An Autonomic Computing System based on a Rule-based Policy Engine and Artificial Immune Systems

Rahmira Rufus, William Nick, Joseph Shelton and **Albert Esterline**
Department of Computer Science, North Carolina A&T State University, Greensboro, NC 27411
{rsrufus, wmnick, jashelt1}@aggies.ncat.edu, esterlin@ncat.edu

Abstract

Autonomic computing systems arose from the notion that complex computing systems should have properties like those of the autonomic nervous system, which coordinates bodily functions and allows attention to be directed to more pressing needs. An autonomic system allows the system administrator to specify high-level policies, which the system maintains without administrator assistance. Policy enforcement can be done with a rule based system such as Jess (a java expert system shell). An autonomic system must be able to monitor itself, and this is often a limiting factor. We are developing an automatic system that has a policy engine and uses Artificial Immune Systems (AISs) to sense its environment and to monitor its components and performance. AISs emulate the natural immune system to defend the body against external malicious entities. The proposed system monitors itself without human intervention and thus addresses the problem of systems complexity.

Introduction

With more systems and devices networked together, more system administrators are required to monitor and maintain these networks, and maintenance becomes costly and time consuming. Yet it may not be clear how a complex system may manage itself in the absence of a human. We here report on an autonomic system we are developing for a network-connected computer system that self-manages by using mechanisms similar to the autonomic nervous system of the body to achieve the so-called self-CHOP properties: self-configuring, self-healing, self-optimizing, and self-protecting. A rule-based policy enforcement engine is used (like the body's hypothalamus) to regulate the autonomic system, and the system has artificial immune systems (AIS) that mimic the functions of the sensory and motor subsystems.

The remaining sections of this paper are organized as follows. The next section provides background on autonomic computing, AISs, and rule-based systems. We then discuss the motivation for implementing an AIS. There follows a section that presents the architecture of our autonomic system. This is followed by the AIS sensoring agent section, where we describe how the AIS aids the autonomic system with its self-CHOP capabilities via anomaly detection. The context monitor is then described, followed by a section describing how the self-CHOP properties are realized by our system. The final section presents the conclusion and future work.

Background

Autonomic Computing

IBM, in a 2001 manifesto, compared complex computing systems to the human body, which has an autonomic nervous system that removes the tasks of consciously coordinating bodily functions (Huebscher and McCann 2008). Complex computing systems should have autonomic properties that independently take care of tasks of regular maintenance and optimization tasks, thus reducing the workload on the system administrator.

IBM also articulated the four self-CHOP properties. Self-configuration is defined as components and systems being configured as per high-level policies (Kephart and Chess 2003). When a component is introduced into a system, it is incorporated seamlessly, and the rest of the system adapts to the presence of the new component. With self-optimization, components and systems continually seek to improve their performance and efficiency. Self-healing is defined as a system automatically detecting, diagnosing, and repairing problems in software and hardware. Finally, self-protection is defined as a system being able to automatically defend against malicious attacks or cascading failures. Kephart et al. (Kephart and Walsh 2004) came up with three types of policies for autonomic computing: 1) action policies, 2) goal policies, and 3) utility function policies. Action policies specify what actions should be taken based on the current state of the system. A goal policy specifies either a desired state or a set of criteria for a desired state. Utility function policies are objective functions that provide a utility value for each possible state.

Artificial Immune Systems

The natural immune system is a defense mechanism that can learn about foreign entities that enter the body and respond to them by creating defensive antibodies. This concept has been artificially simulated for intrusion detection, resulting in an approach known as an artificial immune system (AIS) (Hofmeyr and Forrest 2000). Similar to the biological immune system, the goal of an AIS is to distinguish between self and non-self entities. The natural immune system that this system imitates depicts self as a cell that is innate or safe for the body while non-self is not. One can associate this mapping with detecting or sensing what is as opposed to what is not; this mapping is also known as a detector.

AISs have also been applied to the problems of fault diagnostics, fraud detection and detecting viruses (DasGupta 1993). There are a few methods for using an AIS. One is the negative selection algorithm (Forrest et al. 1994).This technique randomly generates a set of detectors that are trained to match any non-self entities for any system and not match any self entities. More specifically, the detectors are first applied on a set of self entities and the ones that detect the self entities are discarded. The idea is that, if a detector does not match self, it has a better chance of detecting non-self, which would be any anomaly in the system. The surviving detectors can then be applied on non-self to observe how much of the set can be detected.

Rule-Based Systems

The classical application of rule-based systems is in expert systems, which typically use a human experts knowledge for solving real world problems (Abraham 2005). This expert knowledge is often expressed in the terms of rules. These rules and data constitute a rule-based expert systems. Such systems have played a role in modern artificial intelligence and other applications such as fault monitoring. The Ponder rule-based policy language (Bradshaw, Uszok, and Montanari 2014) is the broadest and most widely used policy language.Its policies are rules that define system behavior choices that reflect on objectives set by system managers. Other rule-based policy enforcement systems use the event-condition-action rule paradigm. An example of this is Bell Labs' policy description language, in which a policy is a function that maps a series of events into a set of actions.

Carey et al. created a composite service execution engine for composing web services (Carey, Lewis, and Wade 2004). The policy engine for this execution engine uses the Jess rule engine (Friedman-Hill 2013). The rules are executed to refine goals into service policies. Before the policy engine can execute, additional information is required such as the name of the service and the finite state machine (FSM) models for each of the constituted services. Goals are refined by Jess rules triggered when the state used in high-level policy matches the state in the FSM of the composite service.

Requirements

We aim for a system capable of adapting to unforeseen occurrences in the operating environment. This includes adjusting to situations in a proactive manner as well as reactively supporting system recovery. Here pro-action amounts to perceiving danger then preempting harm or any compromised system state. Fault tolerance in the operating environment permits system execution to continue without interruption, while preempting failure from a system-wide perspective presupposes the occurrence of a compromised system component or operation.

Anomaly detection can be a key component to properly assessing whether events are uncharacteristic of a system's configuration (De Castro and Von Zuben 2000). Furthermore, accompanying this detection method with a response procedure that reduces the damage of the compromised component is crucial, but also crucial is simultaneously permitting uncompromised system components to continue. One goal of this research is to provide a sensoring component that promotes the self-CHOP requirements for an autonomic system.

One immunity-inspired algorithm of interest focuses on danger theory, which extends the role of the innate immune system for discriminating between 'self' and 'non-self' but employs the acquired immune system to react to danger (Brownlee 2011). Sensing danger lets a system protect, recover or heal, optimize and determine whether a reconfiguration assessment is necessary. Unlike Fail2ban, our system will learn new malicious and unwanted traffic to protect itself from malicious and unwanted traffic in the future.

System Architecture

Figure 1 shows our proposed system, which realizes the self-CHOP properties. The external world feeds information into the system through a series of sensors. AISs are used as detectors that identify anomalies from the outside world. The input from the external world also feeds into a rule-based policy engine. The rule-based policy engine connects to a rule base of policies. This engine accepts as input sensor data from the AISs as well as the context monitor, which is the component collecting information about the state of the system's context by monitoring network connections and system resources. In addition to providing status information to the context monitor, the system resources also provide status information to the AISs to help inform the detectors. The system is maintained in a proper state by the rule-based policy engine.

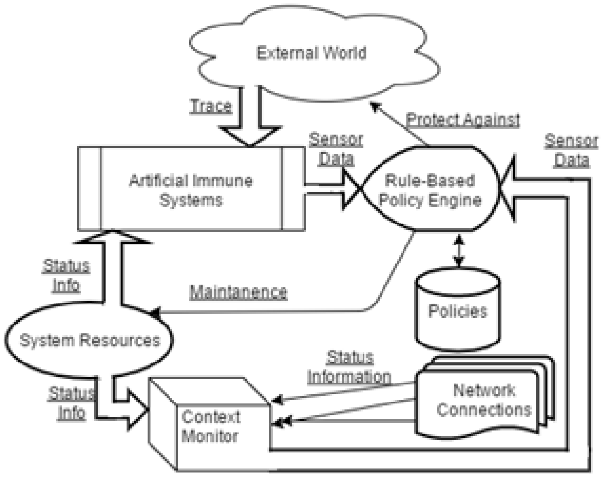

Figure 1: AIS & Autonomic System with Rule Based System

AIS Sensor Agents

We here discuss how the AISs contribute to the self-CHOP properties. Note that sometimes we refer to a single AIS while other times we partition the functionality among several AISs. Although we refer to sensing, note that the AISs detect both external and internal state. The AIS sensoring apparatus will consist of detector populations that monitor system activity as does a network sniffer or intercepting proxy. The AIS component will deploy detectors that compare system activity against normal system behavior. The detection methods behave as intrusion detection systems (IDSs) that detect system irregularities not stated in the innate system configuration. The detection method can also adapt to changes in the system.

The detector set will become a new feature vector for detecting danger in the system. The new vector will allow the system to recognize system behavior that has been previously labeled as dangerous. The new danger vector is then propagated throughout the system to scan for stress levels that are approaching this danger state.

Each round, the AIS will acquire more knowledge about the operating environment via the detector populations. The cloning process allows one generation to be more adept than the previous. The acquired immunity that aids the AIS in determining the health of the system is expressed via the number of danger vectors that have been detected and mitigated properly, with more emphasis on the mitigation process.

Following the detection of a stress level threshold being met, the policy engine will determine whether the system is in danger based upon the signal that the AIS found to indicate danger. If danger is confirmed, then the policy engine will determine what the system needs to repair or heal itself. For each consecutive detection round, the AIS will monitor the healing process by executing a subset of detectors for repair monitoring.

The system will adapt to the configuration expected of systems and components as per high-level policies during self-configuration (Kephart and Chess 2003). Conversely, within the configuration assessment module, the AIS will initially transmit a response to the policy engine that the need for repair or healing is not foreseeable. During the healing or repair process, the repair subset monitors the healing of the system via the change from affinity to a danger vector to the self set. Excessive danger is the signal to the policy engine to determine whether the system should limit the usage of the system component because the component might cause other components to be compromised. To increase the fault tolerant capabilities for the autonomic system, the AIS will monitor the affinity to danger or self for all system components involved or related to this configuration assessment.

Context Monitor

The environment of a computer system can undergo changes for any number of reasons. To deal with this, one exploits context awareness, a computer sensing and reacting to changes in its environment (Abowd et al. 1999). The environment that is monitored by some context-aware device is generally external; however, there is no reason that the environment cannot be the internal workings of a system. There are several consistent workings of any system that are consistent, such as CPU usage, amount of memory available, and network traffic to name a few.

Rule-based Policy Engine

For our rule-based policy engine, we used Jess (Java Expert System Shell) (Friedman-Hill 2013) . The policies use if-then rules as is standard with Jess. The variables in the rules are values provided by the AIS and the context monitor. The rule-based system will control switches and various resources. The sensor data from the AIS and the context monitor will be fused using the Dempster-Shafer theory of evidence (Shafer 1976). Based on the fused data, policies that are appropriate will be executed.

One of the purposes of our proposed system is intrusion detection. With this in mind, there are policies that are focused on this activity. For one thing, if packets from the network come in and the timestamp of these packets are a certain time away from the system's current time, a flag will be raised. If continuous traffic is coming from one specific IP address, it could be indicative of a Denial-of-Service attack. In this case, the system may block packets from that particular IP address. For any input coming into the system, it is assumed that the input is in a standard format. However, if the input appears to be irregular, such as a password having many symbols not numerical or alphabetical, then a warning flag may be raised. There are also policies related to internally monitoring the system. The system should run fairly consistently, where the CPU usage may spike or idle depending on certain actions occurring on the system. If there are times when the CPU spikes or peaks and none of the activities that normally causes this are occurring, then the system may take actions to run a diagnostics check on it. If enough memory is consumed, the system may run a heuristic to delete items that it considers to no longer be necessary. The AISs in the machine are developing detectors to detect anomalies externally and internally. The policies can have additional steps that state that if it catches anything the detectors from the AISs do not, it will prompt the AISs to adjust its detector creation strategy.

enough memory is consumed, the system may run a heuristic to delete items that it considers to no longer be necessary. The AISs in the machine are developing detectors to detect anomalies externally and internally. The policies can have additional steps that state that if it catches anything the detectors from the AISs do not, it will prompt the AISs to adjust its detector creation strategy.

Within any automated system, faults can occur. Faults can be described as unexpected changes from the normal system condition. The area of research dedicated to detecting faults is referred to as the Abnormal Event Management (AEM) in research done by Laurentys et al. (Laurentys et al. 2010). More specifically, the AEM deals with detecting, diagnosing and correcting abnormal conditions in real-time. An AIS can develop detectors that have a great information processing capability, pattern recognition and learning ability. These abilities can be applied towards creating detectors that can detect faults and take appropriate action. In the scope of our problem, the faults would be internal such as if something goes wrong with disk mirroring, or memory swapping from the disk. The self set of a detector would be the "normal" state of the system, and non-self would be anything that is different enough from the baseline of the system.

Realization of the Self-CHOP Properties

A major aspect of protection is sensing attempted intrusion, which is a standard task for AISs, detecting "self" and "non-self." We can generalize this function to perceiving danger (in the environment) in general and perhaps even to recognizing a need to adapt to a change in the environment. The general notion is that of monitoring the environment to protect self. The intent is that an AIS will identify a threat and characterize it sufficiently so that the policy engine may activate resources as per the applicable policies. The expert system may consult with AISs in the course of enforcing a policy.

In a similar vein, we can have the AISs detect when the system is not "itself" and in need of repair; this is the heal CHOP attribute, and what is monitored is self. The role of the AIS here is to identify faulty behavior and to characterize it so that the expert system may activate resources as per the applicable policies. Again, the policy engine may consult with AISs in the course of enforcing a policy.

This vigilance regarding the system's own behavior may extend to self-optimization if we have a baseline reading of system behavior. An AIS would help us distinguish acceptable patterns that deviate from this pattern (as when some resource is accessed) from unacceptable patterns. Something similar could address self-configuration. The most obvious occasions for self-configuration, however, are where a new device is attached. In such cases, a simple signal from a context monitor would suffice to provide the policy engine all the information needed to apply policies.

Conclusion & Future Work

We have sketched the autonomic system we are developing that uses artificial immune systems (AISs) augmented with a context monitor to provide data to a rule-based policy engine. The architecture is conceptualized as the AISs and context monitor providing sense data to the policy engine, but not that the data is also from the internal state of the system. We discussed how the system supports the self-CHOP properties: self configuring, self-healing, self-optimizing, and self-protecting.

We are implementing our autonomic architecture on a stock workstation that is attached to the Internet (inviting intruders) and to which we can attached multiple accessories (requiring self-configuration). In the future, we plan to investigate autonomic cyberphysical systems. In the future, we shall look into other biological metaphors and implement some into our system.

References

Abowd, G. D.; Dey, A. K.; Brown, P. J.; Davies, N.; Smith, M.; and Steggles, P. 1999. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, 304–307. Springer.

Abraham, A. 2005. Rule-based expert systems. *Handbook of measuring system design*.

Bradshaw, J. M.; Uszok, A.; and Montanari, R. 2014. Policy-based governance of complex distributed systems: What past trends can teach us about future requirements. *Agile Computing*.

Brownlee, J. 2011. *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee.

Carey, V. K.; Lewis, D.; and Wade, V. 2004. Automated policy-refinement for managing composite services. *M-Zones White Paper June* 4:114–130.

DasGupta, D. 1993. *An overview of artificial immune systems and their applications*. Springer.

De Castro, L. N., and Von Zuben, F. J. 2000. Artificial immune systems: Part i–basic theory and applications. *Universidade Estadual de Campinas, Dezembro de, Tech. Rep* 210.

Forrest, S.; Perelson, A. S.; Allen, L.; and Cherukuri, R. 1994. Self-nonself discrimination in a computer. In *null*, 202. Ieee.

Friedman-Hill, E. 2013. Jess, the rule engine for the java platform. *Java Expert System Shell, http://jessrules.com, United States*.

Hofmeyr, S. A., and Forrest, S. 2000. Architecture for an artificial immune system. *Evolutionary computation* 8(4):443–473.

Huebscher, M. C., and McCann, J. A. 2008. A survey of autonomic computingdegrees, models, and applications. *ACM Computing Surveys (CSUR)* 40(3):7.

Kephart, J. O., and Chess, D. M. 2003. The vision of autonomic computing. *Computer* 36(1):41–50.

Kephart, J. O., and Walsh, W. E. 2004. An artificial intelligence perspective on autonomic computing policies. In *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, 3–12. IEEE.

Laurentys, C.; Ronacher, G.; Palhares, R. M.; and Caminhas, W. M. 2010. Design of an artificial immune system for fault detection: a negative selection approach. *Expert Systems with Applications* 37(7):5507–5513.

Shafer, G. 1976. *A mathematical theory of evidence*, volume 1. Princeton university press Princeton.