Journal of Computer Science 2 (6): 496-504, 2006
ISSN 1549-3636

Autonomic and Dependable Computing: Moving Towards a Model-Driven Approach

[1]Yuan-Shun Dai, [1]Tom Marshall and [2]Xiaohong Guan
[1]Department of Computer and Information Science, Purdue University School of Science
Indiana University, Purdue University, Indianapolis, IN, USA
[2]Department of Automation, Tsinghua University, China

Abstract: The rapidly increasing complexity of computing systems is driving the movement towards autonomic systems that are capable of managing themselves without the need for human intervention. Without autonomic technologies, many conventional systems suffer reliability degradation and compromised security. Autonomic management techniques reverse this trend. This study describes the roles and functions of various autonomic components, and systematically reviews past and current approaches that have been/are being developed to address specific areas of the autonomic computing environment that focus on improving system dependability including both reliability and security concerns. Analyzing past research can lead to the design of a more advanced, dependable autonomic computing system. A novel and promising prototypical system that is a work-in-progress will be presented finally.

Key words: Autonomic computing, self-x, reliability, security, model-driven

INTRODUCTION

Dependability can be defined as the quality of a delivered service such that reliance can justifiably be placed on the service[1]. Five attributes of dependability are: (1) Reliability, (2) Availability, (3) Safety, (4) Security, and (5) Robustness. Our goal is to incorporate each of these attributes into holistic autonomic management architecture.

Ensuring dependable system performance is important because of the high costs typically associated with failure and/or fault recovery. Patterson, estimates that, "companies spend 33 – 50% of their total cost of ownership recovering from or preparing against failures." Improving system dependability can have a dramatic impact on a company's bottom line.

The economic impact of dependable systems is but one reason why improving system dependability through autonomic means is important. However, there are situations in which the general well-being of a geographical region's populace is at stake. For example, a series of failures that occurred at the Three Mile Island nuclear power plant in 1979 nearly led to catastrophic consequences. Making systems safer, without relying on human intervention, is a kind of insurance policy on the health and welfare of a society's citizens.

Furthermore, in the post 9/11 era it is extremely important to develop autonomous methods of self-protection to ensure that a nation's computerized defense systems remain reliable in the face of malicious attacks. If a terror organization is successful in bringing down a nation's power grid, or render communication channels useless, the security of that nation is at dire risk. The integration of a security component into the autonomic system design enhances dependability.

Although the study of autonomic computing is in its relative infancy, there have been some projects and articles on the subject. Autonomic computing is a concept envisioning self-managing systems. The term "autonomic" is intentionally chosen because the idea is to mimic the autonomic nervous systems found in biology. The systems self-configure, self-optimize, self-heal, and self-protect. Because of the increasing complexity of computing systems, human systems management is rapidly becoming obsolete. Humans are simply not able to optimally configure these large, complex, heterogeneous, and dynamically evolving systems in an effective manner. Therefore, there is a strong need to move away from human managed systems to autonomic managed systems

This study identifies some important technologies and theories that have been proposed to be useful in an autonomic computing environment focusing on those areas that will enhance the dependability of such a system. The three areas this study will address are: (1) Self-configuration – automatic adaptability to changes in a system's physical topology, software environment, or communication channels; (2) Self-healing – automatic recovery from faults that have occurred or are about to occur; and (3) Self-protection – automatic protection from malicious attacks or insider abuses. Finally, a preliminary design for a highly dependable autonomic system is presented. Monitoring plays a central role in the implementation of these concepts in an autonomic computing system[2].

Corresponding Author: Yuan-Shun Dai, Indiana University, Purdue University, Indianapolis, USA.

Another primary contribution of this study is the introduction of the Model-Driven approach for autonomic management. This scheme utilizes a hierarchical approach to provide for customization, flexibility and adaptability. The Model-Driven approach is a new idea in the development of a comprehensive, dependable autonomic management system. It uses probability-based monitoring, has a machine learning component that enables it to perform better the longer it runs, and is able to focus its resources on those components that are most likely to experience problems. A thorough description of this novel design is presented in Section 5 of the study.

SELF-CONFIGURATION

Self-configuration[3] refers to the ability of the system to automatically adapt to changes in its physical topology as well as its software environment. Configurations may also need to be altered based on a change in the quality of network connections. Self-configuration improves system reliability by reducing human configuration errors and minimizing downtime to increase system resource availability[4,5].

Adaptive domains: Motezenko[6] presents the idea of a generic adaptation environment for building distributed object systems with multiple self-managing attributes. Each domain is separately managed based on policies that are either input by a human or loaded from a parent domain during runtime. The system is hierarchical. A Host Manager oversees the operation of the Domain Manager which manages parent and child nodes of a specific domain. The Domain Manager then is tied to a specific Managed Object which may be a sensor that is responsible for one or more domains.

Adaptation strategies can be reactive, event-driven; proactive, preventive; or retroactive. Adaptation decisions are implemented by using three types of actuators: (1) Configuration Manager – implements dynamic reconfiguration based on the system components and connections graph which can be built at runtime; (2) Adaptation Commands – used for communication between parent and child domains; and (3) Mobile Adaptation Agents – used for executing changes within managed objects

The adaptive domain strategy improves system reliability by eliminating or dramatically reducing the need for human systems managers to configure complex, dynamic systems. Human configuration of these systems, because of their complexity, can often be error prone and suboptimal.

Adaptive Techniques Using AOP: Chan and Chieu[7] present the idea of using Aspect-Oriented Programming (AOP) to implement a monitor system that is outside the OS kernel or applications.

The application being monitored is connected to sensors and effectors of the autonomic manager via an aspect crosscut layer. The effectors self-configure if the sensors suggest corrective action needs to occur. The main advantage of this system is that it provides for clear separation of concerns and the ability to crosscut the concerns without modifying the source code which makes it suitable for providing monitor services to legacy applications. AOP functions are treated as a concern, developed separately, and integrated selectively in applications either during development or at runtime.

Utilizing the Aspect-Oriented approach enhances system reliability because each monitoring system can be specifically tuned to the application(s) being monitored without modifying the application's source code and does not impose additional overhead on OS resources because it is not kernel resident.

Topology based adaptation: Paulson[5] considers self-configuration to be concerned with the physical design and deployment of the system. This approach considers both static and dynamic aspects. Static is concerned with the physical topology of the system and dynamic is concerned with adapting to changes from initial state. A current project that utilizes this approach is called LAMDA (Lights-out, Automated Management of Distributed Applications). For self-configuration, LAMDA uses Hierarchical Queuing Petri Nets to model the environment.

Again, because the human element is removed from configuring a complex, dynamic system, reliability is improved by reducing the possibility of configuration errors and suboptimal tuning.

Utility optimized adaptation: Boutillier[8] and Buyya[9] use the economic theory of utility to determine how resources should be allocated among competing resource users. Resources are limited, but wants are not. Economic utility theory analyzes what the consumer is willing to give up in order satisfy its wants and/or needs. Individual utility curves are constructed to form an aggregate utility curve which can be fitted against the available resources to find various combinations of resources that can maximize organizational utility.

Boutillier[8] uses an automated resource manager, called a provisioner, which works in concert with a workload manager to allocate resources to clients. The provisioner's task is to allocate resources to the workload managers in a way that maximizes total organizational utility and solves the problem of resource allocation in autonomic systems.

To determine maximum organizational utility, samples of individual machine utility curves are taken at certain critical allocation levels. Based on these samples, the maximum utility function is created using

regression analysis. The problem is framed as a form of cooperative negotiation between the provisioner and the workload managers[8].

A problem arises in how one defines utility. If reliability is chosen as the metric of utility, total system reliability is enhanced because each machine's utility is maximized when it is operating in its most reliable state.

Decentralized multi-agent management: An agent-based model is a dynamic description of a system, its entities and properties, and has the capability of adapting to its environment. DeWolf and Hoelvet[10] postulate that multi-agent systems allow a natural modeling of the system, and explicitly consider autonomous behavior and distributed interaction. Dynamical systems theory allows analysis of the dynamics of these models. Decentralized control analysis can use insights gathered from the analysis to create decentralized control mechanisms to control the dynamics of autonomous systems. Multi-agent systems, dynamical systems theory, and decentralized control in combination can make significant contributions in achieving the development of highly reliable, autonomic systems.

Dynamical systems theory offers a conceptual framework with which dynamical systems can be characterized. The choice of what data to monitor is a critical aspect of dynamical analysis.

Decentralized control systems consist of controllers that are designed and operated with limited knowledge of the complete system. It is actually a self-organizing dynamic property of the system that allows control to be applied to complex systems.

A hierarchical control mechanism is a good compromise to avoid the performance degradation of a completely decentralized system as well as the high cost and complexity of a centralized solution. It still has a degree of decentralization but also keeps a degree of centralization through a higher-level control layer that manages lower level units. The advantage is that control can be enforced through multiple hierarchies, each with their own effect thus improving overall system reliability.

SELF-HEALING

Self-healing is concerned with the ability of the system to automatically recover from faults. Accurate detection is critical to designing an effective self-healing system. Fault detection is accomplished by some form of monitoring. The questions to be answered are what parameters to monitor, how to determine accurately if a fault has occurred or is likely to occur and what corrective measures can be taken to repair the system unobtrusively. The goal is to repair only that component that has failed without bringing down the entire system so that resource availability is maintained even if the QoS is somewhat degraded. Fault tolerance is crucial to system operation[11].

Protective techniques using AOP: AOP can be used for self-healing in addition to self-configuration. The basic approach is same in that the source code need not be accessed to implement the system. For self-healing applications, the parameters to be monitored are those that are associated with fault-detection rather than configuration. Corrective action takes place when a monitored parameter indicates that a fault is occurring or about to occur.

Sensors monitor the state of the system and pass the information to an analyzer which can then determine if the system is in a steady state. If not, the analysis is passed to a planner which contains a set of rules and action plans. Once an action has been selected it is passed to the executor which implements the change through an effector that is interfaced to the monitored application via an aspect crosscut layer which is developed independently of the application and the autonomic manager[7].

Self-healing using LAMDA: In the LAMDA project, the critical prerequisite for self-healing is the same as that for self-configuration[12]. For self-healing the design uses multi-agent architectures coupled with distributed correlation algorithms that correlate across network, computer, and software infrastructure layers. The reason given for this approach is that it provides the ability to decentralize decision making as related to root cause isolation and provides a means for machine learning to identify causal patterns of faults that occur in complex systems. The decentralization improves system reliability because determination of the cause of a fault is distributed, and not dependent on one central control mechanism which may itself be affected by a fault. Decentralization can also provide for redundancy, which has historically been an effective reliability strategy.

Recovery oriented computing (ROC): Patterson *et al.*[4] state that ROC focuses on MTTR rather than MTTF in order to provide higher system availability. ROC suggests the following six techniques: 1) Redundancy; 2) Failure containment; 3) Fault insertion testing; 4) Error diagnosis; 5) Non-overwriting storage systems; and 6) Orthogonal mechanisms.

Traditional fault diagnosis methods use dependency models that have a static nature. The problem with static dependency models is that they are ill suited for dynamically evolving systems.

Dynamic dependency models are used in the ROC approach. The dynamic analysis methodology is automated and implemented by tracing real client requests through a system. The success or failure is

recorded together with the components that served the request. Standard data clustering and statistical techniques correlate the request failures to the components most likely responsible for the failure. The use of data clustering to analyze successes and failures provides for the discovery of the combination of components that are most highly correlated with the failed requests. The prototype system using this approach is called "Pinpoint."

Once failures have occurred, how is recovery initiated? Rather than perform a hard reboot on the system, a recursively restartable system that gracefully tolerates successive restarts at multiple levels is used. Also, fine grain partitioning of the system enables bounded, partial restarts that recover a failed system faster than a full reboot. It also enables strong fault containment and diagnosis providing for enhanced system reliability.

Machine learning to promote self-healing: Based on the RAS Cluster system, Sahoo *et al*.[3] propose a self-healing approach for the system to "learn" the series of events that lead to failures thereby providing prediction and system management process control.

To diagnose the path of failure propagation in a large cluster, a machine learning algorithmic approach is needed to make the system prediction more intelligent and reduce failures thus improving system reliability. One of the first steps for developing machine learning algorithms is to carefully select variables that can be analyzed using event log collection.

Time-series algorithms are used to predict system parameters such as percent of system utilization, idle time, and network I/O. Rule-based classification algorithms are synonymous with statistical control mechanisms. Events of a critical nature must be identified and prediction mechanisms must be constructed in order to anticipate the occurrence of the critical event(s). The goal is to create a set of rules that can accurately predict with high probability that a critical event will occur. Bayesian network techniques are utilized for root cause analysis. This involves "learning" the cause of critical events through probabilistic dependency models based on event log data.

A hybrid prediction and proactive control model has been theoretically designed utilizing the above techniques[3]. The system "learns" over time the patterns that lead to critical events, and should be able to send an alarm when patterns occur that it believes with high probability will lead to a critical event.

Autonomic middleware services: The use of middleware for achieving an autonomic system shows much promise for autonomic computing. In AUTONOMIA[13], an autonomic computing project being conducted by the University of Arizona, autonomic middleware services are described as an operating system that provides applications with all the services and tools required to achieve the desired autonomic requirements. It contains five modules, the most important of which is the Policy Engine and Autonomic Services module. This is where the autonomic "rules" and action plans are stored.

In AUTONOMIA, self-healing is realized by analyzing data from an event server. If any data is "suspect", the policy engine is notified where the data is compared against a pre-defined set of rules. If a rule is violated an action plan associated with that rule is activated initiating healing activity.

Nonintrusive healing using backdoors: Sultan *et al*.[14] propose a method whereby the system utilizes an architecture that supports monitoring and repair actions on a remote operating system or application memory image *without* using the processor(s) of the target machine. It uses technology that provides support for remote DMA read and write operations.

External monitoring overcomes the weaknesses of internal monitoring. For example, it is not dependent on the resources or adversely affected by component failures in the monitored machine, it does not rely on the integrity of the machine, and it can detect intrusion from an already compromised machine.

To support remote healing a computer system must be equipped with a backdoor, a specialized network interface that allows external accesses to its resources without involving its processors. Backdoors enable intervention on a system even when it is "dead." There must be generic support by the OS to enable remote healing via a backdoor and channels to allow remote access to system memory. Backdoors can be implemented using remote memory communication (RMC). RMC significantly reduces communication overhead associated with TCP/IP networking by bypassing the OS in the send/receive path while providing a protected channel for communication.

Nonintrusive Remote Monitoring alleviates the problems of imperfect system knowledge, network unreliability, resource contention, and does not incur the overhead at the monitored nodes. Furthermore, it does not suffer from a lack of direct access to the state of the monitored node.Once a failure has been detected, healing is performed using RMC. To support remote healing the OS must provide remote access hooks to provide an interface for enforcing actions on the OS or the applications running on it. The hooks must be registered with the RMC for remote access by another system that runs recovery or repair code.

SELF-PROTECTION

The self-protection component of an autonomic computing system is concerned with protecting the

system from malicious attacks. Most of these attacks are assumed to be generated from external sources, but the system must be prepared to address attacks that are initiated from within the system as well. Examples of types of attacks that should be monitored fall into three basic categories: (1) Denial of Service; (2) Viruses and Worms; and (3) Application level attacks or failures. The introduction of a security component in the autonomic system design moves the focus from reliability to dependability.

Dynamic defense theory: Kewley and Bouchard[15] implemented their ideas in experiments conducted by DARPA. The goal was to determine if dynamically modifying one's defensive posture would hinder the adversary's intelligence gathering process thus decreasing system vulnerability. Three mechanisms were employed: prevent, detect, defend.

Two basic categories of attacks need to be defended: (1) outsider attacks; and (2) insider abuses. Statistical methods, machine learning algorithms, and rule-based approaches are examples of methods that have been used in an attempt to solve the problem of cyber defense.

An effective defense mechanism relies on a thorough understanding of the system to be protected. An intrusion detection strategy that is specifically designed for that system must then be implemented. Points of vulnerability must be identified and protected. Once the detectors are properly implemented, an autonomous response system is possible. The three pronged scheme of prevention, detection, and defense provides multiple layers to help ensure system dependability. By dynamically changing the prevention mechanism, intrusion is made more difficult. Highly accurate detection mechanisms reduce the risk of a compromised system in the event that the prevention layer is penetrated. Finally, if intrusion is detected the defense mechanism can automatically be initiated to minimize the potential damage of the attack.

Feedback control as a defense mechanism: Kreidl and Frazier[16] propose a feedback mechanism that considers the tradeoffs of compromised information systems resulting from "false negatives" and the maintenance costs of unnecessary ongoing defensive countermeasures that result from "false positives."

They propose to combine online implementation using sensors to detect intrusions and an offline component that stores models and numerical optimization utilities. The approach makes heavy use of probabilistic models for decision-making.

The offline component performs the mathematical, probabilistic computations for decision-making that is then stored in an online module to determine if the system state is normal. In order to accurately detect whether the system state is normal, the choice of a tolerance range is critical. Choose too small a tolerance range then the rate of false positives increases. Alternatively, a tolerance range that is too large may result in attacks going undetected. Feedback can allow for the dynamic adjustment of tolerance ranges.

The mechanism they propose is called an Autonomic Defense System (ADS). The ADS consists of the information system to be protected, sensors to detect attacks, actuators to implement appropriate responses, and a controller to coordinate the sensors with the actuators. A Feedback Controller is also part of the ADS system. Feedback control continuously receives sensor information, estimates its implication based on historical data, and generates an appropriate response. Feedback is important because it can cause decision making to change over time based on historical data from both the sensors and actuators so that better, more appropriate decisions can be made thus enhancing system dependability.

Software agents for self-protection: Qu *et al.*[17] propose a scheme whereby software agents monitor several attributes of a system online to characterize the state of the network as normal, uncertain, or abnormal. Measurement attributes that are deemed appropriate at various network levels are used to quantify the behavior of a network or its components. A recovery mechanism is executed once it is determined that the system or a component is functioning abnormally.

The approach is proactive, and uses the self-protection engine of AUTONOMIA[15]. Self-protection mechanisms are implemented to protect against attacks based on predefined metrics used to monitor the states of services and resources. If an attack is "identified", proactive measures are initiated by the self-protection mechanisms.

By constantly examining specific activity at various levels of the OSI reference model, overall dependability can be improved by defending against attacks at the point of intrusion. The system is monitored to determine whether it is operating in a normal, uncertain, or abnormal state. When it is determined to be in abnormal state, appropriate recovery procedures are initiated to affect self-protection.

MODEL-DRIVEN AUTONOMIC APPROACH

Characteristics: Although some approaches have been proposed to address the requirements of specific autonomic components, our goal is to develop a comprehensive system that deals with *all* of the areas of autonomic computing that relate to dependability improvement: self-configuring, self-healing, and self-protecting. The prototype we propose is based on, what we term, the model-driven approach.

In developing the framework for our system, not only did we attempt to weave together the best ideas from previous approaches, but also integrate novel ideas into the design.

One of the most significant differences in the model-driven approach is that network components (e.g. nodes, channels and traffic) are not monitored evenly/randomly. Rather they are monitored based upon the predicted reliability or security provided by the models. For example, the components that are predicted to have lower dependability at any given point in time are monitored more intensively than highly reliable components. Furthermore, the dependability is not static. As time passes a component that may initially be highly reliable could become more prone to failure. Therefore, the monitoring frequency of that component should be adapted accordingly over time. Failure correlations should also be integrated into the monitoring decision[19]. For example, Component B may be highly reliable, but if Component A fails, Component B may then have a higher probability of failure, perhaps due to the heavier load moved from A, and thus requires more intensive monitoring.

To initially test this approach, a software simulator[20] was developed that utilized fault insertion testing to determine if uneven/probability-based monitoring and healing was superior to traditional "even/random" monitoring approaches. The results were clear: "uneven/probability-based" monitoring consistently showed that more nodes were operational at any given time, and the healing process was performed more quickly. The development of the automated fault/intrusion detection/healing mechanism is the first step in improving system reliability autonomously. The simulator is publicized by[20].

Following validation of the significant improvement provided by the "uneven/probability-based" monitoring approach, we determined what characteristics the system design should possess. First, it is important that the model be *general* so that it can be applied to as many network topologies and structures as possible. Second, the system should provide interfaces to identify different levels of models so that new reliability and security services can be built on top of, and integrated with, existing models. In other words the system components should be *pluggable*. Third, the system model needs to possess the ability to *adapt* to component changes "on the fly" rather than require periodic reevaluation that is computationally expensive. Fourth, the structure should be *hierarchical*. In fact, our design develops two types of hierarchical architectures: a monitoring hierarchy and a modeling hierarchy as depicted in Fig. 1 and 2 respectively.

Monitoring hierarchy: In the monitoring hierarchy, the higher levels monitor the levels beneath them, as depicted by Fig. 1. Additionally, monitors in each level monitor other monitors on the same level except at the machine level.

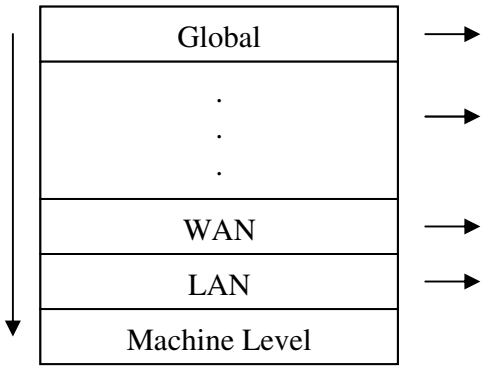

Fig. 1: Monitoring hierarchy

Hierarchical control allows monitoring to be utilized in different ways at different levels. Machine level monitoring can be accomplished either through an OS resident approach by patching the OS to provide the additional functionalities, or perhaps the AOP approach offers more potential because it can be implemented without altering the OS or applications' source code. At the machine level it is important that the monitoring system not consume much computational or storage resources. It should be a background process that transparently runs without degrading the performance of the applications that are running on the machine.

Moving up the monitoring hierarchy, the next layer is the LAN level. At this level the individual machines would be monitored for aliveness. In the event the machine is detected to be in a state whereby it is unable to perform the autonomic functions that are resident in it, this layer could revive the system, and heal it through Backdoor mechanisms. Self-protection would also be present at this layer. In fact, the self-protection that is present at the machine level could be viewed as the last line of defense. The outermost layer would be the first line of defense. It would probably be appropriate to utilize redundancy as we move farther away from the machine level, i.e. employ multiple monitors that not only monitor the individual machines, but also watch other machines or monitors.

At the LAN level, monitoring and control mechanisms can be configured to meet the specific requirements of the organization. For example, tolerance ranges would be configured more tightly for safety-critical organizations. Alarms and corrective measures may be designed to be more aggressive and proactive in these organizations. Furthermore, at this layer and other layers that are farther away from the machine, the resources consumed by the monitoring processes is not a constraint as it is at the machine level. This is because the hosts/servers/agents in these higher layers are dedicated solely to the management process (coordination, diagnosis, modeling, analysis, and

healing) The complexity of the monitors at the organization level may be greater than the more generic monitoring processes of the higher layers which might encompass multiple organizations, thus reducing the potential for customization at those layers.

Modeling hierarchy: The modeling hierarchy depicted in Fig. 2 can be briefly described as follows: There are three layers in the hierarchy. Communication between the layers is provided by interfaces that allow pertinent information to be passed to the appropriate model modules resident in each layer.

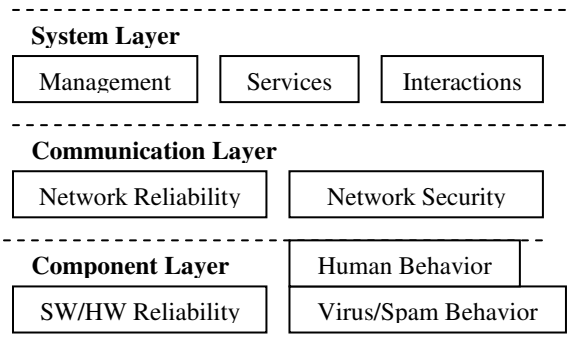

Fig. 2: Modeling hierarchy

Each layer is comprised of various model modules. For example, in the component layer (which is the lowest level of the modeling hierarchy) the SW/HW Reliability module contains models that enable the autonomic system to determine the state of the software and hardware components. Stochastic control charts may be one method that is used to determine whether these components are in a normal or abnormal state.

The Human Behavior models in this layer are responsible for ensuring system dependability in spite of human errors which can never be eliminated. Three types of human operator errors[18] that our Human Behavior models must address are: (1) Slips/Lapses - operators not doing what they intended to do, (2) Unintentional Mistakes – operators doing what they intended to do, but their action was the *wrong* action to perform, and (3) Intentional Mistakes – malicious actions carried out by hackers, intruders, or eavesdroppers.

The Virus/Spam Behavior models are concerned with system security and protection and are thus designed to improve system dependability.

The model modules resident in the communication layer are self-descriptive. Network Reliability models are responsible for ensuring reliable communication between nodes, files and services that may be distributed throughout the network. Network Security at the Communication Layer includes models for authentication via keys and certificates, for example. Also cryptography is utilized to ensure secure data transfer and communications.

At the highest level of the Modeling Hierarchy is the System Layer. The most important model module in this layer is the Management module which stores the models for three distinct management areas: (1) Resource management, (2) Reliability and Security management, and (3) Service management.

It should be noted that the model modules shown in Fig. 2 is not an all-inclusive list. Other modules can be added based on organization-specific requirements. This provides for additional customization offered by the model-driven approach. For a more comprehensive discussion of the specifics of various models, please refer to[21].

Overall architecture of model-Driven autonomic management: Once the monitoring hierarchy and modeling hierarchy have been established, the next step is to determine how to integrate these ideas into an architecture that produces the prototype design for a dependable, secure, and comprehensive autonomic system that meets all four of the aforementioned design goal characteristics. The proposed architecture is depicted in Fig. 3.

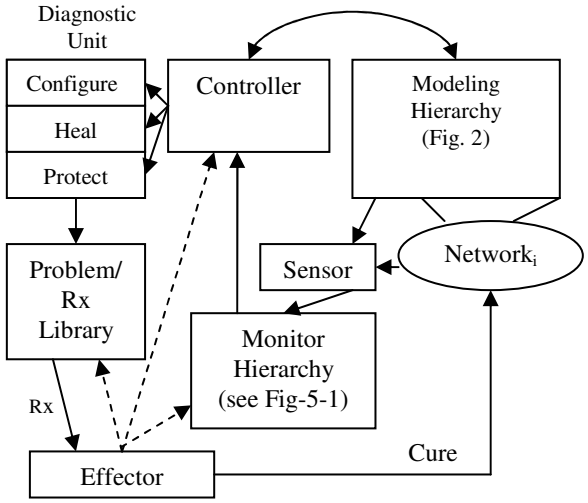

Fig. 3: Autonomic system architecture

The modeling hierarchy is present in each network that is resident in the entire system (only one network is shown for illustration purposes). This allows for a new LAN, for example, to be "plugged in" to the existing architecture without the need for manual reconfiguration.

The sensor collects data deemed relevant for evaluating the state of the network or machines for which it is responsible. The sensor can initially be configured with default values, or parameters may be loaded from other domains that are already present in the system. However, the sensor is dynamic. This is accomplished by integrating a machine-learning component into the sensing and monitoring mechanism.

Data retrieved and stored by the sensor is passed to the monitor where a decision regarding the system state is made. If the system is determined to be in an abnormal or uncertain state, the pertinent data is passed to a high-level controller that makes a determination on the nature of the problem at hand. The controller then chooses the configuration, healing, or protection component of the diagnostic unit, which is linked to a problem/prescription library. The library contains a list of known problems (configuration issues, fault/failure issues, and security/protection issues). Each problem is mapped to a list of proposed corrective "prescriptions". The prescription that is determined to be the best to solve the current problem is chosen and passed to an effector, which implements the cure by sending it to the component, that needs to be healed.

A feedback loop from the effector to the problem library, the controller, and the monitor modules provides a machine learning mechanism for the system architecture, and thus provides for adaptability. This feedback loops allow the system to perform better with the passage of time.

First, the feedback to the Problem/Prescription library tells this module whether the prescription successfully solved the problem. If not, the library notes that the fix was unsuccessful then chooses another prescription to try.

Second, the feedback to the Controller is used to notify the Controller that it erroneously decided corrective action was necessary when in fact no problem existed. For example, the Controller may have incorrectly identified a user attempting to access the network as an "intruder" and thus triggered a protection mechanism. The feedback loop will tell the Controller that the user was "safe". Not only must this information be passed back to the Controller, but cooperative communication between the Controller and Modeling Hierarchy must also be present. This is required so that the models can be updated as necessary, and the Controller can make better decisions based upon the updated models.

Lastly, the feedback to the Monitor provides real-time updates to the parameters being monitored to determine the state of the system and adjust tolerance ranges if necessary. For example, if the tolerance range is too narrow for a specific parameter, the system will experience a higher frequency of false alarms than would typically be expected. Therefore, the effector provides this information back to the Monitor where adjustments can be made autonomously.

It should be noted that the components of the system architecture are distributed and varied within and across network layers. They are dynamic and customizable.

Although the system is a work-in-progress that needs validation through testing, we believe that the model-driven approach will improve system dependability for many reasons, some of which are:

* Monitoring resources are well allocated on components according to their predicted reliability and security, and real-time behaviors,
* Model-Driven provides for not only a reactive, event-driven healing mechanism but more importantly a proactive, predictive technique to prevent failures before they occur,
* Network resources are not wasted on unnecessarily monitoring highly reliable components intensively,
* MTTR is reduced,
* The hierarchical structure provides for task-specific and organization-specific tuning at the various monitoring levels and within the modeling hierarchy,
* The hierarchical structure provides multi-level intrusion detection, protection, and response,
* The feedback loops provide a mechanism for continuous real-time updates to critical components of the Autonomic System Architecture.

DISCUSSION AND COCLUSION

The goal of developing a completely autonomic computing system has been the motivation for much research. This study has presented research that appears promising in designing an autonomic strategy that will improve overall system dependability. While the study looks at ideas in a categorical manner, the challenge in realizing a truly dependable autonomic system lies in weaving together the best ideas from each area into a cohesive and complete system and building new ideas into the system.

The Model-Driven approach we present utilizes a hierarchical structure; one in which the system is viewed as a series of layers. This design provides for both customization and adaptability.

Monitoring is implemented either by statically establishing parameters to be monitored or dynamically through communication with the monitored nodes at run-time. Sensors and effectors are used to monitor the system and adjust the system as necessary.

The preliminary design of the Model-Driven approach presented utilizes some aspects of previously proposed approaches but adds significant novel techniques that we believe will produce a better and more intelligent autonomic system thus providing dependability improvements that are superior to the current technologies. To validate our design and assumptions, the prototype design must be implemented and tested on a distributed system at our research facilities.

REFERENCES

1. Hennessy, J.L. and D.A. Patterson, 2003. Computer Architecture: A Quantitative Approach. 3rd Edn., Morgan Kaufmann, San Francisco.
2. Kephart, J. and D. Chess, 2003. The vision of autonomic computing. Computer, pp: 41-50.
3. Sahoo, R.K., I. Rish, A.J. Oliner *et al.*, 2003. Critical event prediction for proactive management in large-scale computer clusters. 9th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, pp: 426-435.
4. Patterson, D., A. Brown, P. Broadwell et al., 2002. Recovery Oriented Computing (ROC): Motivation, definition, techniques, and case studies. Technical Report CSD-02-1175, Univ of California-Berkeley, pp: 1-25.
5. Paulson, L., 2002. Computer System, Heal Thyself. Computer, pp: 20-22.
6. Motuzenko, P., 2003. Adaptive domain model: Dealing with multiple attributes of self-managing distributed object systems. Proc. 1st Intl. Symp. on Information and Communication Technologies, pp: 549-554.
7. Chan, H. and T. Chieu, 2003. An approach to monitor application states for self-managing (autonomic) systems. 18th Annual ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages and Applications, pp: 312-313.
8. Boutillier, C., R. Das, J. Kephart, et al., 2003. Cooperative negotiation in autonomic systems using incremental utility elicitation. 19th Conf. on Uncertainty in Artificial Intelligence, pp: 89-97.
9. Buyya, R., H. Stockinger, J. Giddy and D. Abramson, 2001. Economic models for management of resources in peer-to-peer and grid computing. SPIE Intl. Symp. Convergence of Information Technologies and Communications, pp: 1-13.
10. Wolf, T.D. and T. Holvoet, 2003. Towards autonomic computing: Agent-based modeling, dynamical systems analysis, and decentralised control. 1st IEEE Intl. Conf. on Industrial Informatics, pp: 470-479.
11. Tohma, Y., 2004. Incorporating fault tolerance into an autonomic-computing environment. IEEE Distributed Systems Online, pp: 1-12.
12. Bellur, U., 2004. Topology based automation of distributed applications management. 4th Intl. Workshop on Software and Performance, pp: 171-173.
13. Dong, X., S. Hariri, L. Xue, H. Chen, M. Zhang, S. Pavuluri and S. Rao, 2003. AUTONOMIA: An autonomic computing environment. IEEE Intl. Conf. on Performance Computing and Communications, pp: 61-68.
14. Sultan, F., A. Bohra, I. Neamtiu and L. Iftode, 2003. Nonintrusive remote healing using backdoors. The First Workshop on Algorithms and Architectures for Self-Managing Systems, pp: 1-6.
15. Kewley, D. and J. Bouchard, 2001. DARPA Information assurance program dynamic defense experiment summary. IEEE Trans. on Systems, Man, and Cybernetics, Part A, pp: 331-336.
16. Kreidl, O. and T. Frazier, 2004. Feedback control applied to survivability: A host-based autonomic defense system. IEEE Trans. on Reliability, pp: 148-166.
17. Qu, G., S. Hariri, S. Jangiti, J. Rudraraju, S. Oh, S. Fayssal, G. Zhang and M. Parashar, 2004. Online monitoring and analysis for self-protection against network attacks. Proc. Intl. Conf. on Autonomic Computing, pp: 324-325.
18. Reason, J.T., 1990. Human Error. Cambridge University Press, New York.
19. Dai, Y.S., M. Xie and K.L. Poh, 2005. Markov renewal models for correlated software failures of multiple types. IEEE Trans. on Reliability, 54: 100-106.
20. Simulator for Model-Driven Monitoring vs. Even/Random Monitoring, http://www.cs.iupui.edu/DASC06/demo.htm
21. Xie, M., Y.S. Dai and K.L. Poh, 2004. Computing Systems Reliability: Models and Analysis. Kluwer Academic/Plenum, New York.