3. The best method for exploiting assessments of trustworthiness requires making explicit the information underlying decisions about how (and where) to perform a computation, and on formalizing this information and the method used to make the decision in a decision-theoretic framework. The overall system adapts to the dynamism of the environment and to the changing degrees of compromise in its components by deciding dynamically which approach to rendering a service provides the best likelihood of achieving the greatest benefit for the smallest risk. We do not require that the system uses explicit decision-theoretic calculations of maximal expected utility to make runtime decisions; the system may instead use the decision-theoretic formalizations to decide on policies and policy changes, which then are used to compile new code governing the relevant behaviors.

4. The system must consider selected components to be fallible, even if it currently regards them as trustworthy, and must monitor its own and component behaviors to assure that the goals of computations are reached. In the event of a breakdown, the system must first update its assessments of the trustworthiness of the computational resources employed and then select an alternative approach to achieving the goal.

2.2.2 How Active Trust Management can support Autonomous Adaptive Survivable Systems

These considerations motivate an architecture both for the overall computational environment (Active Trust Management) and for the application systems which run within it (Autonomous Adaptive Survivable Systems), which we depict in Figure 1. The environment as a whole must constantly collect and analyze data from a broad variety of sources, including the application systems, intrusion detection systems, system logs, network traffic analyzers, etc. The results of these analyses inform a "Trust Model", a probabilistic representation of the

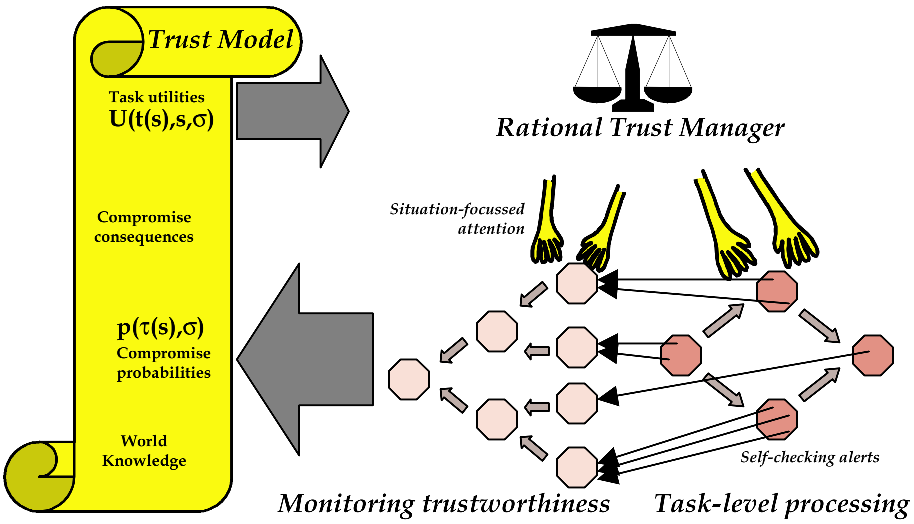

Figure 1: An Overview of Active Trust Management

trustworthiness of each computational resource in the environment. The application systems use this trust model to help decide which resources should be used to perform each major computational step; in particular, they try to choose that resource which will maximize the ratio of expected benefit to risk. This "rational decision making" facility is provided as a standard utility within the environment. The application systems also monitor the execution of their own major components, checking that expected post-conditions are achieved. If these conditions fail to hold, diagnostic services are invoked to determine the most likely cause of the failures and thereby to determine the most promising way to recover. In addition to localizing the failure, the diagnostic services can also infer that underlying elements of the computational infrastructure are likely to have been compromised and these deductions are forwarded to the monitoring and analysis components of the environment to help inform its assessments of trustworthiness. Finally, having accumulated sufficient evidence, the monitoring and analysis systems may decide that it is likely that some resource has, in fact, been compromised. This will have an immediate impact if the resource is being used to perform a computation which would be damaged by the specific form of compromise; in such cases, the monitoring and analysis components transmit "alarms" into the running application, causing it to abandon its work and to immediately initiate recovery efforts. Of course, a monitoring system which transmits such alarms too frequently is the computational equivalent of the shepherd boy who called "wolf" too often; the system again uses rational decision-making facilities to decide whether the circumstances warrant this choice.

Thus the application system forms a tight feedback control loop whose goal is to guarantee the best possible progress towards providing the services the application is intended to provide to its users (i.e., the applications are Autonomous Adaptive Survivable Systems "AASS's"). The computational infrastructure also forms a feedback control loop whose goal is to maintain an accurate assessment of the trustworthiness of the computational resources; this assessment can then inform the application systems' decision making and self-monitoring which in turn helps inform the long-term assessments of trustworthiness (Active Trust Management "ATM").

This vision leads us to focus our efforts in five major areas:

1. Models of Trust and Compromise
2. Perpetual Analytic Monitoring
3. Autonomous Adaptive Survivable Systems
4. Rational Decision Making in a Trust-Driven Environment
5. A testbed within which to experiment with these concepts

These areas of research are intimately related to two of our ongoing research efforts at MIT. The first project, the Dynamic Domain Architectures (DDA) project, is developing the runtime architectural features and the development environment facilities needed to support self-adaptivity. The second project, the Monitoring, Analysis, and Interpretation Tool Arsenal (MAITA), is developing the infrastructure needed for perpetual, analytic, and adaptive monitoring. However, neither of them has so far made it a central concern that there are malicious agents compromising resources, although the MAITA project is currently exploring applications in the Information Assurance domain. Thus each of them needs to be extended to fit within the context of Active Trust Management.

We end this section with brief sub-sections describing each of these five major areas of work. We will then return to each of these topics in more detail.

2.2.3 Trust and Compromise Models provide explicit models of the Trustworthiness of computational resources and of the forms of of their Compromise

Trust assessments necessarily involve many dimensions along which a system can be trusted, since a system might be trusted for one operation (delivery of a message) but not for another (privacy of the message), and might be trusted with one type of information but not another (as in security classification systems).

We plan to develop preliminary trust-state models quickly, but view development of models fully adequate to the task as a research topic that will require more time. Formalizing trust-state models also requires a language for making assertions about trust states, and reasoning about trust models requires effective methods for evaluating these statements. We will need to bound the language above to ensure feasible computations, and bound it below to ensure it provides the necessary utility. There are obvious relations to QoS measures that need to be taken into account.

We will explore ways of characterizing and combining the trust management protocols with other protocols, for example, mixing protocols from Quorum with trust models. This also requires developing models of the trust properties of protocols and of protocol combination methods. In some cases, combinations of protocols may be more trustworthy than the protocols being combined.

2.2.4 Perpetual Analytic Monitoring keeps the Trust Model current by detecting events and Trend Patterns which are indicative of compromise

The scenario illustrates that building a trust model involves more than just detecting an intrusion. Indeed, what was important was a template of activity patterns consisting of several temporal regions: First there was a period of attacks (particularly password scans). Then there was a "quiescent period". Then there was a period of increasing degradation of service. Finally, there was a leveling off of the degradation but at the existing high level. We call such a temporal pattern a "trend template". In our previous work we have developed a representation language for trend templates and tools for online monitoring and analysis of data streams so as to recognize instances of trend templates in the stream.

The goal of Perpetual Analytic Monitoring is to assess the trustworthiness of the computational resources in the environment. This in turn requires us to make estimates of the likelihood that a resource has been compromised in a particular way. We believe that trend templates represent a necessary tool for doing so. We intend to develop a library of trend templates that correspond to a variety of compromises of computational resources. Of course, the overall matching of trend templates depends on tools which can detect periods of uniform behavior (e.g., uniformly increasing, constant, oscillating at constant frequency). We intend to refine a highly extensible and modularized architecture for our monitoring tools which will allow a broad variety of such tools to be integrated.

Trend templates are necessary, but not sufficient in themselves. We also need to make inferences about the factual situation at hand (e.g., are international tensions rising?) and about the intentions, and states of mind of significant players (e.g., would it be likely that they are trying to attack me?). All of these inferences involve the combining of evidence to provide assessments of the likelihood of certain propositions. Bayesian networks provide a convenient formalism for representing and reasoning with basic probabilistic information. We have long exploited such models in medical decision-making problems and will use them in extending trend template models as needed.

Our scenario illustrates that monitoring mechanisms must be capable of assimilating information from a broad variety of information sources including Intrusion Detection systems, self-monitoring by the application system, system logs, network traffic analyzers, etc. We intend to create a highly modularized and extensible architecture for the data-collection portions of our monitoring system.

The principal goal of our Monitoring and Analysis tools is to keep the Trust Model current. However, when these tools have achieved a high degree of confidence that a compromise has occurred, the monitoring and analysis system must generate an alarm which may lead currently executing application components to rollback and attempt to use alternative strategies and resources. Deciding when to generate such an alarm is not trivial; if it is done too liberally then applications will never get useful work done as they service an endless stream of alarms. If it is done too conservatively, then application components will be corrupted. The decision to sound the alarm must be based on a decision theoretic analysis of the expected benefit and risks associated with raising an alarm.

2.2.5 The Autonomous Adaptive Survivable System infrastructure uses Trust Models and models of the purpose of expected behavior to select computational strategy and to detect and recover from compromises

Autonomous Adaptive Survivable Systems have the goal of adapting to the variations in their environment so as to render useful services under all conditions. In the context of Intrusion Tolerance, this means that useful services must be provided even when there have been successful information attacks.

AASS's achieve adaptivity in two ways: First, they include many alternative implementations of the major computational steps, each of which achieves the same goal but in different ways. Before each step is actually initiated, the system first assesses which of these most appropriate in light of what is known about the environment. In the Survivability context, such decisions must be rational decisions rooted in the Trust Model.

The second way in which the system achieves adaptivity is by noticing when its components fail to achieve the conditions relied on by other modules, initiating diagnostic, rollback and recovery services. This depends on effective monitoring of system performance and trustworthiness which in turn requires a structured view of the system as decomposed into modules, together with teleological annotations that identify prerequisites, post-conditions and invariant conditions of the modules. These teleological models also include links describing how the post-conditions of the modules interact to achieve the goals of the main system and the prerequisites of modules further downstream. We are already working on developing a language for representing teleological annotations, and on techniques for using these annotations to automatically construct monitors that help assess proper functioning.

Model-based diagnostic services play a key role in an AASS's ability to recover from a failure. This is done by providing models not only of the intended behavior of a component but also of its likely failure modes. These application-layer models are linked to models of the behavior of the computational infrastructure on which the application components execute. Again these include models both of expected and compromised behavior. The diagnostic task then is to identify the most likely set of such models which is consistent with the observed behavior. This helps the application decide how to recover from the failure and restore normal functioning. It also provides evidence to the overall monitoring environment about the trustworthiness of the underlying computational resources, particularly when the most likely diagnosis is that one of the resources has been compromised.

In addition to component selection and diagnosis services, the Autonomous Adaptive Survivable Systems infrastructure provides a variety of other services including rollback/recovery and resource allocation.

2.2.6 Rational Decision Making uses decision-theoretic models and the Trust Model to control decisions about component selection and resource allocation

We assess system trustworthiness and performance according to the trust and teleological models in order to make decisions about how to allocate computational resources. To ensure that these decisions represent a good basis for system operation, we will develop detailed decision-theoretic models of trustworthiness, suspicion, and related concepts as applied to information systems and their components. These models will relate notions such as attractiveness of a system as a target, likelihood of being attacked, likelihood of being compromised by an attack, riskiness of use of the system, importance or criticality of the system for different purposes, etc.

The models will also relate estimates of system properties to an adaptive system of decision-theoretic preferences that express the values guiding the operation of both system modules and the system as a whole. We will develop mechanisms that use these elements to allocate system resources optimally given task demands, trustworthiness judgments, and the resources available.

2.2.7 A Testbed

The preceding description of our main ideas has been at a somewhat abstract level. Many important issues can only be addressed within the context of a real implementation. To what degree should facilities be replicated? How do we protect the key components of our proposed infrastructure from being compromised themselves? To address these issues we plan to configure a small ensemble of computers running a distributed application similar to that described in our scenario. We will develop this application and the computational infrastructure along the lines described above and experiment with the variety of design options. Fortunately (in a manner of speaking), we are blessed by a never-ending stream of attempts by hackers to compromise our computational environment, so we will get some "red teaming" for free.

2.3 Models of States of Trust and Compromise

Making rational decisions about how to use resources in an environment of imperfect trust requires information about what resources can be trusted, and for what purposes. We propose to develop models of trust states that go beyond mere information about whether or how a system has been subject to attack to represent whether or how different properties of the system have been compromised, and finally to represent whether they can be trusted for a particular purpose even if compromised. We also represent the degree to which these judgments should be suspected or monitored.

These models provide the point of intersection among all the other elements of the proposed approach. The dynamic domain architecture automatically configures software components to produce the monitoring data that the active monitors need to maintain the accuracy of the trust models on which rational resource allocation decisions depend.

Indeed, trust plays a central role in resource allocation decisions. All decisions about what to do must be based on beliefs about the situation in which the action is to be taken. We can think of the degree of trust one places in a system as the degree to which one is willing to rely on the proper functioning of the system without also dedicating unusual effort to preparing for the contingency of failure. Since preparations for contingencies consume resources, this makes trust management a central resource allocation issue.

The trust model is organized into three levels above that of raw behavior:

1. The lowest level of the trust model represents the results of initial interpretations such as *attacks* and *anomalous behavior*. At this level we collect, filter and organize the necessary information so that it can trigger trend templates and feed into Bayesian inference networks. As we saw in our scenarios at earlier, we are not primarily interested in what attacks or anomalous behaviors have taken place, but rather in what they imply about what compromises might actually be present. We plan to draw on the work of the intrusion detection community and others to help form annotated taxonomies of attack types and to determine how these feed into the next layer of the model.

   Current intrusion detection efforts have developed a taxonomy of types of attacks, summarized in the `AttackID` descriptor from the Common Intrusion Specification Language (CISL) draft of October 16, 1998 prepared for the Common Intrusion Detection Framework (CDIF; see http://seclab.cs.ucdavis.edu/cidf/). There are four basic categories of attack in the common vocabulary already: penetration, denial of service, unusual access, and flooding, with numerous specific types under these. The CISL also provides a vocabulary for describing various actions occurring in the course of attacks, such as copying, moving, and deleting of files, together with some informal information about causality among different events.

2. The middle level of the trust model deals with *compromises*. The attack level only tells us that malicious or anomalous activity has taken place. But what we are interested

in is whether someone has actually succeeded in an attack and has used that to exploit or corrupt resources. That such a compromise has been occurred can be inferred by matching the temporal patterns of activity to a trend template for a particular compromise. In the scenario we saw an example of this in which the gaining of unauthorized user level access was indicated by the temporal pattern of password sweeps followed by quiescence followed by increasing resource consumption.

The categorization of compromise states is relatively virgin territory, and one of the main aims of this research. We plan to attack this initially by first of all considering the properties that standard security techniques protect, such as the standard concepts of privacy, integrity, authentication, and non-repudiation. Within each of these dimensions, one can identify a variety of finer compromise types. For example, within the privacy dimension, one might distinguish knowledge of passwords from knowledge of the secrets these passwords protect, and from knowledge of the activity patterns of the authorized users of the secrets. One might distinguish visibility of these secrets to everyone from visibility to a narrower group. Within the integrity dimension, one can distinguish incompleteness from incorrectness.

A second dimension involves operational properties, essentially the different factors that make up quality of service. These properties include rate, timeliness (lack of delay), and evenness (freedom from jitter).

A third dimension involves command and control properties, such as the degree of confidence, the observability of security and operational properties, and the degree of controllability. This dimension is not covered by the first two sets. For example, loss of the ability to control the operational system might come about through a denial of service attack or other compromises of the control system, and loss of observability might come about through attacks on the monitoring system.

A fourth dimension distinguishes different subsystems, such as the intra-system communications network, the primary operational processes, the security mechanisms, or the monitoring subsystem. These distinctions may be made even more finely, to distinguish the types of information, operations, information sources or destinations affected, and the identity and roles of people participating in or affected by the compromises. (In this last situation, one possible compromise might be that someone inside is leaking passwords or other access information).

However, we believe that all these will require careful formalization and will need to be refined carefully. We can also gain some leverage on this issue by studying the properties which our testbed application requires in fine detail.

3. The highest level of the trust model deals with *trustworthiness.* The fact that a resource has been compromised does not in and of itself imply that it is totally unsafe to utilize

it. That conclusion depends on the precise way in which the consumer wants to utilize the resource as well as on assessments of the *intention* of the compromiser. In our scenarios, we presented two different examples: in the first the system was compromised by "teenaged hackers" looking for free resources, in the second it was compromised by state-sponsored malicious agents. Clearly, we should generally be more wary of using a resource in the second case than the first; but if we are not very sensitive to quality of service and perhaps only care about the integrity of our data, then the first case is not all that risky.

Knowledge of attack types mainly guides the organization's attempts to defend against future attacks. Knowledge of compromises indicates the threats to operations. Knowledge of trust states guides how the organization carries on in the face of partially-understood compromises. Because intent plays a central role, it too must be modeled throughout the three layers, moving from raw reports about behavior at the base level, to statements about intent in the middle layer and finally entering into assessments of trustworthiness at the highest level. We have built up considerable experience through our projects in the HPKB and IC&V programs in evidential reasoning about political intent that should transfer into this aspect of the model.

Although it is not the central goal of our project to reason about whether a national scale information attack is ongoing, we believe that our work has great relevance to this question. One plausible approach to this problem involves using "plan recognition" to recognize the difference between a coordinated national attack and widespread hacking. But plan recognition works by recognizing the tactics which represent methods for achieving a sub-goal of the plan. These sub-goals are precisely the *compromising* of certain resources which can serve as the platform for their exploitation in compromising yet other more important resources. Thus our work on modeling and recognizing compromises is essential groundwork for strategic intrusion assessment.

2.4 Adaptive Systems: Dynamic Domain Architectures for Autonomous Adaptive Survivable Systems

> "If you only know one way to do a task, you'll eventually discover a situation in which you don't know how to do the task at all" - Marvin Minsky
> "If at First you Don't Succeed, Try Try Again" - Ben Franklin

The Autonomous Adaptive Survivable Systems component of this project will extend our Dynamic Domain Architecture work. The basic insight of this work is that adaptivity can be realized by capitalizing on the "variability within commonality" which is typical of the service layers in any software domain. A Dynamic Domain Architecture structures a domain into service layers; each service is annotated with specifications and descriptions of how it

Figure 2: Dynamic Domain Architectures provide for Intrusion Tolerance by coordinated efforts in the Development and Runtime Environments

is implemented by the services at lower levels. However, a Dynamic Domain Architecture provides multiple instantiations of each service, with each instantiation optimized for different purposes. In a Dynamic Domain Architecture, all the alternative instantiations of each service, plus the annotations describing them are present in the run-time environment.

Dynamic Domain Architectures late bind the decision of which alternative instantiation of a service to employ. We may view this as an extremely dynamic and information rich version of OOP in which method invocation is done in decision-theoretic terms, i.e., we invoke that method most likely to succeed given the current trust state. The possibility of coordinated intentional information attacks makes it necessary to add trustworthiness as a primary concern in this process.

Dynamic Domain Architectures recognize that in many open environments (e.g., image processing for ATR) it is not possible to find *a priori* grounds for selecting the correct operator with precision; in the context of information survivability, there is the additional complexity that the best operator may be corrupted or otherwise untrustworthy. Therefore, Dynamic Domain Architectures support an even later binding of operator selection, allowing this initial selection to be revised in light of the actual effect of the invocation. If the method chosen doesn't do the job well enough, alternatives selections are explored.

Dynamic Domain Architectures remove exception handling from the purview of the programmer, instead treating the management of exceptional conditions as a special service provided by the run-time environment. The annotations carried forward to run time include formal statements of conditions which should be true at various points of the program if it is to achieve its goals. The architecture provides monitors which invoke error-handling services if the conditions fail to be true. The exception-management service is informed by the Dynamic Domain Architecture's models of the executing software system and by a catalog of breakdown conditions and their repairs; using these it diagnoses the breakdown, determines an appropriate scope of repair; possibly selects an alternative to that invoked already and then restarts the computation.

In summary, a dynamic domain architecture provides the following services:

1. Synthesis of code that selects which variant of an abstract operator is appropriate in light of run-time conditions.
2. Synthesis of monitors that check whether conditions expected to be true at various points in the execution of a computation are in fact true.
3. Diagnosis and isolation services that locate the cause of an exceptional condition, and characterize the form of the breakdown which has transpired.
4. Selection of alternative methods that achieve the goal of the failed computation using

different means (either by trying repairs or by trying alternative implementations, or both).

5. Rollback and recovery services that establish a consistent state of the computation from which to attempt the alternative strategy.
6. Reallocation and re-optimization of resource allocations in light of the resources remaining after the breakdown and the priorities obtaining at that point. These services may optimize the system in a new way in light of the new allocations and priorities.

A key component of this overall structure is the diagnostic service, based on our earlier work [2], [13] and the work at Xerox Parc [3], [27]. This work all deals with diagnosing physical faults in hardware, our plan is to extend these techniques to handle the diagnosis of software failures which are in turn used as the symptoms of compromise of the underlying computing infrasturcture (including the OS and the hardware).

Model-Based Diagnosis works by using models of the structure and intended behavior of an artifact's components to predict the overall behavior. When the diverges from observation, a symptom has been detected and dependency records maintained during the prediction phase can quickly identify set of components which cannot all be functioning correctly given the observed behavior. In addition to modeling the "functional properties" of the components, we also model quality of service properties (e.g., delay) because these are particularly easy to reason about.

Each component is provided with multiple models, one for the normal behavior, one for each known abnormality (e.g., unusual long delay, or unusually short delay) and a final "null model" to cover all other anomalous behavior. The behavior of the software components, however, is dependent on whether the underlying system has been compromised; to model this we also provide both normal and abnormal behavioral models for each underlying computational resource, modeling the possible compromised states of that resources (including a model for "unknown anomalies"). These are connected to the software component models by conditional probabilities which estimate how likely the software would be to fail in that particular way, given that the underlying resource (the Scheduler of the OS for example) has been compromised in a particular way. This network of models at two levels connected by conditional probabilities forms a Bayesian network through which evidence (e.g., software component misbehavior) can influence our estimate of the likelihood of underlying causes (e.g., compromised OS). The diagnostic service chooses the most likely set of behavioral models which are consistent with the observations, in the process updating the Compromise Layer of the Trust Model.

2.5 Perpetual Analytic Monitoring: adaptive monitoring for active trust management

Trust monitoring is perpetual, analytic, and active. Trust monitoring is perpetual because eternal vigilance is the price of survivability. Every defense becomes, when discovered by adversaries, a spur to invention of a new offense. Trust monitoring is analytic because the raw data received from instrumented computational modules must be analyzed at a number of levels to determine the significance of the observed overt events. Trust monitoring is also adaptive. The monitoring system will expend more effort during a time of frequent attacks than during a period of happy isolation, and may spend less during frequent attacks if system capabilities have been degraded sufficiently to place such limits on the monitoring effort.

The Perpetual Analytic Monitoring component of this project will extend the MAITA system and integrate its mechanisms with the Dynamic Domain Architecture. The MAITA system began development for rapid, knowledge-based construction of monitoring systems under DARPA's High Performance Knowledge Bases program. The MAITA system consists of a library of monitoring methods, an architecture for operating networks of monitoring processes, and a flexible, display-oriented control system for quickly constructing, composing, modifying, inspecting, and controlling monitoring networks. See [6] for more details.

2.5.1 A library of recognition methods

The MAITA system provides an extensible and growing knowledge-based library of monitoring methods and components, already annotated with structural and teleological information, which system developers can use to construct new monitoring systems.

The monitoring method library includes entries at all levels of computational detail, from very abstract procedures covering virtually all monitoring tasks, to intermediate-detail procedures capturing more specific algorithmic ideas, information about the class of domain or enclave, or types of information being monitored, to highly detailed procedures involving specific representations, code, domain details, and signal sources. For example, the most abstract levels speaks of constructing and comparing a set of hypotheses about what is going on, without providing any details about how the hypotheses are constructed or compared. At an intermediate level, the $\mathrm{TrenD_x}$ [12, 11, 18, 15, 9] trend monitoring system developed at MIT uses a partial-match strategy operating over a set of trend templates, each of which consists primarily of temporal constraints characterizing some temporal event. More refined monitoring models emend this procedure to take probabilistic or default information into account, or to embed background knowledge of the domain in the matching strategy. The most abstract control and interpretation procedures serve as a base for more specific ones.

We will construct a library of abstract and special signal correlators called *trend templates*, after the representation by that name developed at MIT by Haimowitz and Kohane in the $\mathrm{TrenD_x}$ system [12, 11, 18, 15, 9]. A trend template (TT) is an archetypal pattern of data variation in a related collection of data. Our earlier explanation of the pattern illustrated in the scenario is an example of such.

Each TT has a temporal component and a value component. The temporal component includes landmark time points and intervals. Landmark points represent significant events in the lifetime of the monitored process. They may be uncertain in time, and so are represented with time ranges (min max) expressing the minimal and maximal times between them. Intervals represent periods of the process that are significant interpretation. Intervals consist of begin and end points whose times are declared either as offsets of the form (min max) from a landmark point, or as offsets of the form (min max) from another interval's begin or end point. Value changes are described in terms of various standard types of curves, and we are expanding the original representation to include statistical information as well. The representation is supported by a temporal utility package (TUP) that propagates temporal bound inferences among related points and intervals [17, 16]. The value component characterizes constraints on individual data values and propositions and on computed trends in time-ordered data, and specifies constraints that must hold among different data streams.

In matching a trend template to data, two tasks are carried out simultaneously. First, the bounds on time intervals mentioned in the TT are refined so that the data best fits the TT. For example, a TT that looks for a linear rise in a numeric parameter followed by its holding steady while another parameter decays exponentially must find the (approximate) time boundary between these two conditions. Its best estimate will minimize deviations from the constraints. Second, an overall measure of the quality of fit is computed from the deviations. The most appropriate language of trends and constraints will vary from domain to domain, and we expect to build a rich set of capabilities to populate the ontology of trends.

We plan to develop TTs for intrusion detection, compromise identification, and a library of processes for monitoring trustworthiness and suspiciousness of system components and behaviors. Some of the elementary processes in the library will be based on the relationships between trust concepts as delineated in models of trust states and trust-based resource allocation methods. Other library elements will correspond to techniques used in extant system security methods, such as boundary controller patterns, intrusion and misuse signatures, and statistical models of normal, abnormal, and anomalous behaviors. Obviously this relies on a lower level of signal processing techniques, such as standard transforms and correlators.

2.5.2 Monitoring for trust management

The central concept in the MAITA architecture is that of a network of monitoring processes, operating under the control of a "monitor of monitors" or "MOM". The set of monitoring processes form the nodes of the network, and the communication paths form the edges or links of the network. Each process in the network may have a number of "terminals", each of which receives or emits streams of reports via one or more standard transmission protocols.

The network may exhibit a hierarchical structure, as some monitoring processes may consist of a subnetwork of subprocesses. The metaphor we used for thinking of the operation of these networks is that of electrical networks, in which we “wire together” various processes and network fragments by connecting their terminals together. A MOM provides means for constructing, maintaining, inspecting, and modifying the monitoring network and its operation. We achieve a degree of uniformity in the control process by organizing MOMs as special types of monitoring processes. We provide some standard programming classes that act as wrappers, for adding MAITA terminals and functionality to application systems.

We plan to integrate the MAITA architecture with the Dynamic Domain Architecture in order to further reduce the effort of constructing and tailoring monitors. Our approach will be to use the DDA itself to automatically configure the backbone monitoring systems. Specifically, the DDA will be automatically configuring software modules to produce the data streams needed for monitoring. To make use of these streams, the DDA will also configure the network of monitoring to make use of these streams, both to feed into resource allocation decisions and to feed back into the operations of the software components themselves.

The first step of this integration modifies the DDA to add MAITA functionality to each instrumented software module. The second step calls on the DDA to set up those monitoring processes needed to analyze the instrumentation data and to maintain the trust models. The third step is to tailor the configuration of alerting models used in the monitoring process to conform to resource allocation decisions made by the trust-management system, so as to not overwhelm the resources with monitoring alerts.

2.6 Rational Trust Management

The aim of managing resources in an environment of imperfect trust is to find an allocation of resources to tasks that maximizes overall utility. This includes allocations of resources to carry out the primary tasks, and also allocations of resources to ensure survivability through monitoring, replication, reconfiguration, etc. Since an easy way to deny service is to feint attacks that convince a system to expend all its resources on defensive measures, the primary aim of rational trust management is to allocate resources to tasks in the face of compromise, and to balance resources spent on defense and survivability against the effects of these expenditures on the primary operational tasks. Doing this requires development of methods for making allocation decisions that take information about degree of trustworthiness into account. Our approach to these decisions seeks to formulate the decisions in decision-theoretic, terms, in order to identify rational allocations as ones of maximal expected utility. Since it is not always possible to compute the rational decisions rapidly, we also plan to explore various means for making rational decisions in practice, ranging from rule-based policies justified in decision-theoretic terms to approximation algorithms for computing good-enough allocations.

2.6.1 Trust-based resource allocation

Trust-based resource allocations form a special case of resource allocations in general. There is a sizable literature on resource allocation decision, especially in the financial realm, such as methods for allocating investments across investment vehicles. These models are all based on base notions of probability of performance and utility of performance. Trust-based models will also involve these variables, but will modulate the judgments of probability and utility by the degree of trustworthiness, suspiciousness, etc., of the resources involved.

In the simplest case, we might view total allocation utility in a state of the world σ as simply the sum of the utility of each allocation of tasks to systems, i.e.,

$$\sum_s EU(t(s), s, \sigma)$$

where $EU(t(s), s, \sigma)$ denotes the expected utility of having system s perform the task $t(s)$ in state σ. In this simplest case, we presume only one task assigned to a system at a time; in the more natural case of multiprocessing systems, the expected utility would accord to the complete set of tasks assigned to each system. The separability of utility of assignments to different systems also represents a potential target for improving reality, since in some cases the overall utility must take into account the entire allocation across systems, not just sum the individual contributions. For example, purely local measures might call for redundancy, assigning the same process to several systems. But this may be unacceptable if there is a larger set of processes that must be performed than the servers left after the duplications used for redundant computing. Nevertheless, the local utility estimations provide a good and feasible starting point for investigation of the effect of trust on resource allocations.

In this simplified setting, the expected utility of a task assignment, $EU(t(s), s, \sigma)$, is itself expressed by

$$\sum_\tau Pr(\tau(s)|\sigma)\, U(t(s), s, \sigma)$$

where $Pr(\tau(s)|\sigma)$ is the probability that trust state τ obtains for system s given that the world state is σ, and where $U(t(s), s, \sigma)$ is the utility of performing task $t(s)$ in system s in the world state σ.

In some cases, we may be able to decompose the utility $U(t(s), s, \sigma)$ into factors depending on the task, trust state, and the world state, for example,

$$U(t(s), s, \sigma) = U_1(t(s), \tau(s)) \cdot U_2(\tau(s), \sigma)$$

Such decompositions may simplify the task of estimating these utilities, and part of our research on trust models will be to investigate the practicality and realism of such decompositions.

We plan to refine and elaborate these initial and simplified decision-theoretic models for trust-based resource allocation. One task here is to identify situations in which the simplifying assumptions hold true. We also will seek to find models appropriate to more complicated situations that cannot be simplified in these ways. Finally, we will develop a range of concrete models for how the rich and varied structures of trust states influence the probabilities and utilities in these abstract models.

2.6.2 Decisions affecting resource consumption

Resource allocations must address both gross static allocations, in which the system allocates processes to hosts based on their expected CPU, memory, and network requirements, but also finer-grained, dynamic controls on the processes that affect their expected resource consumption. For example, monitoring systems that produce floods of false alerts may consume bandwidth and processing on the part of the recipients that the system cannot spare.

We will seek to apply our ongoing research on qualitative representations of preference information to the trust-management task. These representations characterize preferences among classes of situation or object rather than the usual preferences among individuals that form the foundations of standard decision theories. The primary applications in trust management are to representing the alerting preferences of monitoring processes, so as to be able to easily control the flow and quality of alerts generated, and to representing the resource allocation preferences.

We will build on our past work [25, 7, 26, 8] on qualitative representation of utility information, which has developed logical languages that can express generic preferences ("prefer air campaign plans that maintain a center of gravity over those that distribute forces more widely"), and that relate this notion of preference to the notion of problem-solving or planning goals (interpreting goals as conditions preferred to their opposites, other things being equal). We will develop utility models that combine both qualitative preference information with approximate numerical models of common utility structures (e.g., utility models that increase up to some time and then drop off to model deadline goals, as in [10]), along with automatic procedures for combining such information into qualitative decision procedures and numerical multiattribute utility functions suitable for quick evaluation of alternatives.

2.6.3 Practical rational decision making

We expect to employ different decision-making procedures for different decisions within operational, monitoring, and control processes. Some decisions, such as those made by higher-level monitoring analysis process that can afford to take some time in coming to decisions, may use direct decision-theoretic computations. For these we will use implementations based on Bayesian probabilistic networks and influence diagram representations, and make use of the latest algorithms. Such algorithms are entirely adequate to many specific problems, and a large class of monitoring library entries will be focussed on specific problems.

Other decision problems will require other methods, for they will involve probabilistic models of a scale that makes the standard methods infeasible, or will involve distributed decisions difficult to cast in the form of a centralized decision problem. We will explore a number of approaches to practical rational decision making, each based on existing progress. These include methods based on derivation of decision policies or rules from the decision model; approximation algorithms that work to improve initial decisions as much as possible given the available time; determine dominance relations and other conditions that greatly reduce the complexity of a decision problem in advance, permitting the runtime decision to be made by checking a few class-membership questions.

In addition to these direct approximation methods, we will explore distributed resource allocation methods that seek to reduce the complexity of decision-making tasks by dividing them up according to specialized interests of different system components. To do this, we will build on our previous DARPA-funded research on market-guided reasoning and planning [4, 5]. The main goods in these economies represented computational resources (time, memory, etc.) and the achievement of different goals; the producers represented problem-solving or planning methods; and the consumers represented goals (external demands or internal subgoals). This work and experience carries over fairly directly to the task of allocating and reallocating large-scale survivability resources in the face of changing resources and demands.

3 Comparison with Related Research

Current systems for intrusion detection focus on developing two basic types of mechanisms, plus hybrid methods combining these. The first basic type is that of signature recognition, in which a specific pattern of events signals a possible intrusion. The second basic type is that of statistical anomaly detection, in which comparison of system, user, or operation statistics across different timescales and from different time periods identify the existence of potential intrusions. The knowledge embodied by these systems tends to focus on fairly low levels of signals. In the Emerald system, for example, rules tend to be simple and statistical methods tend to be prominent at the lowest levels of the monitoring hierarchy; at higher levels, rules become more complex, while the contributions of statistical methods diminish. The main bodies of knowledge for rules concern signature of varying degrees of complexity, but most represent fairly local considerations, rather than the wider situational awareness we seek to capture in the proposed monitoring knowledge libraries. The statistical models may be constructed manually or constructed and adapted mechanically, but most are based on fairly gross properties of the system being monitored. Our proposed work develops methods for basing monitors on more complicated statistical and probabilistic models, using Bayesian networks involving terms related to concepts in the situation knowledge base, and exhibiting situational dependence in which the probabilistic network used may itself be changed as the situation changes.

Specific monitoring systems, as opposed to codifications of libraries of knowledge for constructing monitoring systems, are well represented in the literature and in commercial products. The most relevant work, other than our own, on monitoring knowledge and methods appears in the literature on trend detection and "temporal abstraction", especially in the work of Shahar [21] and Das [1] at Stanford. These efforts focus on representing temporal relationships and on methods for identifying patterns of temporal relationships as instances of more abstract events. This work provides a good foundation for monitoring and analysis activities, but intelligent monitoring and analysis involve more than just temporal information. Structuring relevant sorts of non-temporal information, especially information about logical implication, statistical correlations, and causation, is crucial, but lacking in most abstraction-based treatments. Statistical trend detection, on the other hand, does not adequately exploit the constraints and structuring information that templates provide. We plan to design representations for monitoring conditions that integrate the best representations devised for each of these separate types of knowledge.

Artificial intelligence and computer science have made use of economic and decision-theoretic models for resource allocation for some time, notably the SPAWN system of Waldspurger et al. [23] for allocating processor time across workstations, work on rational negotiation and auction structures by Rosenschein [20], Zlotkin [29], Kraus, Sandholm and others, and the work of Wellman and his students on the WALRAS [24] artificial economy and its applications. None of these have addressed methods for trust-based resource allocations, however. There also has been significant work done on rational decision making under time and resource constraints, notably in the family of notions of anytime algorithms, flexible computation, and bounded optimality introduced by Horvits, Dean, Boddy, Russell, Zilberstein, and others. We expect that some of these mechanisms will prove suitable for some of the trust-based survivability decisions, though these methods have not been studied very thoroughly in the setting of distributed action.

Our work on Dynamic Domain Architectures draws on much previous research on software representations and Model-Based Troubleshooting. However, the integration of these two threads is completely novel to our project. The application of these techniques to Information Survivability is also novel to our project.

There is a considerable body of work on model-based diagnosis (see e.g., [14] for a good overview) that provides the foundation for the work proposed here. In that body of work the recent efforts by Williams are particularly relevant and offer good comparison points. Earlier work in our group at MIT [2] and at Xerox Parc [3] established the basic framework for Model Based Troubleshooting. In particular, the idea of modeling both expected and known faulty behavior comes from the work at Parc. Later work at MIT [13] first demonstrated that with well-chosen abstractions one can apply these techniques to real-world artifacts of significant complexity.

Brian Williams's work at NASA [28] is an excellent recent development that is closest to the variety of task being examined here. Williams offers an existence proof that model-based diagnosis can be used on systems that are complex hybrids of hardware and software with time-varying histories of behavior, and that the same framework also supports reconfiguration. We should note that Brian Williams has recently joined our Faculty and is will aid in this effort.

None of these efforts, however, deal with the context of a coordinated attach by an intentional malicious agent. This extra complexity leads us to a multi-tiered modeling framework in which the observable behavior (and misbehavior) of the software components are modeled at one level and the behavior (and compromised states) of the underlying infrastructure are modeled at another.

We intend to draw heavily on our earlier Programmer's Apprentice for representations of software systems [19]. This work established an important set of abstractions which have shown up in later work on software architectures, in particular, modeling in terms of abstract components and connectors. It also introduced a represenation of dependencies between modules (purpose links) which seems to be a good starting point for generating wrappers for monitoring the intended behavior of components. Of course, this was quite preliminary work and it was not intended to deal with the complexities posed by the information survivability context.

4 List of key personnel

Howard Elliot Shrobe, Ph.D., is Associate Director of the M.I.T Artificial Intelligence Laboratory. He has done research in Software Engineering, VLSI design, Computer Architecture and Artificial Intelligence. He served as Chief Scientist of the DARPA Software and Intelligent Systems Technology Office (SISTO) and of the Information Technology Office (ITO) from September 1994 through August 1997. In this capacity he had a significant role in the creation of several programs including EDCS and Information Survivability. He has also served as Chief Technology Officer of Symbolics Inc. He served as Chair of the AAAI Conference Committee for 5 years and is a Fellow of the AAAI.

Additional key personnel identified at this point include Drs. Jon Doyle, William Long, and Professor Peter Szolovits of the MIT Laboratory for Computer Science.

Jon Doyle, Ph.D., will devote 50% of his time to this effort in different years of the covered period. His principal research goal is to develop theories and techniques for representation and reasoning that have a sound basis in decision theory, economics, and logic, and to apply these to practical problems of planning and medical informatics, especially to the representation and use of qualitative and quantitative models of preferences and utilities. He has published widely on the roles that economic notions play in the structure of reasoning and representations, and together with his students, has conducted investigations into means for mechanizing rational reasoning. Dr. Doyle has made many contributions to the theory of rational and economic reasoning, has developed representations for rational agents and market-guided reasoning systems, and has worked on structuring ontologies for planning, the process of planning, and general medical health-maintenance monitoring, all under previous and current DARPA funding. He currently serves as PI of our work on the MAITA system. He is a Fellow and member of the Executive Council of the American Association for Artificial Intelligence, as well as a director of Principles of Knowledge Representation and Reasoning, Inc., which organizes the KR conferences.

William Long, Ph.D., will devote 25% of his time to this effort in each covered year. He is involved in research in causal and temporal reasoning. This work over the past dozen years has been focused on the cardiology domain and the diagnosis of heart disease. This is a rich domain for causal reasoning because the underlying mechanisms take from seconds to years and the challenge is to fit the findings into a consistent, plausible scenario in time. The strategies for generating and evaluating such causal hypotheses will be useful in the proposed work. Dr. Long has also been involved in the use of classification trees and neural networks for the development of classification tools from data sets. While this was also done in a medical context (detection of cardiac ischemia in the emergency room), the methodology is applicable in a wide range of domains. The third area of expertise is the detection of trends. This work has been carried out in the context of managing therapy, both the adjustment of digitalis and the management of ventricular arrhythmias. Dr. Long is a Fellow of the American College of Medical Informatics. He currently serves as a key researcher in the MAITA project.

Peter Szolovits, Ph.D., will devote 15% of his time to this effort in each contract year. He has worked on problems of knowledge representation, reasoning under uncertainty, and diagnostic and therapeutic planning and monitoring, mostly in applications to medical decision making. He and his students pioneered diagnostic reasoning methods that rely on detailed models of causality and temporal relationships among aspects of a hypothesized disorder, and investigated multi-level reasoning systems that pursue a simple analysis of a problem when all data consistently indicate a single solution but that engage in much more detailed analyses when discrepancies arise between data and expectations. Prof. Szolovits is currently participating in the development of a new architecture for medical record systems that exploit the technologies of the World Wide Web to support sharing and commonality of access to records from multiple institutions, and is engaged in building life-long active patient-centered health information systems that orient medical information processing, decision making, health and treatment monitoring, task-specific education and communication around the individual patient. This project, called Guardian Angel, was begun with DARPA support [22]. Experience from these efforts motivates and contributes to the design of the present proposal, and advances in the ability to support monitoring and analysis tasks will greatly contribute to the future of these projects as well as to applications in the challenge domains. Prof. Szolovits is a fellow of the American Association for Artificial Intelligence and of the American College of Medical Informatics. He currently serves as co-PI of the

MAITA project.

References

[1] A. K. Das and M. A. Musen. A comparison of the temporal expressiveness of three database query methods. In *Nineteenth Annual Symposium on Computer Applications in Medical Care*, pages 331–337, New Orleans, LA, 1995.

[2] R. Davis, H. Shrobe, W. Hamscher, K. Wieckert, M Shirley, S. Polit. Diagnosis Based on Descriptions of Structure and Function. In *AAAI, National Conference on Artificial Intelligence* Pittsburgh, PA., 1992 pp 137-142.

[3] J. deKleer and B. Williams. Reasoning About Multiple Faults. In *AAAI, National Conference on Artificial Intelligence*, Philadelphia, Pa., 1986, pp 132-139.

[4] J. Doyle. A reasoning economy for planning and replanning. In M. H. Burstein, editor, *Proceedings of the 1994 ARPA/Rome Laboratory Knowledge-Based Planning and Scheduling Initiative Workshop*, pages 35–43, San Francisco, CA, 1994. Morgan Kaufmann.

[5] J. Doyle. Toward rational planning and replanning: rational reason maintenance, reasoning economies, and qualitative preferences. In A. Tate, editor, *Advanced Planning Technology: Technological Achievements of the ARPA/Rome Laboratory Planning Initiative*, pages 130–135. AAAI Press, Menlo Park, California, 1996.

[6] J. Doyle, I. Kohane, W. Long, and P. Szolovits. The architecture of MAITA: A tool for monitoring, analysis, and interpretation. Technical report, Massachusetts Institute of Technology, Laboratory for Computer Science, 1999.

[7] J. Doyle, Y. Shoham, and M. P. Wellman. A logic of relative desire (preliminary report). In Z. W. Ras and M. Zemankova, editors, *Methodologies for Intelligent Systems, 6*, volume 542 of *Lecture Notes in Artificial Intelligence*, pages 16–31, Berlin, Oct. 1991. Springer-Verlag.

[8] J. Doyle and M. P. Wellman. Representing preferences as *ceteris paribus* comparatives. In S. Hanks, S. Russell, and M. P. Wellman, editors, *Proceedings of the AAAI Spring Symposium on Decision-Theoretic Planning*, 1994.

[9] J. Fackler, I. J. Haimowitz, and I. S. Kohane. Knowledge-based data display using trendx. In *AAAI Spring Symposium: Interpreting Clinical Data*, Palo Alto, 1994. AAAI Press.

[10] P. Haddawy and S. Hanks. Representations for decision-theoretic planning: Utility functions for deadline goals. In B. Nebel, C. Rich, and W. Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 71–82, San Mateo, CA, 1992. Morgan Kaufmann.

[11] I. J. Haimowitz and I. S. Kohane. Automated trend detection with alternate temporal hypotheses. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 146–151, Chambery, France, 1993.

[12] I. J. Haimowitz and I. S. Kohane. An epistemology for clinically significant trends. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 176–181, Washington, DC, 1993.

[13] W. Hamscher. Modeling digital circuits for troubleshooting. *Artificial Intelligence*, 51:223–227, 1991.

[14] W. Hamscher, L. Console, and J. de Kleer, editors. *Readings in Model Based Reasoning.* Morgan Kaufmann, 1992.

[15] I. Kohane and I. Haimowitz. Hypothesis-driven data abstraction. In *Symposium on Computer Applications in Medical Care*, Washington, DC, 1993.

[16] I. S. Kohane. Temporal reasoning in medical expert systems. In R. Salamon, B. Blum, and M. Jørgensen, editors, *MEDINFO 86: Proceedings of the Fifth Conference on Medical Informatics*, pages 170–174, Washington, Oct. 1986. North-Holland.

[17] I. S. Kohane. Temporal reasoning in medical expert systems. TR 389, Massachusetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA, 02139, Apr. 1987.

[18] I. S. Kohane and I. J. Haimowitz. Encoding patterns of growth to automate detection and diagnosis of abnormal growth patterns. *Pediatric Research*, 33:119A, 1993.

[19] Charles Rich and Howard E. Shrobe. *Initial Report on A LISP Programmer's Apprentice.* MIT Artificial Intelligence Laboratory Technical Report 354, December 1976.

[20] J. S. Rosenschein and M. R. Genesereth. Deals among rational agents. In B. A. Huberman, editor, *The Ecology of Computation*, pages 117–132. Elsevier, Amsterdam, 1988.

[21] Y. Shahar. *A Knowledge-Based Method for Temporal Abstraction of Clinical Data.* PhD thesis, Stanford University, Stanford, CA, 1994. Available as Computer Science Department report CS-TR-94-1529.

[22] P. Szolovits, J. Doyle, W. J. Long, I. Kohane, and S. G. Pauker. Guardian Angel: Patient-centered health information systems. Technical Report MIT/LCS/TR-604, Massachusetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA, 02139, May 1994.

[23] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and S. Stornetta. Spawn: a distributed computational economy. *IEEE Transactions on Software Engineering*, 1992.

[24] M. P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[25] M. P. Wellman and J. Doyle. Preferential semantics for goals. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 698–703, 1991.

[26] M. P. Wellman and J. Doyle. Modular utility representation for decision-theoretic planning. In *Proceedings of the First International Conference on AI Planning Systems*, 1992.

[27] B. Williams and J. deKleer. Diagnosis with Behavior Modes. In *Proceedings of the 11th Joint Conference on Artificial Intelligence, IJCAI-89, pages 1324–1330*, Detroit MI, 1989 .

[28] B. Williams and P. P. Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of AAAI-96*, pages 971–978, 1996.

[29] G. Zlotkin and J. S. Rosenschein. Mechanisms for automated negotiation in state oriented domains. *Journal of Artificial Intelligence Research*, 5:163–238, 1996. http://www.cs.washington.edu/research/jair/abstracts/zlotkin96a.html.