Volume 10 Issue 04 Published, November 9, 2006 ISSN 1535-864X DOI: 10.1535/itj.1004

Intel® Technology Journal

Autonomic Computing

Advances in computing, communication, and software technologies have resulted in explosive growth of information services and their underlying infrastructures. To deal with the resultant complexity, researchers are taking design approaches modeled after biological systems, a research area referred to as Autonomic Computing (AC). This issue of Intel Technology Journal (Volume 10, Issue 4) reviews the research work at Intel Corporation on this important direction for future computing.

Inside you'll find the following articles:

More information, including current and past issues of Intel Technology Journal, can be found at:
http://developer.intel.com/technology/itj/index.htm

Volume 10 Issue 04 Published, November 9, 2006 ISSN 1535-864X DOI: 10.1535/itj.1004

Intel® Technology Journal

Autonomic Computing

Articles

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

By Lin Chao
Publisher, *Intel Technology Journal*

A fundamental tenet of autonomic computing is to increase the intelligence of individual computer components so that they become "self-managing," i.e., actively monitoring their state and taking corrective actions in accordance with overall system-management objectives. The autonomic nervous system of the human body controls bodily functions such as heart rate, breathing and blood pressure without any conscious attention on our part. The parallel notion when applied to autonomic computing is to have systems that manage themselves without active human intervention. The ultimate goal is to create self-managing computer systems.

The motivation for autonomics is one of rapid growth and complexities in computer systems. Annually, the number of connected computing devices is expected to grow at 38% [1]. To manage this complexity, the human labor cost is exceeding equipment costs by a ratio of up to 18:1 [1]. This results in complexity in the computer networks making them hard to control manually by human operators. The economics of this highlights the necessity for autonomics in computer systems.

The six papers in this issue of *Intel Technology Journal* (Volume 10, Issue 4) review in depth the research work at Intel Corporation on autonomic computing, an important direction for future computing.

Platform Support of Autonomic Computing: an Evolution of Manageability Architecture
The first paper explains the Intel® technologies that provide platform support for autonomics. Specifically, these are computer platforms with sufficient support and on-board intelligence to enable autonomic capabilities, such as self-healing and self-protecting, as well as discovery and asset tracking. To achieve this, dedicated platform resources and firmware with well-defined standard interfaces implement a set of management and autonomic capabilities. This paper also explains the Intel® Active Management Technology (Intel AMT), which is the first Intel product that supports autonomic computing.

Service Orchestration of Intel-Based Platforms Under a Service-Oriented Infrastructure
The second paper describes research on Service-Oriented Infrastructure (SOI) that enables higher-level service orientation and autonomic computing. We demonstrate how the concept of platform as a service (PaaS) may be applied to real-world Information Technology (IT) operations. The results show that SOI is viable and PaaS is achievable.

Standards for Autonomic Computing
In the third paper, we highlight important standards that enable components from heterogeneous sources to interact with each other. This interaction is fundamental to enabling intelligent decision making at the lowest possible level in the systems management hierarchy. We describe the standards required for external interfaces for autonomic elements such as Web Services (WS) Management and WS Distributed Management. We also explain the general design philosophy of these two approaches.

Towards Autonomic Enterprise Security: Self-Defending Platforms, Distributed Detection, and Adaptive Feedback
In the fourth paper, we describe three building blocks that help realize the ultimate goal of "autonomic operation" of the enterprise. First, we describe the notion of self defending platforms which enable an end-host to detect program-level anomalies and unauthorized modifications. Next, we describe the concept of distributed detection and inference, where end-hosts collaborate regarding the state of the entire network. Finally, we discuss an adaptive policy management architecture that supplements the self-defense and distribution detection capabilities.

Machine Learning for Adaptive Power Management
In the fifth paper, we describe how a machine-learning methodology could be applied to adaptive power management. We propose a system that learns when to turn off components based on user patterns. We describe the challenges of building such a system and explore a range of solutions.

A Self-Managing Framework for Health Monitoring
In the sixth paper, we propose a self-managing framework for health monitoring using body-wearable bio-devices that can reduce the doctor's intervention for patient management. We illustrate three usage models where this self-managing framework can be applied: remotely monitoring patients at home; monitoring fetal well-being in a maternity ward; and monitoring critical patients in an Intensive Care Unit (ICU).

Today, as networked devices grow rapidly, the complexities of managing them necessitates that we work both in industry and in academia to explore how to build modern, networked computing systems that can self-manage. Research in this complex area of computer science can help foster a whole new type of computer systems and capabilities for the next decade.

[1] Wikipedia on autonomic computing http://en.wikipedia.org/wiki/Autonomic_computing

Foreword

By Mazin Yousif
Principal Engineer
Corporate Technology Group, Intel Corporation

Advances in computing, communication, and software technologies have resulted in explosive growth of both information services and their underlying infrastructures. Information services are inherently complex, dynamic and heterogeneous, as are the large information infrastructures that support them. This is typified by the Internet, which comprises a large number of independent computing and communication resources. The amalgamation of the two results in management complexities that can easily break current computing paradigms and render both of them brittle, unmanageable and insecure. The shortage of skilled IT professionals in the current growing global economy further exacerbates the problem. To deal with such complexity, dynamism and uncertainty, researchers are taking design approaches modeled after biological systems. This research area is referred to as Autonomic Computing (AC).

IBM [1] coined the term *Autonomics* in 2001 and defined four main features to highlight it, namely, self-healing, self-protection, self-optimization and self-configuration. Since then other terms such as *Organic Computing* and *Self-managed Computing* have been applied to similar concepts, and the terms have been expanded to include features such as self-recovery, self-diagnosis, self-adaptation AC is inspired by the human autonomous nervous system, which handles human body complexities and uncertainties without requiring conscious effort. Clearly, the goal of AC research is to design computer infrastructure that manages itself with minimal human intervention. Of course, this is easier said than done. The first step in achieving the AC vision is to establish a detailed understanding of the underlying environment, including resource specifications, workload characterization, business processes and workflow, and service-level agreements. Only then can technologies such as provisioning, virtualization, monitoring, automation and feedback control be deployed to build AC features such as self-protection, self-healing and self-optimization.

Two major requirements must come together to build AC environments: (1) AC systems must be built of self-managing hardware and software components in which AC-related features are built into the architecture (not bolted-on); and (2) the environment must include a comprehensive management framework and architecture that facilitates building and deploying AC systems. Intel Corporation has been diligently working on both fronts, and is at the forefront of adding autonomic features to its products. Intel® Active Management Technology (Intel® AMT), is the first product incarnation of a framework and dedicated platform execution environment for AC. Intel has also been quite active in the AC research community, not only working on new AC research, but also developing vehicles to make it easier for the AC community to conduct related research in more realistic settings with a wide range of usage scenarios.

Although the AC research community has delivered good results so far in many AC-focused conferences and workshops, AC-related research is still in its infancy given the broad scope of the vision. One obvious reason is the correspondingly wide research scope, which can easily be grouped into two buckets:

(i) AC Systems' Architecture, including (a), self-managing components such as processors, memory, network, storage, software blocks, and operating system; (b) guaranteeing that systems composed of self-managed components will always be autonomic, enforcing system-level policies and goals through establishing effective feedback control mechanisms principles such as stability, observability, and controllability; and (c) handshaking/interfacing among components within an AC system and among AC systems.
(ii) AC Systems' Intelligence, including fundamental sciences such as machine learning, heuristics, statistical computing, fuzzy logic, probabilistic reasoning and tracking/forecasting. Clearly, this bucket includes the widest scope and requires a great deal of attention.

Along with research work, there are many technical challenges such as:
(i) AC Systems' prototyping, including advanced development of AC systems from subsystems with capabilities that allow us to emulate various autonomic scenarios
(ii) AC Support Ecosystems, including the software and hardware such as tools, orchestration software and lifecycle management needed to deploy AC systems in datacenters
(iii) AC standards for resource description, communication and messaging, etc.

In summary, we have made big strides in our efforts to achieve the AC vision, but there is still a great deal that needs to be done.

Reference

[1] Kephart, J. and Chess, D., "The Vision of Autonomic Computing," *IEEE Computer, 36(1)*, 2003, pp. 41–50.

Technical Reviewers

Robert Adams, Corporate Technology Group
Ravindra Bodhe, Corporate Technology Group
Bert Cave, Intel Information Technology
Prashant Dewan, Corporate Technology Group
Tom Gardos, Intel Information Technology
Ran Gilad-Bachrach, Intel Information Technology
Toby Kohlenberg, Intel Information Technology
Kai Miao, Intel Information Technology
Dennis Morgan, Intel Information Technology
Vijay Tewari, Corporate Technology Group
Georgios Theocharous, Corporate Technology Group
Chris S. Thomas, Sales and Marketing Group
Johan Van De Groenendaal, Digital Enterprise Group
Mazin Yousif, Corporate Technology Group

THIS PAGE INTENTIONALLY LEFT BLANK

Platform Support of Autonomic Computing: an Evolution of Manageability Architecture

Lenitra M. Durham, Corporate Technology Group, Intel Corporation
Johan van de Groenendaal, Digital Enterprise Group, Intel Corporation
Jackson He, Digital Enterprise Group, Intel Corporation
Jim Hobbs, Information Technology, Intel Corporation
Milan Milenkovic, Corporate Technology Group, Intel Corporation
Mazin Yousif, Corporate Technology Group, Intel Corporation

Index words: autonomic computing, dynamic data center, platform manageability, IT agility, service-oriented architecture, self-managed infrastructure, utility computing

ABSTRACT

Autonomic Computing (AC) is maturing from a design philosophy to an emerging set of technologies and products that addresses the complexity of managing today's heterogeneous data centers and computing environments. This overview paper explains our motivation and outlines our technologies that provide platform support for AC. Specifically, we are developing platforms with sufficient support and on-board intelligence to enable autonomic capabilities, such as self-healing and self-protecting, as well as features such as discovery and asset tracking, even when the host Operating System (OS) is inactive. To achieve these ends we are dedicating select platform resources and firmware, both exposed via well-defined standard interfaces, to implement a set of management and autonomic capabilities, and in the future, we hope to extend these platform autonomic capabilities, with appropriate management policies, to groups of platforms and eventually to the entire data center. Such interconnected autonomic platforms will provide the infrastructure and fabric to support service-oriented, grid, and utility computing.

This paper also expounds on the Intel® Active Management Technology† (Intel AMT) which is the first product incarnation of a framework and dedicated platform execution environment for AC. We also discuss how manageability architectures need to evolve to support autonomic behavior at a group level, such as defining interactions among platforms within a group to collectively deliver on specific goals and implement group-level policies. We cite examples related to malware detection and power management that illustrate how this new approach to managing IT infrastructure works.

INFORMATION TECHNOLOGY OVERVIEW

Today, Information Technology (IT) departments are plagued by increasing complexity, poor utilization of assets, space, and high power consumption. Further, system management in such environments is fragmented and labor-intensive requiring considerable human involvement for mundane functions such as configuration and provisioning. Such common inefficiencies in data centers force IT managers to provision their data centers for peak loads resulting in low average resource utilization and availability in the range of 99.99% or possibly less. Current data center deployments tend to have strong and static binding between servers and applications, as well as between servers and administrators, due mainly to a lack of technology to deploy otherwise.

To improve the efficiency of assets, space, and power, IT managers turned to virtualization, which provides the ability to consolidate multiple applications onto a single server (a server could be stand-alone, rack-mounted or bladed). These servers may include advanced technologies that support multiple logical and physical partitions within each *physical machine* such as virtualization, partitioning hooks, and multiple cores. These advanced technologies have increased the cost and complexity of platforms and require that management solutions adapt to deal with them. Unfortunately, management solutions have not kept pace.

Information Technology Evolution

It is obvious that the resource consolidation currently taking place in data centers is only the first step in the evolution of IT environments. Enterprise IT departments ultimately want to manage their computing resources and infrastructure to support business processes and services in accordance with business objectives. This is sometimes referred to as the Service-Oriented Enterprise (SOE). The motivation for moving to an SOE include guaranteeing service and site availability and satisfying a certain Quality of Service (QoS), and Service Level Objectives (SLO) while optimizing across constraints such as operational expenses and business value. The result is a reduced data center total cost of ownership.

Our vision for supporting SOE in future data centers is to make them into pools of resources that can be accessed as a utility. Such grid-like data centers allow their resources to be dynamically allocated and deallocated based on the compute and I/O resource requirements of running applications. Achieving such a grid vision requires a dynamic infrastructure that decouples services from infrastructure and incorporates the ability to map applications to infrastructure adaptively. This is known as Service-Orientated Infrastructure (SOI). In addition, treating a data center as a utility allows clients to pay only for the necessary resources to run their services, when appropriate metering tools are applied.

Service-Oriented Infrastructure

As IT departments move to Service-Oriented Architecture (SOA), applications may be decomposed and common elements identified as shared, reusable services. Services may be purchased, outsourced, or (if otherwise unavailable) created; applications that implement business processes then evolve to use the shared common services.

There are significant advantages to executing these services in a dynamic environment. As we move to a re-factored, service-oriented application environment, the infrastructure must be able to adjust to changing loads by creating additional service instances as needed (as well as extinguishing them as appropriate).

A service-oriented, autonomic infrastructure can provide an environment that will assign resources as needed under given policies. Most reviews of SOA cite the need for stateless interactions [1, 2]. When services are stateless, there generally is freedom to add additional instances as necessary to handle load, reduce latency, or guard against equipment or facility failure (for business continuity of disaster recovery). This freedom means that we can scale out services by adding computing resources where needed and of the size needed. Just as a utility infrastructure may be compared to a power utility, this provides an opportunity similar to *microgeneration* [3] where power can be generated close to the load being served by smaller than usual power plants—in essence, an alternative method of scaling.

New service instances created to handle load might be automatically created alongside existing ones. Keeping an additional instance nearby may have advantages: for example, it might ease routing and provide the ability to make local decisions based on policy. Within a group of servers, or even within cores of a multicore implementation, a local decision can be made to scale out and/or cut back instances, without having to refer the decision to another management entity. Creating an additional instance in a remote location could provide a natural way to handle failover and disaster recovery, since appropriately positioned instances could take on loads that previously had been served by failing components. Creation (or extinction) of additional service instances in the appropriate location could take place autonomically, perhaps within a platform group, without the need for intervention from a centralized console.

IT ENVIRONMENT IMPLICATIONS

Many current implementations of manageability were designed for an environment in which services are statically assigned to resources. As the environment changes to one where resources are dynamically allocated, management must change to address the increased complexity of the environment.

Ideally, this will be done in a way that reduces visible complexity. Rather than binding a specific service instance to a specific resource, services should be related to a class of resources. The mapping of a specific instance of a service to a particular server is best accomplished in the infrastructure.

These are two of the core components required to achieve this vision:

1. *A model*. This would capture both static and dynamic states of the IT infrastructure, services, and applications, together with their relationships, resource requirements, and SLOs.
2. *Autonomics*. This would enable the IT infrastructure to achieve the required SLOs by making policy-based autonomic decisions.

In the remainder of this paper, we focus on these two core components.

MODEL

IT is evolving from deploying services by statically assigning them to resources, to dynamically allocating services to available resources. The ability to map services

to the infrastructure, based on the required SLOs, requires a model of the resources, their capabilities, and overall topology. This model ultimately enables the computing utility to be dynamic and self-regulating, thereby facilitating scale-out of services.

The model serves as a foundation for effective management of the IT facility [4]. It is typically divided into a static and dynamic component. The static model typically maintains these:

1. *Asset information*. This comprises information on all the elements within the IT facility (e.g., hardware, OS, and software).
2. *Static topology*. This maps the relationships both physical and logical between devices (e.g., network, compute, storage, clients, OS, and applications).

The dynamic model captures the *current state of the IT environment at an instance in time*. Typically, the dynamic model may include dynamic topology that maps the current relationship (typically logical) in the environment, such as between devices and between logical platforms and currently deployed services.

Benefits of the Model

The ability to deploy new *builds* to classes of servers rather than to individual servers will also ease the task of change and release management, which will be concerned with an inventory of elements and class relationships, rather than specific allocations of resources. This effectively changes the meaning of *deployment*. In addition, the ability to deploy to a logical representation of a computing resource makes it possible to refactor the infrastructure, giving additional flexibility to the implementers, while providing stability through virtualization. An example of such refactoring might be changing from an enclosure in which there were many server blades, each having a complement of processors, memory, storage, and network components, into an enclosure that separates compute, storage, and network functionally.

Without the model, decisions on how a service should be managed over its lifecycle could not be made. The model is used to analyze whether a service is meeting its desired state by collecting metrics from the service and the resources hosting that service. It can also be used to make informed decisions on available capacity to deploy the service in such a manner that it performs within the required SLOs.

AUTONOMICS

Autonomic systems [5, 6] represent an approach to managing complexity by making individual system nodes and components self-managing. Such systems manage themselves by monitoring their operation, detecting any anomalies, and then adjusting accordingly to achieve normal operation.

Autonomic Computing (AC), also referred to by other terms such as Organic Computing and Self-Managed Systems, is inspired by the human autonomous nervous system, which handles the complexities and uncertainties of the human body without requiring our conscious efforts. AC emerged as a new strategic and holistic approach to the design of complex distributed computer systems, aiming at realizing computing systems and applications capable of managing themselves with minimum human intervention. In [5, 6] IBM researchers defined autonomic managers as being responsible for monitoring a managed element, and creating and executing plans based upon analysis of the data and knowledge they have acquired. Thus, autonomic systems are represented in closed-loop form consisting of four stages: monitor, analyze, plan, and execute, as shown in Figure 1.

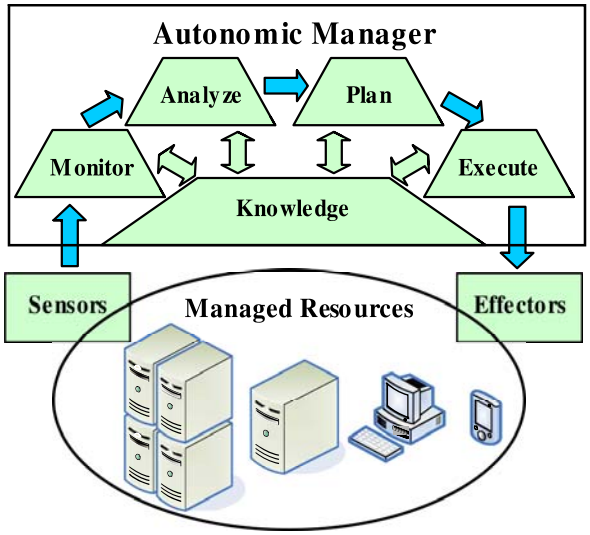

Figure 1: A typical abstract autonomic framework

The autonomic manager monitors the managed resources it controls and analyzes the data received. Based upon the data received and analyzed, the autonomic manager constructs and executes plans to achieve the management goals in accordance with the policies and rules in effect. It accumulates and uses knowledge (policies and rules) from observations, past experiences, and updates received from its peers and other components of the management hierarchy, such as management consoles. The manageability interface between the autonomic manager and the managed resources allows the autonomic manager to "sense" data from the managed resources and to "effect" the desired actions.

Self-Management Attributes

The following four underlying attributes were originally defined to constitute self-management.

1. **Self-configuring** is a system's ability to change its configuration automatically in reaction to runtime changes or to assist in self-healing, self-optimization, and/or self-protection. For example, if a hard memory error is detected on a memory bank, a platform can isolate the particular memory bank allowing the system to continue functioning until the faulty part can be replaced.
2. **Self-healing** is the ability of a platform to effectively recover when a fault occurs. Self-healing can be either reactive or proactive. A reactive self-healing platform attempts to correct or isolate a fault once it has occurred. A proactive self-healing platform attempts to predict whether a fault may occur and takes appropriate action to ensure the health of the system is maintained.
3. **Self-optimization** is the ability of the system to optimize its operations based on a given operation profile. This involves monitoring its operation and optimizing accordingly, given a set of policies. It may also react to dynamic policy changes within the platform as indicated by a user.
4. **Self-protecting** is the ability of a system to defend itself from accidental or malicious external attacks by being aware of potential threats and being able to handle those threats.

The above list of self-management attributes has been growing steadily and substantially covering features such as self-anticipating, self-adapting, self-critical, self-destructing, self-diagnosing, and self-recovering. We expect that the list will continue to grow.

For a system to meet the self-management objectives, it must be aware of its internal state (self-awareness) and current external environment (environment-awareness) [7]. Self-awareness and environment-awareness are achieved through the ability of a platform to collect raw internal (self-monitoring) and external data (environment monitoring) that is used for the following:

- *Data aggregation*. This automatically transforms raw data gathered over time into information upon which predictions, actions, and strategies are based.
- *Data analysis*. This is the analysis of raw and aggregated data that is used to aid in self-healing, self-protection, and self-optimization.

As a platform monitors its internal and external environment, changes may be detected that require the platform to adjust itself accordingly (self-adjusting).

The self-management features of an autonomic platform are dependent on one another. Using our memory example again, if memory fails in the platform, the platform will need to take corrective action (self-healing and self-configuration), optimizing itself in an attempt to continue to meet SLOs (self-optimization). To illustrate the dependencies between the self-management features, a more representative taxonomy of an autonomic system is shown in Figure 2. The figure shows the enabling technologies (e.g., virtualization and automation) that are required to build a self-managing system (as represented by the disc) with capabilities (represented as petals) such as self-optimizing, self-healing, and self-protection.

The figure also illustrates the environment and resources that an autonomic system needs to comprehend, such as the hardware, software, workload, and business requirements. Note that the petals protrude outside the self-managed disc to indicate that systems cannot internally self manage all possible scenarios: there will always be a need for outside intervention, since the self-managed object may be part of a larger self-managed construct. Petals also overlap to indicate that a change in one self-management function may impact one or more other self-management functions. For example, a self-healing action may need to be followed by a self-optimizing action for best performance, as discussed in the memory example.

Autonomic Computing Environment

In an AC environment, there is often an inherent assumption that actions are purely based on predictive approaches. Although predictive technologies are an important evolutionary step in building better autonomic platforms, it is possible to have an autonomic platform that is *reactive* in nature. In other words, its autonomic behavior is driven by enforcing policies reacting to environmental monitoring data already collected. This approach to AC is the most common form found today.

Reactive autonomic platforms are, however, limited in their ability to achieve a high level of self-awareness and therefore may not fully achieve the desired AC operation. In other words, the given policies limit the effectiveness of the AC capabilities.

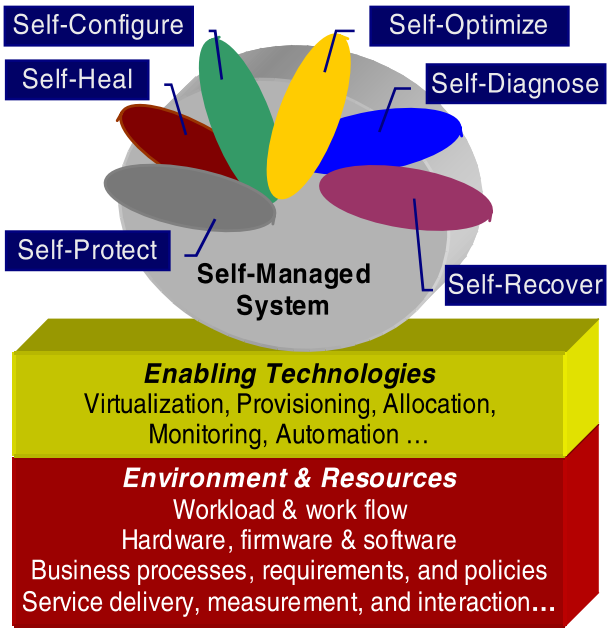

Figure 2: Taxonomy of an autonomic system

Proactive-based AC environments rely on machine learning techniques such as Artificial Intelligence (AI) to analyze collected data and predict operational anomalies before they occur. This approach clearly enables us to achieve the full AC vision. However, this does not come for free; predictive techniques require additional resources resulting in increased complexity.

An ideal AC environment would incorporate a combination of both proactive and reactive approaches. The ability for a platform to take proactive actions needs to be controlled by policies that describe not only what autonomic actions may or may not be taken, but also what the desired normal state of the platform should be.

When autonomic actions are based on local information and limited environmental information, the ability to make accurate decisions may be limited. This may result in a high number of perceived conditions that are incorrectly identified. If one looks at how, for example, enterprise management solutions work today, information collected by a platform is consolidated in a management console, where the management console can leverage the received data from multiple data sources and make a more informed and accurate decision on what has or may have happened on a particular platform.

Group Interactions

When designing an autonomic platform, it is critical that it be able to interact with other platforms in the environment, thereby increasing its own knowledge by using the collective intelligence of other autonomic platforms. The additional information aids in the decision process allowing the platform to make decisions based upon local data and data collected from other platforms. Two examples illustrate the importance of distributed intelligence.

Data centers rely on Network Intrusion Detection Systems (NIDS) to analyze network traffic for patterns and hints in data to determine whether potential threats exist. NIDS is effective only after it examines a sufficient portion of the network traffic. A Host-based Intrusion Detection System (HIDS), on the other-hand, can only make decisions based on what it sees, thus relying on knowledge in the form of signatures and heuristic patterns that it has been provided a priori. By extending HIDS to leverage other HIDS on the network, the reliability of a HIDS increases. The first example is described in the paper "Towards Autonomic Enterprise Security: Self-Defending Platforms, Distributed Detection, and Adaptive Feedback" [8] in this issue of the *Intel Technology Journal.* The authors describe the three building blocks for an end-to-end autonomic enterprise, namely detection, self-defense, and adaptive policy management. They explain how Intel technologies such as Intel AMT and Intel-led standards initiatives are used to create a self-defending platform and how distributed intelligence can be used to defend the enterprise by corroborating the likelihood of infection.

The second example deals with self-optimization. Autonomic systems maintain operations when system conditions vary by monitoring their state and creating and executing plans based upon analysis of the data and on the knowledge they have acquired. A critical component of autonomic systems is their ability to handle uncertainty in the perceived state and the effect of their actions. The article "Machine Learning for Adaptive Power Management" [9], also in this issue of the *Intel Technology Journal*, describes how Partially Observable Markov Decision Processes (POMDPs) are used for modeling autonomic systems and how a specific POMDP model is used for an adaptive power management system in a laptop.

PLATFORM AUTONOMIC REQUIREMENTS AND ARCHITECTURE

We begin this section by discussing key issues with today's management models. We then describe how manageability architectures need to evolve to support autonomic behavior such as monitoring changes in the environment, making autonomous decisions at the system and group level, and providing policy-driven services such as dynamic provisioning and load balancing. Finally we present our approach to achieving the self-management vision, via platform autonomics.

Manageability Solutions Today

Most commercial management solutions operate by placing software agents in the host execution environment to monitor the health of the OS and applications and to optionally control their operational states. Such management agents implement sensors and effectors for the OS, applications, and in some instances, the underlying platform hardware. Agents are designed to communicate with a remote console (that usually resides at a central management location) to provide information and control interfaces to human operators. In the basic but typical case, local instrumentation data gathered by agents are sent to a database to be analyzed and, when necessary, used to trigger a control decision based on appropriate automation policies. Autonomics may be added to this basic structure in the form of local policy and knowledge engines. These reside in the local management agents and close the control loop in response to changes of the observed local state. Local action eliminates the latency and overhead of a round-trip communication with the remote console and possibly the operator. Other benefits of autonomics include a reduced volume of management data traversing the system (since many stimuli are processed locally near the origin) and increased scalability, due to a reduced load on the management console and its database.

A primary weakness of implementing manageability and autonomics in the same execution environment with the applications and OS that they monitor is that malfunctions of the monitored environment may impair the agents' operation forcing the agents' lifecycle to be limited by the lifecycle of the environment. Put simply, when the OS crashes it takes the agent down with it. As a result, no management is possible when the OS is not running.

Intel® Platform Autonomics Approach

Our platform autonomics approach rectifies the aforementioned problem by providing a separate execution environment for the autonomic manager. This results in an independent lifecycle for the autonomic manager, which is expected to function in both pre-OS and post-OS states. In the pre-OS state, it manages configuration and provisioning actions. An obvious benefit in the post-OS state is the ability to perform forensic analysis by examining the machine state exactly as it was left by the crash. When coupled with event logging within the manager, this can be a powerful tool in determining the root cause of failures.

With proper design, a platform autonomic manager can be decoupled from the power states of the host processor, so that it is powered even when the host processor is off. This is very useful for performing host power on/off operations, performing hardware setup and configuration, and automating provisioning that facilitates platform self-configuring behavior and attributes. In the next section, we describe our first step towards achieving our platform autonomics vision.

As of this writing, most autonomic systems and prototypes reported in the literature seem to be implemented in higher software layers, mostly in user space with perhaps some OS modifications. Our research focus is on platform support for autonomics [10]. Specifically, we are looking at dedicating platform resources and firmware to implement a set of management and autonomic behaviors that are exposed via well-defined interfaces. Our long-term vision is to create platforms with on-board support and intelligence that make them discoverable, configurable, self-managing, self-healing, and self-protecting even when the host OS is not active. Future platforms may provide the agile infrastructure to support the evolution of dynamic, autonomic, distributed computing as described in [11].

To deliver the autonomic vision, systems and applications need to be built out of autonomized hardware and software components. Obviously, the granularity of a component will be an on-going research topic. Regardless, an autonomized component will need to do the following:

- Characterize itself including introspection, discovery, and self-description in a machine-readable way.
- Dynamically monitor its ambient and surrounding environment.
- React intelligently, at least locally, to changes in the environment.
- Interface with other components for communication, particularly with components responsible for managing the whole system.

Core Autonomic Platform Manager Requirements

Overall, for a platform to be effectively managed autonomically, a number of core components are required. We discuss each of these below.

Standard Out-of-Band External Interfaces

One way for platforms to collect environmental information from their surroundings is to communicate with other platforms through Out-of-Band (OOB) interfaces. OOB interfaces are commonly independent of the host operating environment, which could be an OS or a Virtual Machine Monitor (VMM) with one or more OSs. In the article "Standards for Autonomic Computing" [12], in this issue of the *Intel Technology Journal*, a more detailed overview is given of the importance of

standardization in autonomic computing and the benefits of using Web services as external interfaces.

Internal Interfaces to Host Operating Environments
The interface to the host operating environment is needed to get visibility into core metrics from the host OS that may not be available on internal buses. In addition, some components of an autonomic solution may be deployed in the host operating environment to gather additional environmental information not visible otherwise.

Standard Internal Platform Interfaces
These interfaces are needed to dynamically discover and interact with sensors and effectors. Autonomic decisions are based on multiple sensors within the platform. Since the components vary from one platform configuration to another, the ability for the autonomic manager to discover and monitor these sensors dynamically is critical to an autonomic design.

Platform Container
A platform autonomic container is required to implement the autonomic functions. It may be provided in a variety of ways such as a dedicated microcontroller in the chipset or a plug-in option card. In any of these cases, the container is expected to have dedicated physical or virtual execution resources, such as processor and memory, supporting a software execution environment that is isolated and possibly different from the OS and user application execution environment of the *host* platform. Isolation from the host operating environment and separation of manageability and autonomic functions into a dedicated execution environment provides some fundamental advantages, resulting primarily in increased availability. While a separate container provides a number of benefits, it is only as useful as the autonomic functions implemented in it. These must be made available externally for use in various phases of the host system lifecycle—starting with pre-power, pre-OS states, assisting the OS when it is present, and taking over when it is not.

Inter-Platform Container Interfaces
These interfaces are used by autonomic managers to exchange information as in the aforementioned distributed malware detection and power management examples. The inter-platform container interfaces may be standardized, but in general are highly optimized trustworthy interfaces.

Given that these core components need to be embodied in platforms, Intel has taken the first steps to implementing its autonomic vision with Intel AMT.

INTEL® ACTIVE MANAGEMENT TECHNOLOGY

Intel announced the Intel Active Management Technology (Intel AMT) in 2005 to reduce the complexity and cost of platform management. Figure 3 shows a conceptual model of Intel AMT. It provides platform management services through standard Web services interfaces, and can operate in both in-band (IB) and OOB modes. The OOB mode is particularly useful when the OS is not available either because it has not yet been provisioned (bare metal state), or because it has crashed. In this scenario, Intel AMT continues to provide basic platform management functions accessible through a management console. When the OS is running, Intel AMT can interact with both the OS and management console to perform requested platform management functions. In effect, Intel AMT is the first incarnation of Intel's platform support and container for AC.

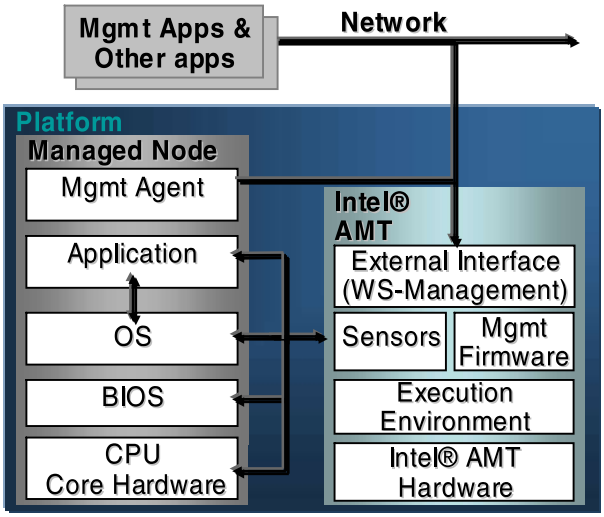

Figure 3: Intel® AMT conceptual model

Due to its OS-independence and availability regardless of systems state, Intel AMT is uniquely positioned to provide basic platform housekeeping functions needed for AC such as asset tracking and management, platform provisioning and re-provisioning, remote diagnosis, and repair. As indicated in Figure 3, Intel AMT services are implemented through a combination of hardware and firmware hooks built into a platform. Such services are persistent, *always on*, and interact with the OS and management consoles seamlessly as part of the overall platform service framework. Intel AMT also provides integrated security mechanisms that protect services from unintended use. It forms an integral yet autonomous part of the computing platform. In the coming years, Intel will continue to enhance Intel AMT to include more comprehensive platform management functions for self-managed infrastructure support.

Features of Intel AMT

Using built-in platform capabilities and popular third-party management and security applications, Intel AMT allows IT managers to better discover, heal, and protect their networked computing assets [13].

Discover–Platform Resource Inventory

Accurate platform, software, and hardware inventories are necessary for regulatory compliance as well as for closely managing maintenance contracts and software licenses. Asset tracking is also needed for proper monitoring and operation of the computing environment. Current software tools available for remote asset inventory may not track platforms that are powered down since they rely on software agents integrated into the OS. Such OS agents can easily be disabled or accidentally removed. As a result, lengthy and expensive manual inventory surveys are often needed to ensure complete and accurate results. By contrast, built-in manageability features of Intel AMT can help eliminate manual inventory costs by allowing asset management tools to remotely access systems regardless of their power state to read their hardware and software asset information. (Such information is usually stored in nonvolatile memory.) This requires that assets must be plugged into active power and active network sockets.

Heal–Addressing Failures in Compute Environments

An inoperable OS, corrupted applications, or hard drive failures often require one or more desk-side visits to diagnose and fix the problem. Proactive alerting and remote recovery capabilities of Intel AMT can reduce the number of desk-side visits by allowing IT managers to recover control of a failed PC, diagnose the problem, and potentially fix it remotely [14].

Protect–Protect IT Infrastructure from Network Threats

Security threats are present in today's compute environment. The damage they inflict on a network can result in significant downtime and loss of productivity for an organization. Intel AMT allows security patching of systems regardless of system power state. One can push patches so that systems are protected from vulnerability exploits before a virus hits. If a system is infected, the Intel® System Defense capability can help to contain the threat before it spreads to other PCs and affects the larger network. Intel AMT also has components that prevent tampering with critical software agents such as virus protection applications. This helps maintain the integrity of the system.

The advantage of Intel AMT over traditional agent-based solutions is that it is not just simply the *addition* of technology capabilities at the hardware and firmware level. It is an *integrated* end-to-end solution that delivers value directly to the end users. Intel has focused on enabling a platform manageability ecosystem through working with management console Independent Software Vendors (ISV) and IT end-users to make sure that Intel AMT delivers the value customers need and in the way they need it. Such end-user orientation at the platform solution level makes Intel AMT more appealing to IT customers at large.

In assessing IT value in a real environment, we have had many proofs of concept and studies of Intel AMT applications in Intel's own IT organization, as well as with many end-users worldwide. Based on extensive analysis, Intel IT estimates significant cost savings as presented in Reducing Enterprise Management Costs with Intel® Active Management Technology [15].

IT ADOPTION OF AUTONOMICS

As automatic processes match demand to resources, reconfigure the environment as needed, and reroute work around failing components, the human focus will shift to ensuring that services and business processes are functional. The availability of any particular server, disk or network component will be nearly irrelevant. These technical changes will result in changes to IT processes. Today, applications are often characterized through a *bottom-up* process, where the stack of an application is developed then tested to determine the configuration of the server that will provide the optimal performance. In an environment where applications are assembled using a number of shared services, and each service might exist in multiple instances and be used by many applications, this approach will give way to a *top-down* process that relies on capacity and performance measurements of services and on aggregations of resources as well as platforms.

Impact on Monitoring and Measurement

When measuring performance of an IT solution, much of the focus has been on the health of individual components, such as application servers and the applications themselves. This is often supplemented by end-to-end user experience metrics, and in many cases, by directly measuring the performance of business processes.

Measurement of business processes is already important: even if all servers are up, all services are operating, and all communication methods are available, there are other factors that can cause problems with performance or interfere with the completion of a business process. Measuring billing backlog or units in inventory may detect anomalies in a way that checking response times or server status will not.

In a self-managed environment where resources are assigned and reassigned dynamically, there may be no direct connection between the performance of a particular component and the health of the business process. On the one hand, sharing of a service means that its performance might affect many business processes. On the other hand, since services may exist in multiple instances, failure of a particular instance (or of an element of a pool of resources) may not have as much impact as the failure of a unique instance.

This means that measurement of business processes moves from important to critical. As a self-managed infrastructure assumes the responsibility for monitoring its components and working around problems, IT will increasingly turn its focus to the business processes that it supports. More attention can be given to monitoring business processes and end-to-end experience. This will help identify patterns of service demands that can be anticipated based on the business process cycle, and will allow IT to bring support of business processes to the forefront.

IT Roles

As AC becomes more entrenched in the IT infrastructure, the duties and responsibilities of IT professionals will also change. For example, in today's environments, there may be proportionately few people engaged in business process design and implementation, and many more involved in application implementation and server configuration. This will change over time due to the presence of the following:

- Facilities that assist business process implementers to map business requirements to reusable services that will accomplish their objectives.
- Tools that map the required services and other components to a logical representation of the infrastructure.
- A service-oriented AC infrastructure that can dynamically map the logical representation onto physical components, and keep services running through self-managing capabilities such as automatic allocation of additional resources as needed and migration of workload from failing components.

As a result, infrastructure design and implementation should be less resource-intensive, and generally should be accomplished outside of the context of a given project or program. The benefits of features such as self-configuration and self-correction will ease the routine operation and maintenance of the infrastructure. The result will be that IT personnel can focus on solving business problems rather than technical ones.

SUMMARY

Future dynamic data centers are evolving to adopt SOA and utility and grid computing that requires the management infrastructure to evolve accordingly. We have described how this evolution should take place to support autonomic behavior at a platform level as well as a group level. Our vision for supporting SOA is to transform data centers into pools of resources that can be accessed as a utility. Achieving this vision requires a dynamic SOI that decouples services from infrastructure, and incorporates the ability to map applications to infrastructure adaptively. Thus, a model which captures both static and dynamic states of the IT infrastructure, services, and applications, together with their relationships, resource requirements and SLOs is required. In addition, autonomics will enable the IT infrastructure to achieve the required SLOs by making policy-based autonomic decisions.

Autonomic systems will adjust to changing loads by assigning resources as needed under given policies. These systems manage themselves with minimum human intervention allowing IT personnel to focus on business processes. Our platform autonomics approach provides a separate container for the autonomic functions that are exposed via well-defined interfaces. The platform autonomic container provides increased availability by allowing the autonomic manager to operate in both pre-OS and post-OS states. Intel AMT is Intel's platform support and container for AC. It currently offers the ability to discover, heal, and protect computing assets using standard Web services interfaces. The features of Intel AMT will continue to be enhanced to provide the platform management functions needed for a self-managed infrastructure.

ACKNOWLEDGMENTS

We thank Bert Cave, Udayan Mukherjee, Steve Patzer, Vijay Tewari, Gregg Wyant, and Raj Yavatkar.

REFERENCES

[1] Hanson, J., "SOA: Refactoring Mainframe Applications into Dynamic Web Applications, Part 1," March 2005 at http://devx.com/Java/Article/27521*

[2] Morgenthal, JP, "Enterprise Architecture: The Holistic View: Managing Data in an SOA," Column published DMReview.com, November 2005 at http://dmreview.com/article_sub.cfm?articleId=1042437*

[3] Shaw, R. W., Jr., "Microgeneration Technology: Shaping Energy Markets," *The Bridge (National Academy of Engineering), vol. 33, no. 2*, 2003 at

http://nae.edu/nae/bridgecom.nsf/weblinks/MKUF-5NTLVQ*

[4] Hobbs, J. and Birkel, S., "Impact of the Utility Revolution in IT," in *Proceedings of the Intel ISTG Technical Community Conference 2005* (internal publication).

[5] Horn, P., "Autonomic Computing: IBM's Perspective on the State of Information Technology," *IBM Corporation,* October 2001 at http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf*

[6] Kephart, J. and Chess, D., "The Vision of Autonomic Computing," *IEEE Computer, 36(1)*, 2003, pp. 41–50.

[7] Sterritt, R., "Autonomic Computing," *Innovations Sys. Software Engineering vol. 1*, pp 79–88, March 2005.

[8] Dave, M. et al., "Towards Autonomic Enterprise Security: Self-Defending Platforms, Distributed Detection, and Adaptive Feedback," *Intel Technology Journal, Volume 10, Issue 4*, 2006.

[9] Theocharous, G. et al., "Machine Learning for Adaptive Power Management," *Intel Technology Journal, Volume 10, Issue 4,* 2006.

[10] Durham, L. et al., "Platform Support for Autonomic Computing: a Research Vehicle," in *Proceedings of the Third IEEE International Conference on Autonomic Computing*, pp. 293–294, June 2006.

[11] Milenkovic, M. et al., "Toward Internet Distributed Computing," *IEEE Computer*, 36(5), 2003, pp. 38–46.

[12] Tewari, V. and Milenkovic, M., "Standards for Autonomic Computing," *Intel Technology Journal, Volume 10, Issue 4,* 2006.

[13] Intel Corporation, "Built-In Manageability and Proactive Security for Business Desktop PCs," *Intel® VPro™ Technology Whitepaper* at http://download.intel.com/vpro/pdfs/vpro_wp.pdf

[14] DeLiberato, D. et al., "Intel and Cisco Collaborate to Improve Enterprise Security," *Technology@Intel Magazine*, September 2005 at http://www.intel.com/technology/magazine/communications/intel-amt-cisco-nac-0905.pdf

[15] Intel Corporation, "Reducing Enterprise Management Costs with Intel® Active Management Technology," *Intel Information Technology Whitepaper* at http://www.intel.com/it/pdf/active-management-technology.pdf

AUTHORS' BIOGRAPHIES

Lenitra M. Durham is a senior research scientist with Intel's Corporate Technology Group. She received her B.S. degree from Rensselaer Polytechnic Institute and her M.S. and Ph.D. degrees from the Georgia Institute of Technology. Her current research focus is manageability and autonomics. Her e-mail address is lenitra.m.durham at intel.com.

Johan van de Groenendaal is a chief manageability architect in the Digital Enterprise Group Client Platform Architecture and Planning. He received his B.Eng (Cum Laude) degree from the University of Pretoria and his Ph.D. degree from the University of Cape Town. His e-mail address is johan.van.de.groenendaal at intel.com.

Jackson He is a lead architect in Intel's Digital Enterprise Group End-user Platform Integration Group. He holds Ph.D. and MBA degrees from the University of Hawaii. His e-mail address is jackson.he at intel.com.

Jim Hobbs is a senior architect in the Strategy, Architecture, and Innovation division of Intel Information Technology. He has a degree in Information and Computer Science from the University of California, Irvine. His current interests include internal use of Intel technology and impact of SOA on infrastructure. His e-mail address is hobbs at intel.com.

Milan Milenkovic is a principal engineer in Intel's Corporate Technology Group. Milan received an M.S. degree in computer science from the Georgia Institute of Technology and a Ph.D. degree in Computer Engineering from the University of Massachusetts at Amherst. His e-mail address is milan at intel.com.

Mazin Yousif is a principal engineer in Intel's Corporate Technology Group. He received his Masters and Ph.D. degrees from Pennsylvania State University. His e-mail address is mazin.s.yousif at intel.com.

† Intel® Active Management Technology requires the computer to have additional hardware and software, connection with a power source, and a network connection. Check with your PC manufacturer for details.

THIS PAGE INTENTIONALLY LEFT BLANK

Service Orchestration of Intel-Based Platforms Under a Service-Oriented Infrastructure

Enrique Castro-Leon, Software Solutions Group, Intel Corporation
Mark Chang, Sales and Marketing Group, Intel Corporation
Jay Hahn-Steichen, Information Technology, Intel Corporation
Jackson He Digital Enterprise Group, Intel Corporation
Jim Hobbs, Information Technology, Intel Corporation
Guy Yohanan, Information Technology, Intel Corporation

Index words: web services, service orientation, IT infrastructure management, service-oriented architecture, service-oriented infrastructure, platform virtualization

ABSTRACT

Adoption of service orientation in all aspects of enterprise computing has gained momentum in recent years. This paper describes research on Service-Oriented Infrastructure (SOI) and key concepts that enable higher-level service orientation and autonomic computing. We demonstrate how the concept of platform as a service (PaaS) may be applied to a real Information Technology (IT) operational environment through a Proof of Concept (PoC) project based on Intel IT operation and Intel® Active Management Technology† (Intel AMT). The results show that SOI is viable and PaaS is achievable, with significant business benefits.

INTRODUCTION

Service-Oriented Architecture (SOA) in the enterprise allows for structural reduction in the cost of both deployment and operations of Information Technology (IT) systems in spite of increasing complexity. This complexity is often dictated by regulatory mandates such as Sarbanes Oxley (also known as the Public Company Accounting Reform and Investor Protection Act of 2002) and HIPAA (Health Insurance Portability and Accountability Act, 1996), and by the increasing integration of global, distributed, and perhaps outsourced resources, factors that can't be controlled by the CIO (Chief Information Officer) or the enterprise. Costs are minimized during provisioning by designing subsystems as services with an eye toward maximizing reuse and interoperability.

In an SOA environment, IT infrastructure can be made to behave operationally in ways similar to a self-adjusting organism: it is essentially regulated by business-defined Service Level Agreements (SLAs), and it can correct itself dynamically in an autonomic fashion whenever deviations from the SLA occur.

These provisioning and operations capabilities in turn require advanced infrastructure orchestration and management features around the hardware. The capabilities are attained by applying service orientation to the concepts of orchestration and management themselves under the umbrella of Service-Oriented Infrastructure (SOI). SOI extends service orientation from the enterprise as a whole down to bare-metal provisioning, when a computer system does not have software of any sort loaded. Using this paradigm, we can architect hardware to become an integral part of a service infrastructure, and hence we can talk about a hardware platform-as-a-service (PaaS) under which Intel platforms directly interact with service-oriented components in the IT infrastructure. A first implementation of such a concept is Intel Active Management Technology (Intel AMT), which provides direct platform management services independent of the OS.

In this paper, we first discuss the different aspects of the SOI concept and illustrate how Intel AMT could be applied in this context. In addition, we describe a Proof of Concept (PoC) conducted by Intel IT to further demonstrate the use of Web-Services (WS-*) technologies for enterprise manageability based on Intel AMT.

It might seem that service orientation is applicable only to large enterprises with the critical mass to provide the reuse opportunities to justify a service-oriented design. This is correct under what we call the "inside-out" SOA paradigm that assumes an internal SOA effort serving the enterprise around it. We envision an "outside-in" paradigm

applicable to the small and medium business (SMB) space and to emerging markets where the organization being served is actually a whole ecosystem, potentially larger than almost any enterprise. PaaS would facilitate the provisioning and orchestration of services by independent service providers, in addition to IT organization. It is possible to envision a rich ecosystem where turnkey services are delivered through multiple levels of aggregation by collaborating service providers.

PaaS stems from the markets adoption of the term Software as a Service (SaaS) with its exposing traditional applications as web services-based interfaces. PaaS is the concept of exposing platform information and capabilities thru APIs (services). In the Web2.0 marketplace APIs based on standards-based interfaces are being used by a new generation of developers to assemble new solutions quickly. While most of the discussion around services has been around solutions "in the network," the premise applies equally to services exposed by computing devices. APIs from simple Java Script calls for PHP coders to full-blown web service calls for application developers that provide information about the nature or state of platforms are what we are collectively calling PaaS. [10]

SERVICE-ORIENTED INFRASTRUCTURE FRAMEWORK

An SOI is a modular, flexible IT fabric based on standard building blocks that are highly configurable to meet rapidly evolving requirements. Looking at service-oriented solutions as a multi-layered structure, the SOI layer focuses on the orchestration and virtualization of compute, network, and storage resources. The SOI ensures resources are made available in the amount and location required by the SOA layer above it. Within the SOI abstraction, the physical details of a device are hidden by services on the platform. The device can then be managed through abstract, SLA-specific service interfaces.

SOI is optimized to handle the high-volume XML traffic associated with Web-services applications. It also uses XML as the lingua franca making possible the interoperability of management and security services built into each of the component building blocks. The SOI provides a way to manage computing resources in lockstep with application requirements both at initial deployment, and as the workload or requirements change, effectively enabling an integrated design, deployment, and management life cycle. The standardized and loosely coupled nature of SOI also reduces complexity and increases the potential for automation by enabling devices to diagnose and repair themselves, with minimal involvement from higher levels.

More specifically, the SOI layer of a service-oriented enterprise manages the following tasks:

- *Orchestration:* Managing hardware as a set of distributed and to some extent, fungible resources, shifting from a static, "one-application-per-box" paradigm to dynamic provisioning based on real-time workloads and activities. This provides the ability to realign compute, network, and storage resources as needed.
- *Asset discovery and management*: Maintaining an automatic inventory of all connected devices, always accurate and updated on a timely basis.
- *Provisioning*: Enabling "bare metal" provisioning, coordinating the configuration between server, network, and storage in a synchronous manner, making sure software gets loaded on the right physical machines, taking platforms in and out of service as required for testing, maintenance, repair or capacity expansion; remote booting a system from another system, and managing the licenses associated with software deployment.
- *Virtualization*: Making it possible to run multiple applications sharing one physical machine or storage device to increase utilization rates, or allocate multiple machines and storage devices to one application to increase performance. In other words, one-to-one dependencies between applications and platforms are removed. This capability provides unprecedented flexibility in meeting SLAs.
- *Load balancing*: Dynamically reassigning physical devices to applications to ensure adherence to specified service (performance) levels and optimal utilization of all resources as workloads change.
- *Capacity planning*: Measuring and tracking the consumption of virtual resources to be able to plan when to reserve resources for certain workloads or when new equipment needs to be brought on line.
- *Utilization metering*: Tracking the use of particular resources as designated by management policy and SLA. The metering service could be used for charge back and billing by higher-level software
- *Monitoring and problem diagnosis*: Verifying that virtual platforms are operational, detecting error conditions and network attacks, and responding by running diagnostics, de-provisioning platforms and re-provisioning affected services, or isolating network segments to prevent the spread of malware.

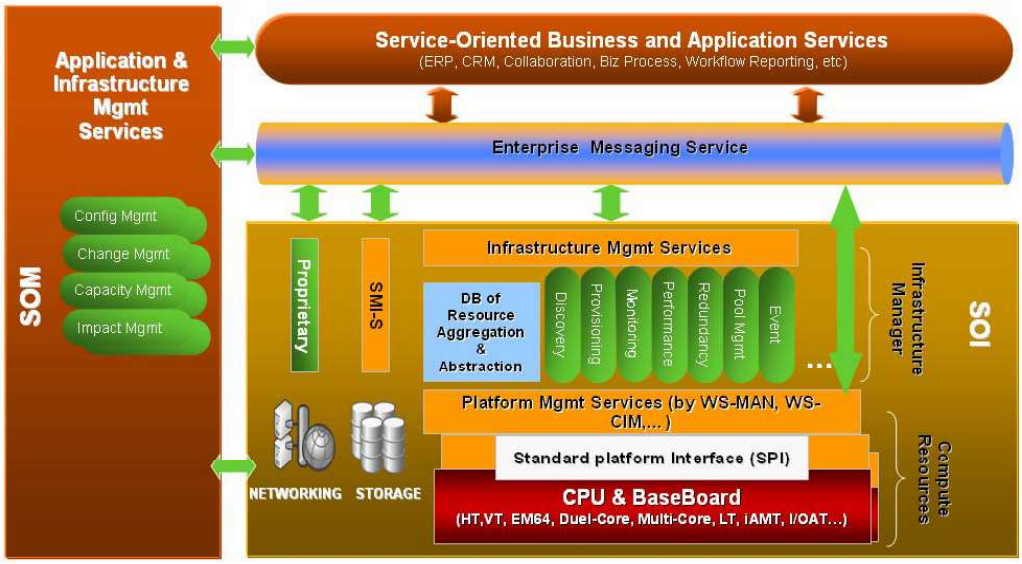

Figure 1: Service-Oriented Infrastructure framework

- *Security enforcement*: Enforcing automatic device and software load authentication; tracing identity, access, and trust mechanisms within and across corporate boundaries to provide secure services across firewalls.
- *Logical isolation and privacy enforcement*: Ensuring that a fault in a virtual platform does not propagate to another platform in the same physical machine, and that there are no data leaks across virtual platforms that could belong to different accounts.
- *IT operations processes*: Setting up generic micro IT operations as building blocks to standardize IT processes and enabling interoperability across heterogeneous system management products.

As illustrated in Figure 1, the SOI provides an abstraction service layer to SOA applications. The backend IT infrastructure management also needs to be service-oriented in order to support and manage the applications and infrastructure. Service-oriented management (SOM) is a part of the service-oriented computing infrastructure that provides management services and interfaces to IT engineers. Platform management is delivered through a Standard Platform Interface (SPI), which is based on open standards, such as WS-Management.

PLATFORM AS A SERVICE AND INTEL® AMT

Along with the service orientation paradigm shift, IT services of different forms have been discussed at [2, 6]. Under SOI, the hardware need not play the customary passive role, providing only the processing power needed for computation. A key observation is that service orientation need not be a software-only proposition; hardware can and should provide services as well. Therefore, we propose the term "platform as a service" or PaaS to denote the idea that a hardware platform can be architected with service-oriented concepts from the very beginning, having built-in service abstractions available at the bare-metal level.

PaaS can be defined as hardware- and firmware-based autonomous devices that provide standard services to interact with other entities in an SOA. PaaS devices constitute basic building blocks for SOI and interact with other orchestration and management services from SOM as well as provide seamless services to the SOA layer.

A PaaS device must have the following defining characteristics:

- *Service abstractions*: Service abstractions define basic service units, hiding hardware and firmware implementation details. This abstraction is essential in keeping the device autonomous and interoperable with other service providers or consumers. Examples of service abstraction could be device status

information, basic device controls (power on/off), provisioning, re-provisioning, systems configurations, utilization metering, etc.

- *Standard interface*: In a service-oriented environment, standard service interfaces are critical to interoperability and sharing of services across heterogeneous platforms. This is especially true with PaaS as there are many different SOA applications running on top of it, and hardware platforms should be designed for general IT infrastructures. It is not practical to expose PaaS services through proprietary interfaces, as it will limit service adoption and defeat the purpose of service orientation. Therefore, PaaS has to support open service standards like Web Services–the same standards that software and applications can understand. In this way, other SOI, SOA, and SOM components can interact with PaaS devices in the same way that they interact with software components.
- *Change tolerant*: Hardware is often seen as very efficient and reliable, but not very flexible and adaptive to change. With PaaS, the concept of hardware could change fundamentally. Hardware becomes flexible and adaptive to provide basic services upon installation with or without software. Platform services could also aggregate at group levels across multiple platforms and interact with other services in the environment as part of the overall SOI.
- *Policy driven*: As part of an overall solution, a PaaS device has to understand the response to policies that direct IT environments. PaaS typically has a built-in policy engine to interpret policies down-loaded from the IT environment. At a minimum it will be able to respond to instructions and directives from the upper layers of IT management services defined in terms of orchestration, SLAs, and Quality of Service (QoS).

PaaS will bring new opportunities and challenges for hardware designers and vendors. Provisioning hardware at higher levels of abstraction and enhanced security will require significant rethinking, especially when these functions are available at the bare metal state. Nonetheless, PaaS will beneficial overall in accelerating the adoption of SOA and it will increase the productivity of IT professionals facilitating business through building end-to-end solutions.

Intel AMT®–a PaaS Example

Intel announced a new platform management technology in 2005 to reduce the complexity and cost of platform management for businesses–Intel AMT.

We illustrate the PaaS concept by pointing out some of the features in Intel AMT, showing that PaaS is technically feasible and some features are available today. The following table summarizes key features and services provided by Intel AMT. They are built-in at the platform level as combinations of hardware and firmware.

Table 1: Key features and services of Intel AMT

	Key Intel® AMT Features
Discover	OS-independent, accurate platform, software, and hardware inventories are necessary for regulatory compliance as well as for closely managing maintenance contracts and software licenses.
Heal	Built-in proactive alerting and remote recovery capabilities reduce the number of desk-side visits by allowing IT to recover control of failed PCs, diagnose the problem, and potentially fix the problem remotely.
Protect	Allows security patching of systems regardless of system state (powered on or off).
Standard Interfaces	Based on Web services standards ready to integrate with heterogeneous management consoles.

With the OS up and running, Intel AMT can operate in in-band (IB) mode and out-of-band (OOB) mode. In IB mode, Intel AMT interacts with the OS and management console to perform requested platform management functions. When the OS is not available, either because it has not been provisioned (bare metal state) or because it has crashed, Intel AMT can operate in OOB mode. It continues to provide basic platform management functions and is accessible through a management console.

Intel AMT is an example of PaaS available in products today—it is autonomous, has clearly defined platform management services that are always available, and exposes those services through standard interfaces. The following section describes a PoC project conducted by Intel IT based on Intel AMT and WS-Management that demonstrates how these work in an autonomic fashion in a real IT environment. For further details of WS-Management and other WS* standards, please refer to [9] in this issue of the *Intel Technology Journal*.

INTEL® IT POC ARCHITECTURE AND KEY RESULTS

Like many other IT organizations, Intel IT is motivated to apply autonomic computing technologies to reduce operational costs and increase IT agility. The vision is that

computing capabilities are autonomous and well-integrated, and that they can maintain themselves and control their own resources to deliver IT services in a consistent and continuous way. The autonomic enterprise vision is part of an agile IT vision, allowing an IT department to continuously adapt in a dynamic and automatic way, in response to changing business needs.

Creating a Common Enterprise "Language"

A source of complexity in the enterprise is the diversity of communication protocols, usually proprietary, that leads to vexing integration efforts. A common language is needed so that enterprise components and systems can be discovered and freely communicate with each other. The PoC demonstrates the use of the WS-Management protocol stack. This is employed in front of each one of the stack layers, components, and systems in the enterprise, in a reusable fashion. It enables a standard, common and unified messaging layer that drives Managed Object (MO) discovery and provisioning. This enables a holistic approach to IT infrastructure management. Using a WS-Management protocol stack we were able to create a common communication approach for the entire enterprise manageability framework.

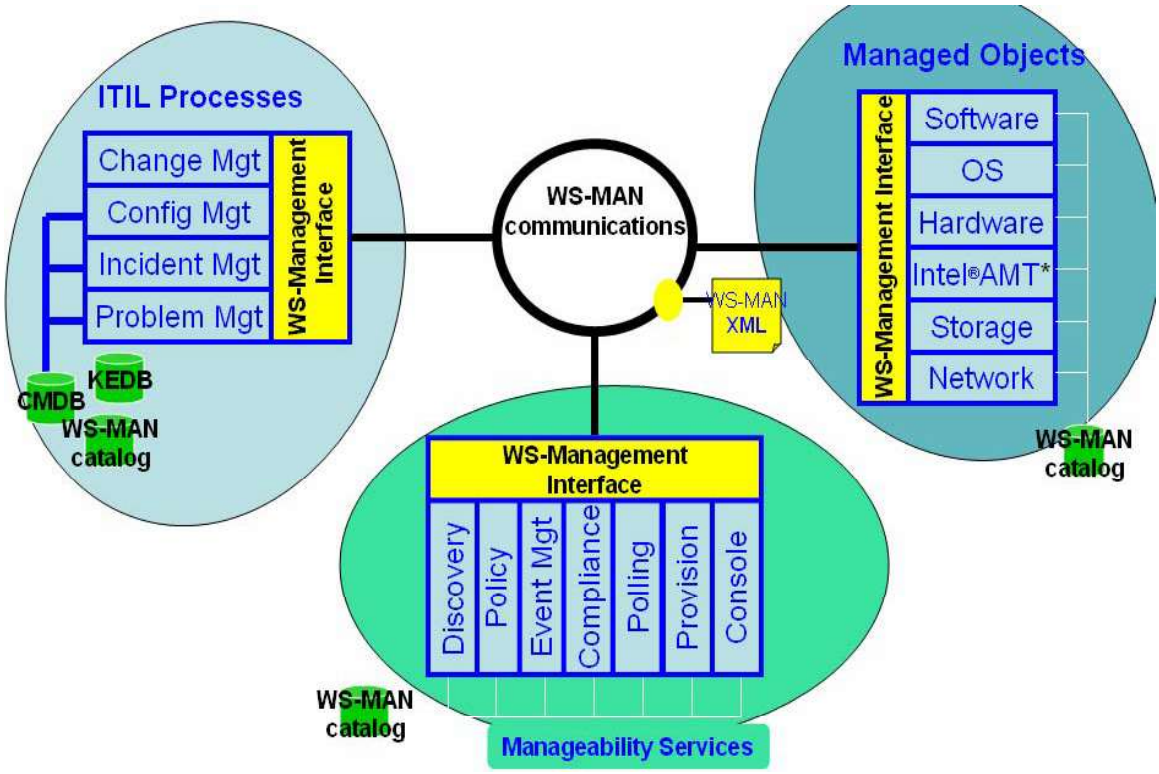

Figure 2: WS-Management stack in front of each component creates a common "language"

The common communication framework simplifies manageability integration with the capability to automatically discover and control resources. This capability advances the notion of the autonomic enterprise, fulfilling the vision of agile IT.

Automated Discovery

A key enabler of the evolution of the autonomic enterprise is the ability to dynamically discover all computing resources and immediately control them.

Dynamic, real-time discovery means the following:

1. The ability to detect when an MO comes onto the network or changes its state.
2. The ability to query the MO for its metadata.
3. Registration of the new MO instance in the Configuration Management Database (CMDB).

Immediate MO control is the ability to control the behavior of the object through business rules or policies. These policies are automatically provisioned to the MO when it comes online or changes its state.

Conceptually, the dynamic, real-time MO discovery process has three main steps:

1. As the MO comes online in the network, it announces itself by sending a thin message.

2. The manager gets the announcement and queries the MO end-points (through the manageability interface) for the required object Configuration Item (CI) information.
3. The manager then registers the MO in the Configuration Management Database (CMDB).

When an MO joins the network, or has a metadata change, it will interact with the backend IT management system (the Manager) in the following way:

1. Send an announcement: send a thin "Hello" message to a Discovery Proxy (DP).
2. Register the change: the DP processes the announcements and forwards them to the Manager.
3. The Manager then queries the MO for more information as needed for enterprise management.

The MO Catalog (following the WS-Management Catalog specification) contains the information needed by the Manager to pull all required data. These may include events that are exposed by the MO, available methods and how they can be used, and MO properties and attributes.

The Manager then creates a new CI record in the CMDB for the new instance of the object type (or updates the record if it already exists, for example, for a status change from "offline" to "online" or vice versa).

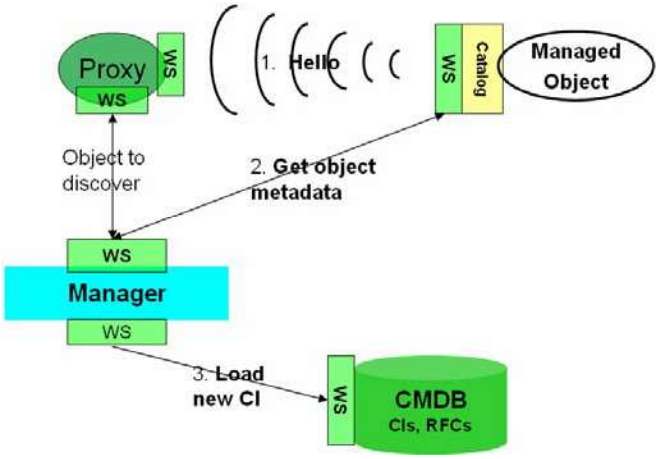

Figure 3: Managed-Object Discovery concept diagram

The announcement process can be implemented through multicast or unicast. DP could be built-in as part of the Manager implementation.

Dynamic, automatic, and real-time MO discovery is an essential capability in the evolution toward a fully autonomic framework. Discovery is not limited to devices only, but must address all layers of the stack, including hardware, the Virtual Machine Monitor (VMM), operating systems, and applications. Probing for a change (pull model) doesn't answer all enterprise management needs because current and future enterprise objects are highly dynamic. A combination of pull model and push model is the most likely solution to the autonomic computing needs. The autonomic computing facilities will reconfigure themselves based on business need, load, fault management, and other factors in order to optimize service availability and throughput. Some key results of the discovery process are an up-to-date enterprise directory (CMDB, another key enabler of the evolution of an autonomic enterprise) and the functionality it needs.

Standards plays a fundamental role in the discovery process. The WS-Management Catalog is the key enabler for MO discovery. It is the entire set of metadata available from a WS-Management agent for a given Resource. WS-Transfer, which is part of WS-Management specification suite, is then used to pull the information out of the catalog. WS-Management is a standard communications protocol for communication between the different components.

Policy Provisioning

Through the evolution of the autonomic enterprise, IT departments look to run their business based on a logical model that fits best with their business processes. This logical model is then mapped to physical business rules and policies. Those physical policies are provisioned to the corresponding physical MOs.

In this PoC, physical policies are stored in the CMDB, associated with the corresponding type of MO. Once a new instance of an MO type is discovered, its attributes (type, location, etc.) are matched with selection criteria in the CMDB to automatically trigger policy provisioning. A compliance tool, once triggered, provisions the policies to the newly discovered MO through the standard WS-Management protocol.

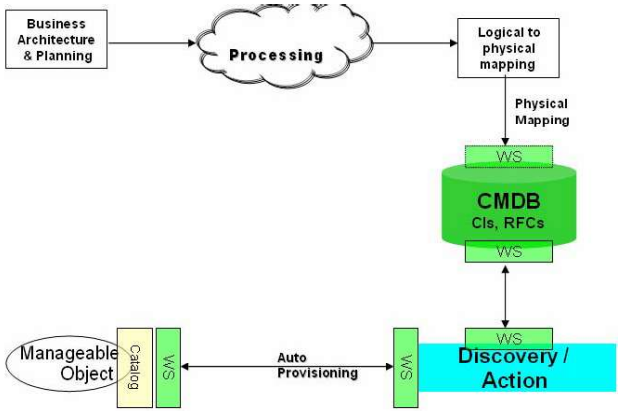

Figure 4: IT business model representation and physical mapping (policies)

Federated Event Processing

Enterprise event management can cross system, domain, and company boundaries. It requires automated cooperation between multiple enterprise systems, including business layer systems, to make it part of an automated response system. The systems involved in processing an event include MOs, Management tool-sets, ITIL (Information Technology Infrastructure Library) functions, and other systems like the Known Error Database (KEDB). The events that MOs expose are described in the MO Catalog (WS-Management Catalog), which can be queried dynamically when needed.

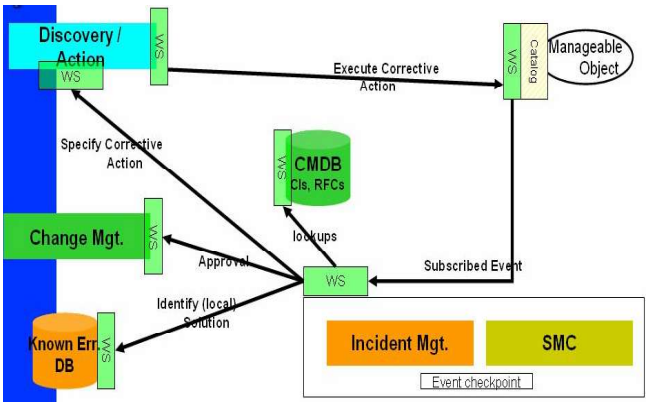

Figure 5: Federated event processing

When a new instance of an object type is discovered, the event management system is triggered. It subscribes to those events at the MO end-points that are of business interest. The subscription process can be done either directly by the event management system, or as part of policy provisioning when a new object instance is discovered. WS-Eventing (part of the WS-Management suite) provides the mechanism and protocol for Event Subscription.

When a failure or fault event occurs in an MO, the PoC takes a holistic approach, involving the technical and IT business layers. For example, an event is sent to the event management system using WS-Eventing; the event management system queries CMDB for additional information needed on the MO, and then process the event internally. The event management system queries the KEDB, using WS-Transfer, for a potential known resolution for the problem. The KEDB replies with the resolution action to be executed (also using WS-Transfer).

In cases where the MO cannot make a decision on its own, the event management system sends a request to the change management system (the ITIL function at the business layer) for further actions to correct the failure.

The change management system replies with approval (both using WS-Transfer), and a request is sent to the compliance system (also using WS-Transfer) to execute the corrective action.

The change authorization may include information such as the window for maintenance and the expected time to complete. The compliance system then executes the corrective action using WS-Transfer. All communication in this scenario is implemented using WS-Management standard protocol stack to enable the holistic approach required to automate event management.

Role of Intel AMT

During the PoC we used an Intel AMT simulator as part of the device's manageability stack. It had a fundamental role as the root of the discovery and provisioning processes.

When the device is first plugged into the network, the simulator is automatically and immediately discovered through the process described above. As part of this discovery the simulator sends its identification number as a unique identity key representing the host platform. This key serves as the root of the discovery and provisioning process for the entire stack, and it announces the existence of the new device as a computing resource.

As part of the PoC we were able to automatically discover and provision a new server platform out-of-the-box by having Intel AMT initiate the discovery process for the server OOB.

In response to the discovery of a new server, policies kept in the CMDB directed the bare-metal OS provisioning to be performed OOB via Intel AMT.

Once the OS was brought online it was discovered and provisioned, and the processes of discovery and provisioning were repeated for the entire stack (now IB, or via a WS-Management interface to the OS).

Using this approach we were able to take a new hardware platform out of the box and automatically add it to an existing IT service for added computing power, with minimum delay and manual intervention.

KEY RESULTS AND CHALLENGES

The PoC demonstrated a common and standard communications protocol and a standard and discoverable interface type. These permitted us to abstract the specific implementations of enterprise components, systems, utilities, and IT business processes.

Each of these elements was able to communicate freely and discover others using a common protocol. We were able to see true enterprise-level autonomic behavior in which the low-level objects and technologies behave in a way that fit with the business needs to meet the SLA; we

also saw the business functions involved making the right decisions.

In addition, we were able to reuse the WS-Management protocol stack and integrate it with existing manageability processes and toolsets. The effort required to do this was minimal–more time was spent increasing the flexibility of the stack than in creating the functionality.

We successfully demonstrated the use of the HTTP/SOAP protocol to automatically self-announce a device as it is introduced to the network. We used the same methodology to demonstrate self-announcement of adds or changes to software on the device. We integrated self-announcement with automatic end-point interrogation over WS-Management; and we successfully used WS-Management to automatically populate a CMDB with the data gathered from the interrogation. We designed a means to define a group of behaviors under the umbrella of a policy and to define a way to identify system attributes to trigger the application of these policies.

Using WS-Management, we communicated the need to apply the behaviors to a compliance system. We used WS-Eventing as a trigger to start an event management "transaction." We made event management automatically intersect with ITIL processes and communicate the need to apply a known solution to a compliance system over the WS-Management protocol.

We effectively demonstrated a well-behaved system self-management. We were able to automatically discover and provision new devices, just out-of-the-box, by having the Intel AMT simulator initiate the discovery process OOB, then through policies provisioned the entire stack, resulting in the automatic addition of a new piece of hardware to an existing IT service to provide more computing power. We were also able to utilize WS-Catalog to represent and expose the MO metadata in a standard way to drive standard-based and free communications between components in a loosely coupled manner.

Some of the main challenges moving forward include the industry adoption for the WS-Management protocol suite as open standard: there is a need for all layers in the stack, from the hardware to the applications, to "speak" WS-Management and implement the WS-Management standard and discoverable interfaces. IT departments need solutions that support WS-Management off the shelf.

Another key challenge concerns industry support for the complete MO discovery mechanism through standards, and for making it part of WS-Management. MO self-announcement is a key missing ingredient; most other parts of the discovery process can already be enabled via WS-Management.

Intel AMT can enable the root of the device discovery process and support a hardware layer in all platforms for the autonomic enterprise computing demonstrated in this paper. There is a need to promote a standardized data model in the CMDB for MOs with their attributes and associations, as well as for policies and their association with MO types. Standardization of message content is another key challenge for moving forward toward semantics in the enterprise.

SUMMARY

Service orientation at the infrastructure level is being adopted and is expected to become the foundation for higher-level SOA applications to execute. It is clear that well-defined services directly provided by the hardware platform bring advantages of availability, reliability, and OS independence. By directly exposing standard programming interfaces to developers, PaaS will help enable the use of autonomic computing in IT infrastructure.

Our POC demonstrates that transition to a service-oriented autonomic enterprise takes collaborative efforts of the ecosystem from hardware and software vendors, system integrators, and IT professionals. This transition will neither come naturally as a result of any single company. Standards definitions and interoperable implementations are key to this ordeal.

ACKNOWLEDGMENTS

Many thanks to Intel IT, SMG, DEG, SSG and many friends in other groups at Intel for the advice and support before and during the development of SOI concepts and the POC.

REFERENCES

[1] J. He, M. Chang, E, Castro-Leon "Evolution of Intel's e-Business Data Center Toward Service-Oriented Infrastructure," *IEEE International Conference of e-Business Engineering*, Beijing, October 2005.

[2] N-able Technologies whitepaper "Evolution of IT as a Service," April 2006.

[3] Intel whitepaper "Intel: Service-Oriented Enterprise, the Technology Path to Business Transformation," June 2005.

[4] W.T. Tsai, Bingnan Xiao, Raymond A. Paul, Yinong Chen, "Consumer-Centric Service-Oriented Architecture: A New Approach," in *Proceedings of IEEE 2006 International Workshop on Collaborative Computing, Integration, and Assurance* (WCCIA), April 2006, pp. 175–180.

[5] W.T. Tsai, "Service-Oriented System Engineering: A New Paradigm," *IEEE International Workshop on Service-Oriented System Engineering* (SOSE), Beijing, October 2005, pp. 3–8.

[6] Dan Neel, "Hardware at Your Service," *CRN Magazine, Issue 1199*, June 5, 2006, pp. 16–20.

[7] C. Koch, "The Truth About SOA," *CIO Magazine*, Volume 19, No. 17, pp. 49–60.

[8] G. Yohanan, J. Hahn-Steichen, J. Hobbs, J. He, "Orchestrating Computing Objects using Web-Services technologies–Intel IT Proof of Concept (POC)," *International Conference of Autonomic Computing*, June 12–16, 2006.

[9] Milan Milenkovic, Vijay Tewari, "Standards for Autonomic Computing," *Intel Technology Journal*, Volume 10, Issue 4 2006.

[10] Andy Mulholland, Chris S. Thomas, Paul Kurchina with Dan Woods, "Mashup Corporation: The End of Business as Usual," *Evolved Media Network, 200*, pp. 61-64.

AUTHORS' BIOGRAPHIES

Enrique Castro-Leon is an enterprise architect and technology strategist for Intel Solution Services at Intel working in OS design and architecture, software engineering, high-performance computing, platform definition, and business development. He holds Ph.D. and M.S. degrees in Electrical Engineering and Computer Science from Purdue University. His e-mail is enrique.g.castro-leon at intel.com.

Mark Chang is a principal strategist in the Intel Customer Solution Group specializing in Service-Oriented Enterprise business and technology strategies worldwide. He holds an MS degree from the University of Texas at Austin. His e-mail is mark.chang at intel.com.

Jay Hahn-Steichen is a forward engineer with Intel IT Integration Platform Services group. He holds an MS degree from the University of California at Berkeley. His e-mail is jay.hahn-steichen at intel.com.

Jackson He is a lead architect in Intel's Digital Enterprise Group, specializing in manageability usages and enterprise solutions. He holds Ph.D. and MBA degrees from the University of Hawaii. His e-mail address is jackson.he at intel.com.

Jim Hobbs is a senior architect in the Strategy, Architecture, and Innovation division of Intel Information Technology. He has a degree in Information and Computer Science from the University of California, Irvine. His e-mail is hobbs at intel.com.

Guy Yohanan is an enterprise architect in the Innovation group of Information Technology at Intel. His primary interest is manageability and mobility technology initiatives and path finding. He holds a B.A. degree in Computer Science from the University in Tel-Aviv Israel. His e-mail is guy.yohanan at intel.com.

† Intel® Active Management Technology requires the computer to have additional hardware and software, connection with a power source, and a network connection. Check with your PC manufacturer for details.

THIS PAGE INTENTIONALLY LEFT BLANK

Standards for Autonomic Computing

Vijay Tewari, Corporate Technology Group, Intel Corporation
Milan Milenkovic, Corporate Technology Group, Intel Corporation

Index words: autonomic computing, Web services, management standards, WS-Management, platform manageability, service-oriented architecture, self-managed infrastructure

ABSTRACT

Modern distributed computing systems are expected to be able to span a large number of heterogeneous interconnected devices while providing numerous and increasingly complex services to audiences on a global scale. Such systems are increasingly complex to deploy and operate and are stretching management techniques to a breaking point. Autonomic computing is emerging as an architectural philosophy and design approach that promises to cope with complexity and scale up to the needs of today's distributed systems. Its fundamental tenet is to increase the intelligence of individual components so that they become "self-managing," i.e., actively monitoring their state and taking corrective actions in accordance with overall system-management objectives.

In this paper we describe the standards for autonomic computing necessary for implementing an autonomic computing vision in a heterogeneous world where components and platforms are supplied by different vendors. After providing an overview of existing and emerging standards, we focus on Web-services-based standards for interoperability among autonomic managers and components of distributed systems. We cover specifically the scope and motivation for WS-Management, Web Services Distributed Management (WSDM) and the ongoing effort to converge the two into a common industry standard that will create a strong foundation for autonomic systems and automated management in general.

INTRODUCTION

Computing systems have evolved from single machines in large machine rooms to millions of interconnected devices whose interactions create complex webs built on increasingly complex architectures consisting of multitudes of powerful devices running tens of millions of lines of code. The increase in size and complexity of interconnected heterogeneous systems has led to a similar increase in the cost and complexity of configuring and operating such systems. Complexity is not just a function of the number of devices, but also of diversity in their types and capabilities. Lastly, increasing globalization tends to lead to geographic dispersion of some large systems, which adds another dimension to this already complex scenario.

IBM highlighted this growing complexity and coined the term autonomic computing [1] to describe an approach for managing complexity that relies on designing and building computing systems capable of "managing themselves." Basically rooted in control theory, autonomic computing is a design principle and an architectural philosophy that advocates increased autonomy of computing nodes and self-management at the edge of the network.

It is, however, a challenge to build closed-loop managed systems in an open architecture where a system is composed of multiple components (hardware and software) from different vendors. This is because components need to interoperate seamlessly and have a shared common understanding and representation of policies, managed objects, and system events. At each level of the hierarchical management loop, components will run a closed autonomic loop with well-defined interfaces to communicate with their peers and with other levels of management hierarchy. In this paper, we explore the various standards that facilitate such an interaction, including WS-Management and the emerging model-based approaches.

This paper is organized as follows. Firstly, we briefly outline the conceptual architecture for autonomic computing. We then describe the multiple standards for enabling autonomic computing. Finally, we provide additional details on external interfaces for autonomic computing elements as they enable interoperability between autonomic managers in heterogeneous systems.

CONCEPTUAL ARCHITECTURE FOR MULTI-LEVEL AUTONOMIC COMPUTING

In this section, we outline the architectural blueprint for building autonomic systems. In subsequent sections, using the conceptual architecture, we point out the touch points where standards play a critical role and highlight existing standards as well as point out gaps. Figure 1 depicts [2] the details of a single autonomic manager. The components and functions of a single autonomic manager often referred to as the "MAPE loop" for Monitor, Analyze, Plan, and Execute, supplemented by a Knowledge Base are briefly described below.

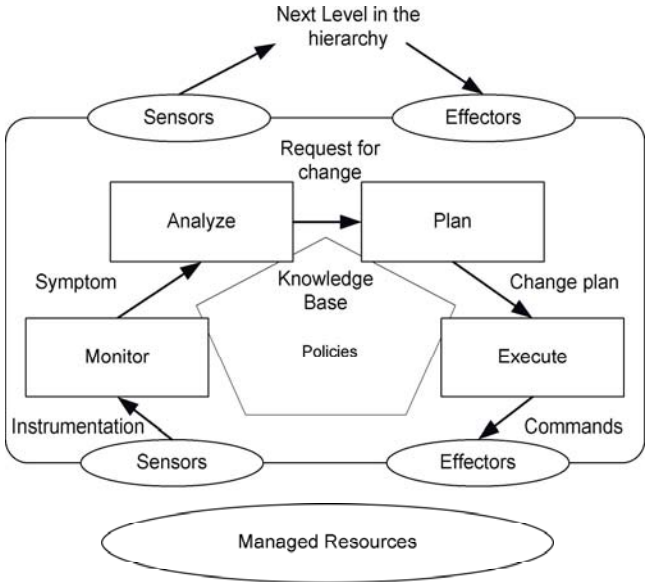

Figure 1: Single-node autonomic manager

Monitor

An autonomic manager monitors instrumentation data from multiple sensors in a system. The sensors "sense" various aspects of the state of the monitored computing system. This can include aspects of the hardware instrumentation (such as temperatures at various points within the hardware platform), ambient information (such as the environmental temperature, physical intrusion of the device), and aspects of the software components (such as various performance-monitoring counters of the operating system and specific counters for application monitoring). Data can be acquired from sensors by polling at specified intervals or can be collected asynchronously when specified thresholds are exceeded.

Analyze

This component of the autonomic manager contains the intelligence required to interpret and correlate the above mentioned instrumentation data. This component usually has the ability to consult historical data and to compare them with current state to detect significant changes. Optionally, in conjunction with the knowledge base component, it may predict the likely future states of the system and use them, as appropriate, for performance adjustment or to work around anticipated faults. System topology, data, and control flows may be captured in formal models to facilitate this analysis. For example, this component may be able to examine patterns of service requests presented to a Web farm and, based on known patterns, conclude that it is likely to result in a spike in such requests in the near future.

Plan

Once an analysis report of the situation is completed, the planning component can define a series of control actions that should bring the system to a normal operating range. Note that in devising these actions; the planning component is guided by the applicable management policies that are in effect. These policies may be defined

by the operator or by the next level in the autonomic hierarchy. For example, in the situation of a likely overload of the Web farm, this component may detect that the maximum response time for the client, specified by the Service-Level Agreement (SLA) in effect, will be violated by the predicted surge in load. Using this information, the Plan component decides that the right course of action is to supplement the Web farm with additional front-end Web servers to avoid this SLA violation.

Execute

This component receives the series of action steps from the planning component, and puts the plan into action. It activates appropriate control points, or effectors, on the managed platform following the proper sequence and timing. For example, given the solution of supplementing the front-end Web servers with additional resources, this component could activate additional physical or virtual machines, provision them as Web servers, and add them to the Web farm.

Knowledge Base

This serves as a repository of knowledge, such as historical data and policies, which can be utilized by the other components in their operation. Knowledge can be obtained from multiple sources: from peer autonomic managers, a management console and operator, or can be obtained using machine-learning techniques from prior observations of system states and corrective changes that were found to be effective.

STANDARDS FOR AUTONOMIC COMPUTING

Computing systems of today are highly diverse, and this heterogeneous nature necessitates the interaction of autonomic managers with resources from multiple vendors. In order to complete the entire self-managing loop, it is imperative that components within a single autonomic manager be able to interact with each other. In a larger system with a hierarchical architecture, multiple autonomic managers must be able to interact with each other. The role of standards is invaluable in facilitating both these interactions.

Another related development in the computing industry is convergence on the concept of building applications and services using the principles of service orientation, often referred to as Service-Oriented Architecture (SOA). The use of Web services as a technology to build SOA systems has gained significant following in the industry. The leading motivator for the use of this technology is the ability to interoperate amongst implementations from multiple vendors. Given the significant investment, implementations, tools, and broad industry support for SOA and autonomic computing, it is both pragmatic and desirable to use the same or interoperable Web-service-based standards for both domains.

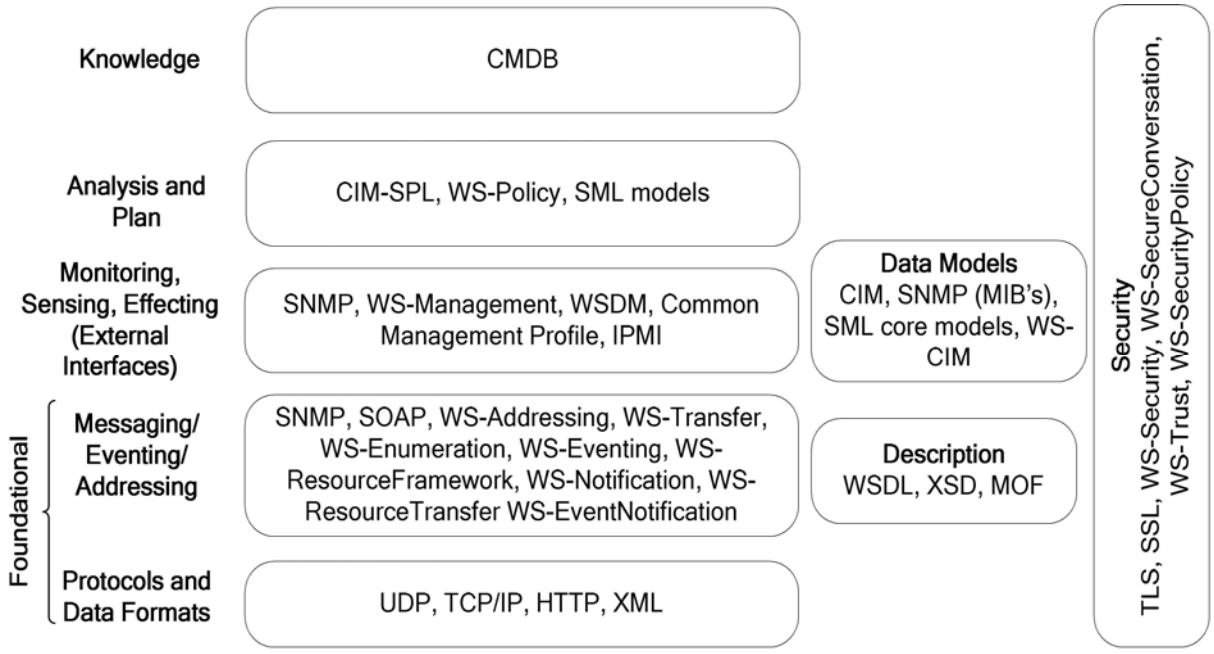

Figure 2: Standards for autonomic computing

Starting with a single autonomic manager, we now examine the various touch points where standards should be defined, point out existing standards in that domain, and highlight gaps where they exist. Figure 2 depicts a standards stack for enabling autonomic computing spanning hardware management, OS/Application management, services, and business process management. In the sections below a representative example of standards from each layer is briefly described. Some descriptions pertain to more than one layer in the stack and have been included only once.

Protocols and Data Formats

All message exchanges between autonomic elements need to be built upon a common set of protocols and data formats. This section briefly describes the common protocols and data formats.

- *UDP, TCP/IP:* IP (v4 and v6) is the most popular network layer protocol. Layered on IP, are the connection-oriented TCP and connectionless UDP protocols.
- *Hypertext Transfer Protocol, HTTP [28]:* The most popular messaging protocol over TCP/IP is the use of HTTP (standardized by the W3C), which is a request/response protocol and which is the foundation of the World Wide Web.
- *Extensible Markup Language, XML [31]:* XML is a simple text format used for the exchange of information and its use has facilitated the sharing of data across disparate systems in a machine-readable manner.

Messaging Eventing and Addressing

This section briefly describes the basic messaging and addressing standards.

- *SOAP [32]:* SOAP is a lightweight message exchange protocol based on XML defined by the W3C for exchanging structured information in a decentralized distributed environment.
- *WS-Addressing [33]:* WS-Addressing is a W3C recommendation and defines a transport neutral mechanism to address Web services and messages.
- Additional details on standards in Figure 2 pertaining to Messaging, Eventing and Addressing are contained in the section "External Interfaces for Autonomic Computing Elements."

Resource Sensing, Monitoring and Effecting (External Interfaces)

At the lowest level of abstraction, is the ability to monitor and control resources using the appropriate external interfaces. Listed below are various standards that fall into this domain

- *Simple Network Management Protocol (SNMP):* This is a network management standard from the Internet Engineering Task Force (IETF) [3]. It consists of a management data-model and a network protocol for querying the status of managed resources, executing remote commands and an eventing mechanism for dealing with asynchronous communication. It is widely adopted and includes an extensible management data model called a Management Information Base (MIB).
- *WS-Management/WSDM:* WS-Management is a Web services protocol standardized by the DMTF for managing devices, services, and systems. It supports retrieval of object properties, enumeration of objects, and a publish/subscribe communication paradigm (eventing). Web Services Distributed Management (WSDM) [7] is a management standard from the Organization for the Advancement of Structured Information Standards (OASIS) which consists of two components, Management Using Web Services (MUWS) and Management of Web Services (MOWS). Additional details are given later in this paper.
- *Intelligent Platform Management Interface (IPMI):* IPMI [11] defines a message-based interface to intelligent platform hardware and a set of standardized records for describing and accessing platform managed devices, such as thermal sensors and fans. Keeping in mind the use of a common modeling schema, the IPMI developers have published guidelines for mapping IPMI to CIM [12].
- *Software Monitoring:* Monitoring of software components remains a fragmented space, in large part due to the diverse programming languages and runtimes. It is expected that with the advent of a common management protocol and a data model (such as CIM) this fragmentation will be reduced. There is, however, good support for retrieving software instrumentation (although not the same across different platforms) using technologies such as Java Management Extensions (JMX) [9], System.Management namespace in .Net [10].

DESCRIPTION

In order to facilitate runtime binding for structured information exchange between autonomic elements, each interface and model should be described in a machine-readable manner. This section highlights the two prevalent standards that provide this capability

- *Web Services Description Language, WSDL [34]:* WSDL (developed by the W3C) is an XML-based description language for defining the message formats for interacting with an interface using Web services. Although not a recommendation, WSDL 1.1 is widely

deployed today. Version 2.0 is expected to be a W3C recommendation.

- *XML Schema, XSD [35]:* XML schema is an XML-based language for documenting in a machine-readable manner the structure of an XML document. In XML-based interactions for the management domain, this is used to document the structure of the data models such that formal validation and processing of the data can be carried out.

Data Models

IT systems need an abstract mechanism to represent data pertaining to all aspects of the system. In order for autonomic managers to monitor and control elements, they require a well-defined and structured representation of the objects. This section highlights some of the standards pertaining to IT models.

- *Common Information Model (CIM):* Developed by Distributed Management Task Force (DMTF), the most significant part of the CIM standard is the object-oriented CIM schema [4], which is a data model that describes managed objects and elements in an IT environment and their properties and supported operations including modeling of the relationships between them. Use of the CIM schema allows information to be shared among autonomic managers. The CIM schema itself is documented in a language called Managed Object Format (MOF).
- *Web Services CIM, WS-CIM:* This DMTF standard [5] defines a translation from the MOF format for CIM into XML schema. This allows CIM model definitions to be transported over the Web-service management protocols, such as WS-Management [6].

Analysis and Planning

In the autonomic manager architecture, the analysis and planning functions provide the necessary intelligence to facilitate decision making down to lowest possible level in the hierarchical autonomic management model. In this section we outline the standards that span this decision making.

- *WS-Policy:* This standard [14] provides a general-purpose mechanism for describing and communicating policies relating to a Web service. This emerging standard gains significant importance in light of the fact that resource monitoring and control are moving to be exposed as Web service endpoints.
- *CIM Simplified Policy Language (CIM-SPL):* This is a CIM-compliant policy specification language from the DMTF.
- *Service Modeling Language (SML):* This is a recently announced modeling language [15] being developed by multiple companies. Although not strictly a policy specification language, SML defines a modeling language based on XML schema that is intended to allow formal modeling of IT systems, including constraints and declarative policies, by using Schematron language [29]. It is expected that such operational models will allow autonomic managers to monitor and enforce service deployment and operational constraints specified by system designers and IT operators in the various stages of the service life cycle, including design, deployment, and operation.
- *Business Process Execution Language (BPEL):* BPEL [27] is a language that models business processes as Web services allowing businesses to implement a service-oriented approach to business processes.

Although there are emerging standards (as above) to enable analysis and planning for autonomic managers, additional standards need to be defined for hierarchical autonomic policies for expressing operator constraints and partitioning the given policy to all managers in the hierarchy.

Knowledge

The Configuration Management Database (CMDB), a fundamental component of the Information Technology Infrastructure Library (ITIL) [30] framework is an emerging standard for the repository of information of an IT system. In addition, standards will need to be defined for interacting with the CMDB (data and interchange).

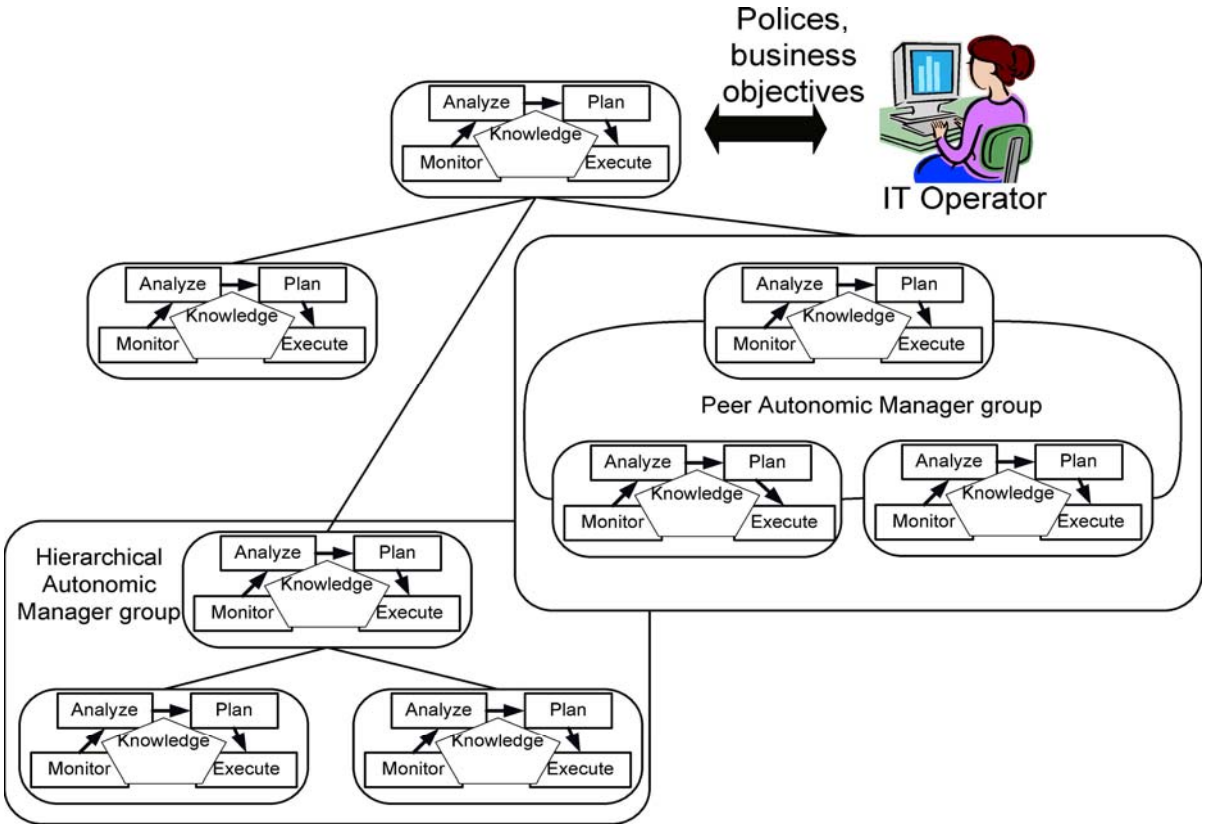

Figure 3: Groups of autonomic managers in a complex system

Security Standards

Given the nature of the autonomic system, robust security architecture is required to prevent a compromised autonomic manager from impairing the operations of the system. Some of the standards that provide the security infrastructure for systems that use Web services are highlighted in this section.

- *WS-Security:* Given the strong reliance of SOAP-based messaging for interaction between autonomic elements, a comprehensive standard that provides confidentiality, integrity, and authentication for SOAP-based messages is crucial. WS-Security [16] from OASIS meets these requirements.
- *WS-Trust:* This specification [17] defines extensions to WS-Security to provide a framework for requesting and issuing security tokens, and to broker trust relationships.
- *WS-SecureConversation:* This specification [18] defines extensions for WS-Security and WS-Trust for establishing and sharing security contexts.
- *WS-SecurityPolicy:* This specification [19] defines the policy assertions for use with WS-Policy that relate to WS-Trust, WS-Security, and WS-SecureConversation.
- *Transport Layer Security (TLS)/Secure Socket Layer (SSL):* The adoption of Web-services-based security standards will take place gradually over time. In the meantime, TLS/SSL is commonly used to provide message integrity and confidentiality over HTTP.
- *Security Assertion Markup Language (SAML):* SAML, developed by OASIS [20], is an XML-based framework for communicating user authentication, entitlement, and attribute information.
- *WS-Federation:* Autonomic managers may need to interact across security domains. The WS-Federation specification allows security information to be shared across security realms.

In this section, we have highlighted some of the important standards used for enabling autonomic behavior. A critical aspect of the interaction is the external interfaces of each element/manager. Details on protocols and standards for external interfaces are given in the next section.

EXTERNAL INTERFACES FOR AUTONOMIC COMPUTING ELEMENTS

A single-node autonomic manager is responsible for the control of a single managed entity. A typical enterprise system is comprised of a large number of interconnected components. In an autonomic system, that translates to a large number of autonomic managers that need to interact with each other to exchange sensor data, policies, and perhaps plan a collective course of action to meet high-level business objectives. The topology of such systems may be hierarchical, peer-to-peer, or a combination of the two. Figure 3 depicts a hybrid system where groups of autonomic managers are organized in a hierarchical manner, with some elements of hierarchy being hierarchical groups of autonomic managers themselves, while constituents of other groups are connected to their peers.

In a hierarchical organization, each group has a "root" node that acts as the next level in the hierarchy. It typically aggregates instrumentation and events from members of its group and translates group-level policies into individual node policies or action plans. The "root" of the tree interfaces with the IT administrator who conveys policies and business objectives for the system. The overall goals and objectives are translated by the root-level autonomic manager into policies and execution plans for autonomic managers that are lower in the hierarchy.

Given the potentially large number and heterogeneity of autonomic managers, construction of large autonomic systems depends on the presence of well-defined standard interfaces for their interaction. Web services have proven their usefulness for construction of heterogeneous distributed computing systems since they are platform independent and support run-time discovery, introspection, description, and binding of components. Moreover, Web services are supported by leading software vendors and open-source projects resulting in built-in run-time support and availability of tools for both Windows* and Linux* environments.

Due to historical reasons, two major initiatives evolved to define mechanisms and standards for monitoring, controlling, and eventing using Web services for systems management, WS-Management, and WSDM. WS-Management [6] is a specification standardized by the Distributed Management Task Force (DMTF). It is built on three fundamental specifications: WS-Transfer [21], WS-Eventing [22], and WS-Enumeration [23]. WSDM [7] is a specification standardized by OASIS, which builds upon the Web Services Resource Framework, and WS-Notification. In the next few paragraphs, we briefly describe the scope and functionality covered by these specifications and outline a path to unifying these two approaches as identified in a white paper [8].

WS-Management utilizes the services of WS-Transfer for performing basic management operations on managed entities: **C**reate, **R**ead, **U**pdate and **D**elete (CRUD) by using WS-Transfer, which provides a generic set of CRUD operations for Web services. In systems management, numerous operations need to enumerate large lists of objects. For example, a remote manager may desire to list all the running processes on a machine in order to determine if there are any unauthorized processes. WS-Enumeration, which provides a set of generic operations to enumerate lists, can be used to iterate all instances of the CIM_Process class to obtain this information in an efficient manner.

WS-Management relies on WS-Eventing to provide the basic publish/subscribe/deliver semantics. A common interaction pattern in systems management is event-based interaction using a publish/subscribe mechanism. In this approach, an event source or its proxy makes known or publishes the type of events that it can generate. Entities indicate their interest in being notified when specific events occur by subscribing to them. The act of subscription consists of storing a subscriber's identity and notification address in the event subscription list maintained by the publisher or a suitable proxy. When events occur in the course of a monitored object's operation, the publisher notifies active subscribers. Publish/subscribe serves as an effective flow-control mechanism: no notifications are sent for events that nobody is subscribed to (they may be cached or logged) and subscribers are notified only about the events that they have explicitly stated interest in.

In the management domain, some objects may have a large representation, making it inefficient to obtain the entire set when only a subset of the information is required. To satisfy this requirement, WS-Management provided for enhanced operations to perform CRUD operations on partial object representations. In the case of events, WS-Eventing defined only a basic "push" delivery mode for the delivery of events. WS-Management defined enhanced delivery modes such as a batched mode, wherein the publisher or its proxy batches a specified set of events before sending them to the subscriber, thereby enhancing the ability of the subscriber to deal with a larger set of event sources.

WSDM and Web Services Resource Framework (WSRF) were devised in the Global Grid Forum community to handle stateful resources and then they moved to OASIS, as they were applicable to the broader Web services community. WSDM consists of two parts: MUWS, which defines basic management operations for managed resources and MOWS, which addresses the management

of Web services. WSDM builds upon the WSRF, which provides details on how to represent and access the XML representation of a stateful resource [24]. The eventing communication paradigm for WSDM builds on WS-Notification (WSN) [25] and defines a standardized event format called WSDM event format [13].

The two specifications have a similar scope, use similar architectural approaches, and both rely on Web-services technology, resulting in both overlap and synergies between the two approaches. Having realized this, the authors of the two specifications started working on a common unified approach that would increase interoperability among components of management systems in general and specifically facilitate automated machine-machine communication such as between autonomic managers. This work is described in a white paper published in March 2006 [8], which outlines a roadmap for unifying the two approaches. The approach is to consolidate the underlying access (CRUD) to resource representations and eventing by defining two new specifications called WS-ResourceTransfer, WS-RT [26] and WS-EventNotification, WS-EVN (not yet published). WS-RT builds upon WS-Transfer and WSRF and provides mechanisms for performing CRUD operations on partial object representations. WS-EVN builds upon WS-Eventing and WSN by providing support for enhanced filtering on events, policy-based subscriptions, and treating subscriptions themselves as resources that are operated upon by using WS-RT semantics (this simplifies the processing of subscriptions as they are treated as managed resources and can be operated upon using CRUD semantics). Building upon the unified resource access and eventing mechanisms, the authors intend to define a common unified management profile, which will provide a common Web-services-based external access to autonomic elements.

SUMMARY

The increasing complexity of IT systems today warrants a new architectural approach, one that scales with the increasing scope, heterogeneity, and size of systems. The autonomic computing paradigm promises to cope with complexity and scale up to the needs of today's distributed systems.

In this paper, we highlight the major standards that enable components from heterogeneous sources to interact with each other. This interaction is a fundamental tenet that enables intelligent decision making at the lowest possible level in the systems management hierarchy.

We described the standards required for external interfaces for an autonomic element namely WS-Management and WSDM and the general design philosophy of converging these two approaches by unifying the underlying plumbing protocols.

ACKNOWLEDGMENTS

We thank Jeremy Duke, Johan Van De Groenendaal, Arvind Kumar, Udayan Mukherjee, and Mazin Yousif for their input.

REFERENCES

[1] "Autonomic Computing: IBM's Perspective on the State of Information Technology," at http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf*

[2] "An architectural blueprint," at http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf.*

[3] "Simple Network Management Protocol, SNMP," at http://tools.ietf.org/html/rfc3411*

[4] "Common Information Model Schema, CIM Schema," at http://www.dmtf.org/standards/cim/ *

[5] WS-CIM, at http://www.dmtf.org/standards/wbem/*

[6] Arora, Akhil et al., "Web Services for Management (WS-Management)" at http://www.dmtf.org/standards/published_documents/DSP0226.pdf*

[7] "Web Services Distributed Management (WSDM)," at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm*

[8] Cline, Kevin et al., "Toward Converging Web Services Standards for Resources, Events, and Management," at http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/Harmonization_Roadmap.pdf*

[9] "Java Management Extensions (JMX)," at http://java.sun.com/products/JavaManagement/*

[10] "Net, System.Management namespace," at http://msdn2.microsoft.com/en-us/library/system.management.aspx*

[11] "Intelligent Platform Management Interface (IPMI)," at http://www.intel.com/design/servers/ipmi/

[12] "IPMI CIM Mapping Guideline, Intel, Dell, HP, Avocent," ftp://download.intel.com/design/servers/ipmi/mapguide.pdf

[13] "WSDM Event Format (WEF)," at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm*

[14]Bajaj, Siddharth et al., "Web Services Policy Framework (WS-Policy)," at http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf*

[15] Dublish, Pratul et al., "Service Modeling Language," at http://www.serviceml.org/SML-200607.pdf*

[16] Nadalin, Anthony et al., "Web Services Security: SOAP Message Security 1.1," *WS-Security 2004* at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss*

[17]Anderson, Steve et al., "Web Services Trust Language," *WS-Trust* at ftp://www6.software.ibm.com/software/developer/library/ws-trust.pdf*

[18]Anderson, Steve et al., "Web Services Secure Conversation Language," *WS-SecureConversation* at ftp://www6.software.ibm.com/software/developer/library/ws-secureconversation.pdf*

[19]Della-Libera, Giovanni et al., "Web Services Security Policy Language WS-SecurityPolicy," at ftp://www6.software.ibm.com/software/developer/library/ws-secpol.pdf*

[20]"Security Assertion Markup Language (SAML)," at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security*

[21]Alexander, Jan et al., "Web Service Transfer (WS-Transfer)," at http://specs.xmlsoap.org/ws/2004/09/transfer/WS-Transfer.pdf#search=%22WS-Transfer%22*

[22]Box, Don, "Web Services Eventing (WS-Eventing)," at http://www.w3.org/Submission/WS-Eventing/*

[23]Alexander, Jan et al., "Web Services Enumeration, (WS-Enumeration)," at http://www.w3.org/Submission/WS-Enumeration/*

[24]Graham, Steve et al., "Web Service Resource 1.2 at http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf*

[25]"Web Services Notification, WSN," at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn*

[26]Reistad, Brian et al., "Web Services Resource Transfer (WS-RT)," at http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer/WS-ResourceTransfer.pdf*

[27]Alves, Alexander, "Web Services Business Process Execution Language Version 2.0," at http://www.oasis-open.org/committees/download.php/18714/wsbpel-specification-draft-May17.htm*

[28]Fielding R, "Hypertext Transfer Protocol–HTTP/1.1," at http://www.w3.org/Protocols/rfc2616/rfc2616.html*

[29] "Schematron, ISO/IEC 19757-3," at http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=40833&ICS1=35&ICS2=240&ICS3=30*

[30]*Information Technology Infrastructure Library*, ITIL, at http://www.itil.co.uk/*

[31] "Extensible Markup Language" at http://www.w3.org/XML/*

[32] "SOAP" at http://www.w3.org/TR/soap/*

[33] "Web Services-Addressing" at http://www.w3.org/2002/ws/addr/*

[34] "Web Services Description Language" at http://www.w3.org/TR/wsdl*

AUTHORS' BIOGRAPHIES

Vijay Tewari is a senior architect and researcher with Intel's Corporate Technology Group. He received his B.S. and Bachelor of Technology degrees from Jawaharlal University, India and his M.S. degree from the University of Minnesota. He has co-authored multiple Web service specifications particularly for the management domain. His research focus is distributed systems and systems manageability. His e-mail address is vijay.tewari at intel.com.

Milan Milenkovic is a principal engineer in Intel's Corporate Technology Group. He received an M.Sc. degree in computer science from Georgia Institute of Technology and a Ph.D. degree in computer engineering from the University of Massachusetts at Amherst. His current research interests include distributed systems, virtualization, and management by "instrumenting the world." Milan's e-mail address is milan at intel.com.

Towards Autonomic Enterprise Security: Self-Defending Platforms, Distributed Detection, and Adaptive Feedback

John M. Agosta, Corporate Technology Group, Intel Corporation
Jaideep Chandrashekar, Corporate Technology Group, Intel Corporation
Denver H. Dash, Corporate Technology Group, Intel Corporation
Manish Dave, Information Technology, Intel Corporation
David Durham, Corporate Technology Group, Intel Corporation
Hormuzd Khosravi, Corporate Technology Group, Intel Corporation
Hong Li, Information Technology, Intel Corporation
Stacy Purcell, Information Technology, Intel Corporation
Sanjay Rungta, Information Technology, Intel Corporation
Ravi Sahita, Corporate Technology Group, Intel Corporation
Uday Savagaonkar, Corporate Technology Group, Intel Corporation
Eve M. Schooler, Corporate Technology Group, Intel Corporation

Index words: self defending, autonomic, detection, self-defending, virtualization

ABSTRACT

Enterprises today face a constant barrage of security threats stemming from worms, viruses, trojans, and other malware. This is in spite of significant levels of investment in defenses such as firewalls and anti-virus and anti-spam products. Dealing with these attacks cost U.S. businesses over $67 billion last year. To make things worse, malware designers are staying slightly ahead of the game with a visible trend emerging of malware becoming stealthier and much harder to detect. Successfully mitigating security threats requires a multi-pronged approach that must include mechanisms that address different levels of the enterprise. Today's enterprise networks are very complex because of the sheer number of heterogeneous enforcement points (involving multiple product lines from multiple vendors), the mobility of end-points, and most importantly, the scale of the network itself (typical enterprise networks contain hundreds of thousands of hosts). Given these challenges, protecting the enterprise is a significant task, and relatively little work has been done in this area up to this point. In fact, enterprise policy management today is still largely a manual, ad-hoc process, lacking useful higher-level abstractions and a systems-level view in the application of security policies. In essence, there is very little *autonomics today* in the operational aspects of enterprise security management.

In this paper, we argue that a successful strategy must not rely on silver-bullet-like approaches, but rather should target different levels of the enterprise. We describe three key building blocks that address different levels of the enterprise and show how these, when used together, provide truly autonomic security for the enterprise network. At the lowest level, we describe the notion of self-defending end-hosts, i.e., hosts that can detect integrity violations or subversion. We show how Intel® Active Management Technology† (Intel AMT) [18] and Intel® Virtualization Technology$^{\Delta}$ (Intel VT-x) can be used to provide software integrity services and enable the end-host to regulate itself. At the next level, we describe how this capability can be significantly enhanced by allowing end-hosts to collaborate and detect network-wide anomalies (such as infections, attacks, etc.). Finally, we propose a feedback-based security management architecture for enterprise networks that views the enterprise at a higher level of abstraction. With these three capabilities, networks built using Intel® platforms can provide autonomic control and protect themselves from day-zero threats, consistently, with enterprise security policies, and without intervention from administrators.

INTRODUCTION

Among the list of challenges faced by IT departments, security consistently ranks as one of the top year after year as reported by the Gartner Group. Firewalls, anti-virus software, and other similar protection mechanisms are ubiquitous in corporate networks. In spite of this, there is little respite from the spate of worm attacks, viral infections, host compromises, spyware, etc. It is estimated that protecting against these threats costs businesses about $67 billion a year, in the U.S. alone [8].

Among the myriad security threats that are seen today, worms and other kinds of self-propagating malware are, anecdotally at least, the single most challenging problem that the Internet faces. The homogeneous makeup of the Internet makes it very vulnerable to these kinds of attacks while its rich connectivity makes it very easy for worms to propagate. Thus far, state-of-the-art Intrusion Detection Systems (IDSs) have made use of the quickly spreading nature of these attacks to identify them with high sensitivity and at low false positive (FP) rates. However, in the race between worm designers and security vendors, the former always seem slightly ahead: there is a growing trend towards the use of stealthy worms that use evasion techniques to cloak their presence on an infected host. Such worms render existing IDSs ineffective. In addition, existing IDS products do not protect from day-zero exploits, which malware designers are adopting far more than in the past.

In this paper, we first describe host-level protection, i.e., *self-defending platforms*, that can defeat (or at least detect) attacks that attempt to subvert the Operating System (OS). This is done using a runtime integrity service that automatically improves the security and robustness of networked platforms by leaving no place for malware to hide. We then describe *distributed detection and inference*, a method whereby protected systems can collaborate (or "gossip") to detect (and signal) network-scale attacks (or infections). Untrusted systems cannot benefit from such gossip protocols as this network-wide information is secured by protecting the software on the end-point. Finally, we describe the *adaptive feedback framework*, a framework in which the "network state" as determined by the distributed detection can trigger feedback mechanisms to mount an automated response to day-zero threat conditions. The rationale for using network-wide information in our approach is also to enforce the autonomic response more intelligently, in a holistic manner, as compared to the more ad-hoc 'per device type' enforcement approach, and to target the most effective control points. Figure 1 shows a schematic of the entire architecture, showing how the three different mechanisms may interact with each other.

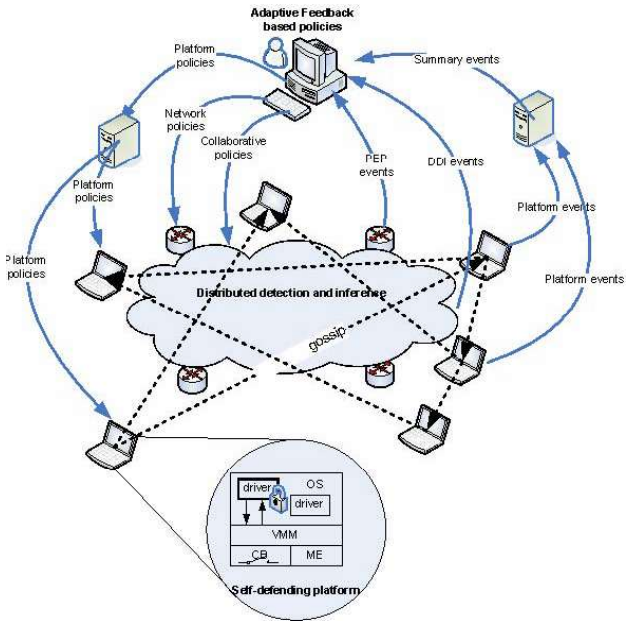

Figure 1: Overall architecture of an Autonomic Enterprise Security system. The architecture consists of three mechanisms: (1) self-defending platforms protect individual end-hosts; (2) distributed detection correlates alarms across end-hosts; and (3) the adaptive framework delivers security polices (as a response to a network threat) to the most effective control points.

SELF-DEFENDING PLATFORMS

Our approach to create a self-defending platform leverages Intel Virtualization Technology (commonly known as VT-x) to efficiently enforce memory protections, and Intel Active Management Technology (Intel AMT) [18] to enforce network policies on the end-point. We provide a brief background on VT-x for the sake of completeness. Interested readers can find a detailed description of this technology in this specification [1]. In the rest of this section, we assume that the reader is familiar with IA-32 instruction set architecture, the details of which can be found in the referenced Intel specification [2].

The term "virtualization" refers to the technique of partitioning a hardware platform into multiple virtual partitions called Virtual Machines (VMs) and running independent OSs in those virtual partitions. A layer of privileged software called the Virtual Machine Monitor (VMM) provides an abstraction of the hardware to each VM. At a high level, VT-x provides hardware support for virtualizing the CPU and the memory subsystem. By providing this hardware support, VT-x helps simplify VMMs, allowing them to support a wider range of legacy and future OSs without compromising performance or

security. One of the key applications of the VMM is to maintain control over the physical memory of the platform. One way the VMM can achieve this on a VT-x CPU is by maintaining a set of parallel page tables for each OS running on the platform. The page tables maintained by the VMM are called the Active Page Tables (APTs). These page tables reference the real physical memory on the platform and are used by the processor for address translation. Each guest OS maintains its own page tables, called Guest Page Tables (GPTs). The VMM synchronizes APTs with GPTs in software using an algorithm called the Virtual TLB (VTLB) algorithm. The VTLB algorithm behaves similar to a processor Translation Lookaside Buffer (TLB). The algorithm relies on the VMM's ability to trap events such as page faults and execution of certain instructions (for e.g., INVLPG, MOV CR3), that are used by a legacy OS to manage virtual memory. A detailed description of this algorithm is out of the scope of this paper, and interested readers are referred to [3] for further details.

SELF-DEFENDING PLATFORM ARCHITECTURE

We propose to build an autonomic platform with the following objectives:

- Enable programs to securely specify their in-memory structure and access policies to the platform, including interaction with verified shared libraries and shared components.
- After the program is recognized by the platform, detect changes to the program code/data.
- Prevent use violation, by bugs or malware, by enforcing code entry at well-known offsets.
- Prevent malware from root-kiting (invalidly hooking) critical services on the platform.
- Self-remediate the platform when an attack is detected.

Our architecture leverages privileges provided by VT-x to the VMM to monitor and protect software agents running inside a guest OS from other components of the same OS. This ensures that malware running in the guest OS is not able to tamper with critical software agents running in the same OS, even if the malware is able to achieve the highest privilege level within that OS. The VMM is measured at boot time as part of the measured boot sequence. Figure 2 shows the architectural overview.

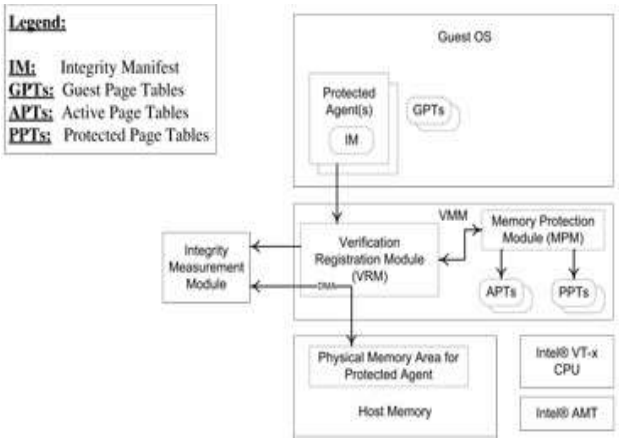

Figure 2: Self-defending end-point architecture

All the agents that need protection implement a data structure called the Integrity Manifest (IM). The IM is cryptographically signed by the vendor or another trusted source and it describes the structure of the agent when it is loaded into memory correctly. Other components of the architecture include the following:

- The Integrity Measurement Module (IMM) to use the IM to validate the agent in-memory. The IMM itself is part of an isolated secure partition in the platform such as a Service OS (VM).
- The Verification Registration Module (VRM) to provide hyper calls for (de)registration.
- The Memory Protection Module (MPM) to manage the page tables.

At a high level, the setup phase for self-defense of software agents proceeds as follows:

1. The guest OS loads the software agent into memory and starts executing it.
2. The agent (or 3rd party) registers with the VRM and requests memory protections. As a part of this registration request, the agent provides its IM to the VRM.
3. The VRM forwards the unauthenticated request along with the Manifest to the IMM. The IMM validates the signature of the manifest belongs to a trusted vendor or other source using a certificate authority signing hierarchy. Integrity of the manifest is then verified using this signature to ensure it was not modified.
4. The IMM uses highest privilege memory accesses to read the agent in the host *physical* memory, and it validates that the agent has not been altered, by comparing the agent's loaded image with the IM's description of that image [7].

5. If the IMM is able to validate that the agent is correctly loaded in memory, the VRM forwards the registration request to the memory protections module (MPM)
6. The MPM creates a Protected Page Table (PPT) for the agent code and data to protect the agent from run-time memory attacks. The MPM ensures this by removing the references to the agent's code and data from the APT used by the OS.

Once the setup described above is completed the MPM enforces various system policies such as preventing invalid jumps into the code and disallowing outside (non-agent code) read and/or write access into protected agent data. The MPM ensures that it can see all these events without risk of circumvention, due to the hardware virtualization capability of the processor. Specific events such as page faults cause transitions into the MPM that moderates these access attempts.

We have prototyped this architecture on an Intel VT-x based platform with Intel's lightweight VMM (LVMM) and used it to protect network drivers running in Windows XP* guest OSs.

STANDARDS FOR INTEGRITY MEASUREMENT

The purpose of integrity measurement is three-fold: 1) to locate the program in host physical memory, 2) to verify that the program has been loaded into the memory without any tampering, and 3) to ensure that the program executes correctly. In this architecture, we propose a standard structure to enable runtime in-memory verification of the software agent in the IM.

The IM data structure contains cryptographically verifiable information about each code and/or static data section of the binary program file that would be loaded to the memory. This includes cryptographic hashes [4] of the various sections that are to be verified. However, when the agents get loaded into memory, the OS performs relocation operations on these sections, in turn modifying the contents of these sections. Thus, to be able to verify the integrity of these sections at runtime, the IM includes information about locating and verifying these modifications were executed correctly. Specifically, the IM contains the following:

1. A cryptographic hash of the contents of each of the sections.
2. External symbols (functions or global data) that are referred to by the sections of interest.
3. Relocation entries that will be used by the OS to relocate the sections of interest (this enables the IMM to revert the relocation changes made by the OS, before the IMM computes the cryptographic hash for verification).
4. A list of allowed entry-points into the code (offset in a section). For example, each entry-point in a protected program may represent a function exported by the program.

The IM has been proposed as a standard in the Trusted Computing Group for evaluating program integrity at runtime. Other auxiliary industry standards activities such as the TCG Trusted Network Connect (TNC) [5] and the IETF Network Endpoint Assessment group [6] can benefit from such a standard to attest platform integrity to the network.

The combination of these standards with built-in platform integrity services allows for fully autonomic monitoring and verification of programs and their behavior at runtime. No user intervention is required to establish and maintain the proper execution of programs once protections have been enabled. Furthermore, the platform can automatically attest to the presence and status of the software executing on it to network access control mechanisms, report errors, or warn of possible attacks to trusted peers in the network, as we discuss next.

DISTRIBUTED DETECTION AND INFERENCE

As presented in the previous section, a secure platform is the best way to secure the enterprise, but it may take a long time for enterprises to upgrade all end systems. Therefore it is still important to have a reliable detection system throughout the enterprise. In this section, we describe a framework to combat the increasingly urgent problem of intrusion attempts within an enterprise. Traditional defenses have relied on perimeter mechanisms such as firewalls to protect the inside of an enterprise from external threats. However, the modern enterprise has very loosely defined boundaries and hosts are generally free to move in and out. Once infected hosts return into the enterprise, the infecting malware is free to spread relatively unchecked. Conventional detection schemes are based on observing traffic entering and leaving aggregation points around the enterprise. These schemes, while moderately successful, have several limitations, the most severe of which is that they are not very good at detecting slowly spreading worms that try to blend in with normal background traffic. The counter-measure that is often used for this type of worm is to use a variety of sensors around the network that measure and track different traffic features and that correlate one piece of information with another piece of information. Our framework extends this idea to the logical extreme: we

consider each end-host in the enterprise to be a potential sensor (or *Local Detector*) and we allow the end-hosts to exchange information and corroborate the state of the network, i.e., whether it is infected or not. As we will show, such a system is able to detect slowly spreading network anomalies at a very low FP rate (which is much lower than those associated with conventional methods and tools). Briefly, there are several intuitive ideas for a collaborative, host-based framework:

1. IDSs deployed selectively might not see any worm traffic for a long time and perhaps see it only when it is too late. Collaboration is seen as a way to remedy this; systems that allow multiple IDSs to share information have been shown to provide greater "coverage" in detection [9, 10, 11, and 12].
2. Analysis of network traffic at the host level allows the weak signal to be compared to a much smaller background noise-level, so the signal-to-noise ratio can be boosted by orders of magnitude compared to an IDS that operates within the network.
3. Host-based detectors can make use of a richer set of data, possibly using application data from the host as input into the local classifier.

The detection and inference framework we describe here is quite simple: end-hosts contain Local Detectors (LDs) that are meant to detect anomalous behavior at the end-host. This could be by means of system integrity checks, watching outgoing network traffic, looking for anomalous behavior, etc. Periodically, the LDs gossip their *local* state (whether an anomaly was detected in some preceding window, or not) to other hosts. Some (or perhaps all) of the nodes in the network also contain Global Detectors (GDs). Their function is to aggregate the signals received from LDs (each GD receives signals from some number of LDs in the network). Thus, each GD computes the probability that a network-wide anomaly is occurring.

In the rest of this section, we describe the LDs that we use and describe how information from different LDs is combined into a single measure. Subsequently, we describe simulation results that compare the performance of different models.

Local Detectors

Simply put, an LD (end-host-based detector) is any entity that generates an output signal, taking as input the state of the end-host. For our specific purpose, the LD simply generates a Boolean signal that is true if an "anomaly" is detected, and false otherwise. We also assume that the LDs are weak in the sense that they may have a high FP rate, and are non-specific, so are likely to fire for a broad range of anomalous behavior. The LD implementation that we use in our proof of concept is quite simple: it counts the number of new "network connections" that are initiated by the host in a certain time window. If this count exceeds a pre-defined threshold, it assumes an anomaly exists. In Figure 3, we show the distribution of the number of new connections initiated by a host in a 50s interval. The plot corresponds to network traces collected at 37 hosts in the Intel corporate network over a five-week period. Also shown in Figure 2 are the average rates for a number of known worms (MS Blaster, Slapper, Code Red II, etc.). Clearly, these propagate at a rate that is quite high and clearly stand out in the distribution. Thus, using a very high threshold for our LD (about 200 connections per 50s) would be sufficient to detect the said worms accurately and without many FPs. However, the trend that has been recently observed is that worms are getting slower and slower, moving to the left side of the distribution. In order to demonstrate the effectiveness of our system to detect slow worms, and at a low FP rate, we push the LD threshold down by orders of magnitude and set it to 4 connections per 50s interval. Note in Figure 2, this number is well within the bulk of the normal traffic distribution. If an individual LD was used by itself, and at this threshold, it would have generated thousands of false alarms over the five-week period; however, it would have successfully detected worms operating at a much slower rate. Thus, a simple way to create a weak, non-specific LD is to drastically reduce the threshold of some standard heuristic, although other standard anomaly detection techniques can be used as well.

The alarms raised by individual LDs in our system are "gossiped" to other peer hosts. Thus, LDs periodically share their "belief" with other nodes in the network. Conceptually, we could think of end-hosts containing an LD (which operates as above) and also a GD which simply aggregates the information received from different LDs, based on some model to compute a network-wide belief. In the following section, we briefly discuss a few models that do this.

A valid concern is how might such a system protect the computing environment against the malicious corruption of the LDs or the GDs. One approach would be to adopt integrity services on the subset of nodes running the distributed detection algorithms–the number of nodes required is substantially fewer than the entire enterprise. An alternate approach, besides the adoption of integrity services, is to place the LD and GD functionality in protected hardware, rather than in more vulnerable software. Work is underway to study other vulnerabilities, e.g., how to make the system robust to some number of rogue LDs sending misinformation to GDs and thereby skewing the analysis completely.

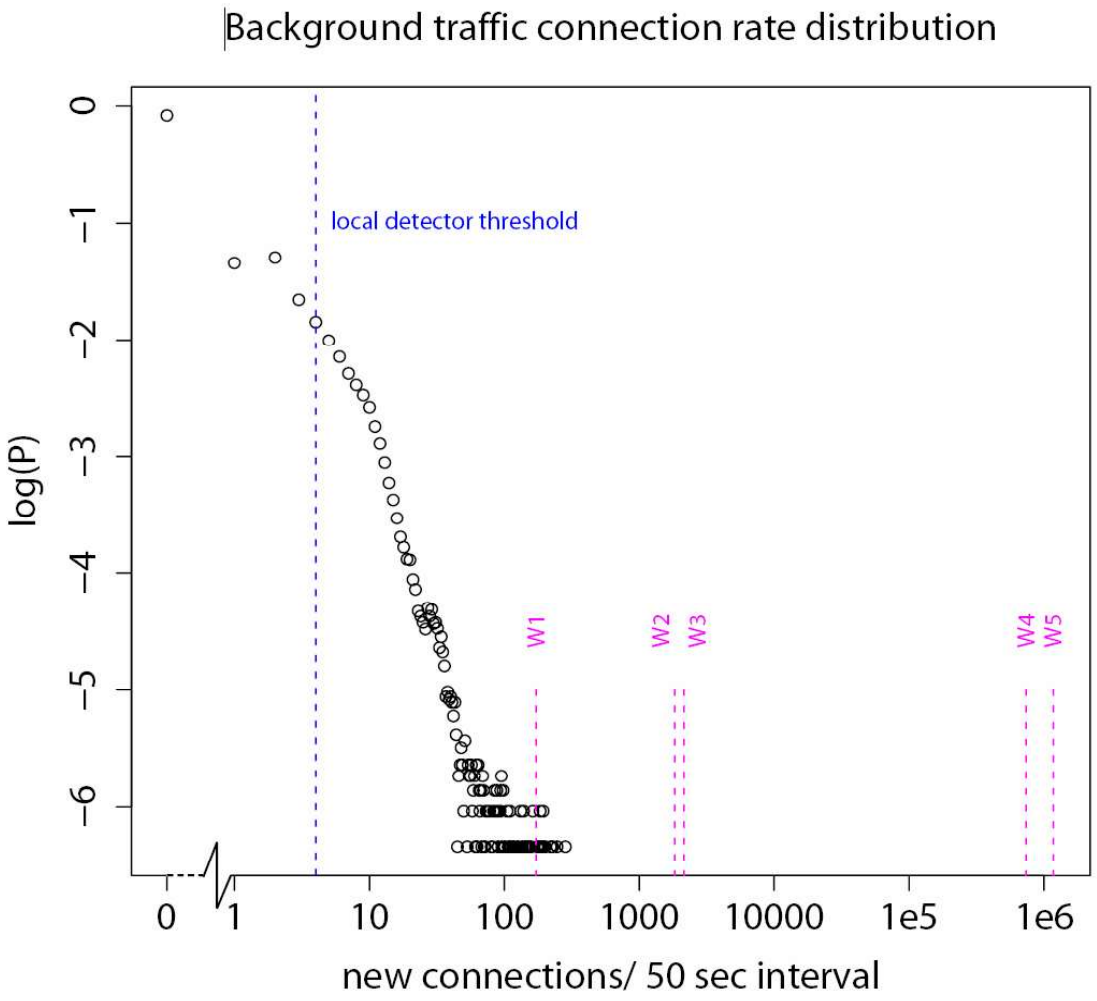

Figure 3: The distribution of the number of initiated connections per 50s interval. Propagation rates of previous worms are indicated as W1=MS Blaster, W2=Slapper, W3=Code Red II, W4=Slammer, W5=Witty. The horizontal line denotes the threshold used in the local detector (4 CPI).

Global Detectors

There are potentially several models that allow combining of the local beliefs, received from individual LDs, into some "global" belief. The simplest possible way would be to count the number of positive firings and threshold this value (that is, use the PosCount model). Another potential model is the CuSum detector, well known in the area of statistical process control, which is used to detect deviations from some mean (over a statistic of interest). However, a drawback with these simpler models is that they do not really support heterogeneous LDs, i.e., detectors of varying "quality." In contrast to these baseline techniques, models based on Dynamic Bayesian Networks (DBN) [13] overcome this shortcoming by taking into account the FP and True Positive (TP) rates of individual detectors in a systematic manner. Essentially, DBNs are a principled formalism for expressing independence relations while modeling temporal (stochastic) processes. In our work, we explore two DBN instances, namely the Change-Point DBN and the Epidemic DBN. The former assumes that up to some time, t_{cp},, the network as a whole is not in an anomalous state; whereas after t_{cp} the network is. In contrast, the latter models the spread of exponentially growing signals (anomalies, in our specific context) in a system. Clearly, each of these models is well suited to specific applications. For instance, if a system were to use high quality LDs (very low FP rate and very high TP rate), then presumably, the PosCount model would perform quite well (and has the advantage of low computational overhead). In the next section, we compare the performance of our system assuming very general (but weak) LDs.

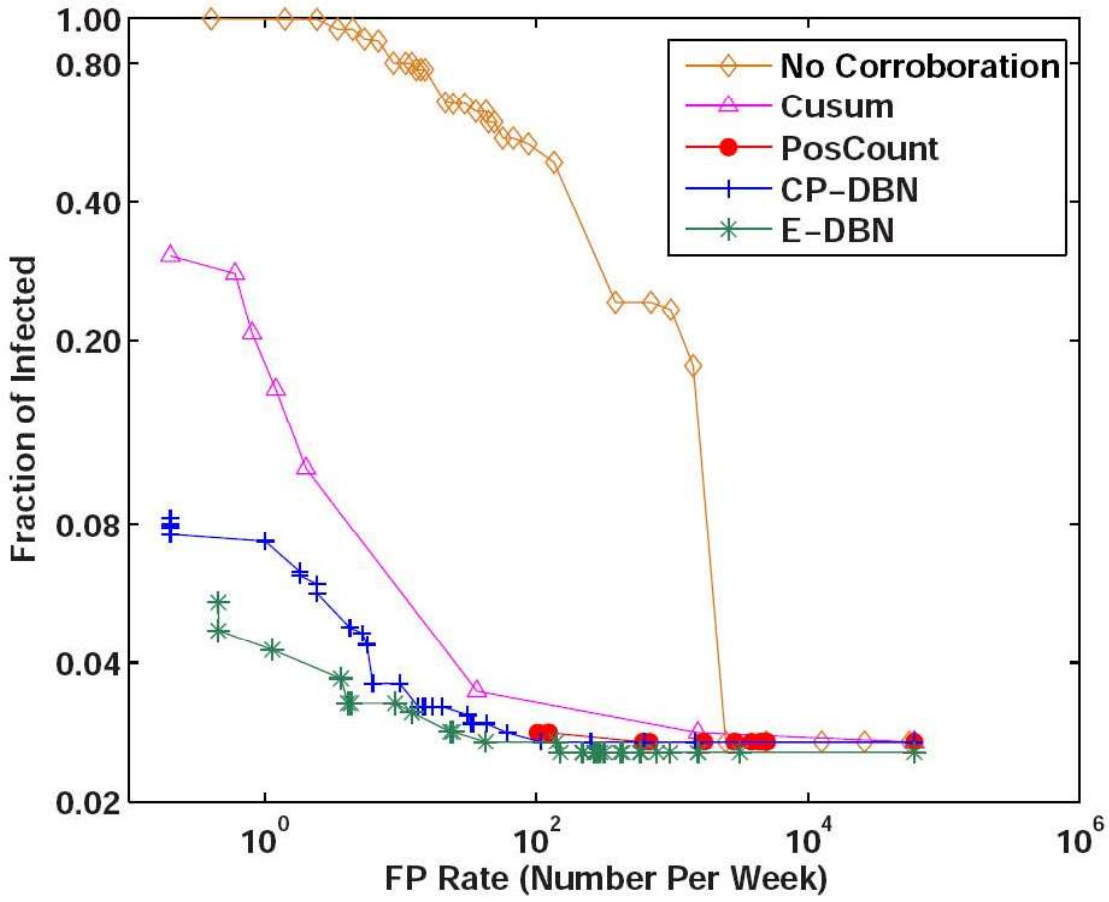

Figure 4: Performance comparison of various Global Detector models with a stand-alone Local Detector (No Corroboration). The x-axis sweeps through the false positive rate (numbers of false alarms per week) and the y-axis plots the fraction of the network infected at the time of a global detection. Notice that the DBN models have the lowest FP rates and also facilitate early detection, whereas the stand-alone Local Detector either achieves low FP rates or a low fraction infected, but never both simultaneously.

SIMULATION STUDIES

Here we present a set of results that highlight the advantages of the system we are proposing. Results are based on data collected from a set of 37 machines over a five-week period. The specific LDs used were as described previously (the heuristic used is the number of distinct connections initiated in a 50-second window; an alarm is raised if this value is more than 4). On top of the actual traffic traces that we replayed, we super-imposed worm traffic that was generated according to an underlying distribution parameterized by the worm spread rate *S,* which is the number of attempted infections per worm per unit time, and the address density of the network that the worm is infecting. The results shown here use an *S=1/20* cps and an address density of 1/1000. In other words, a worm will generate a new infection attempt every 20 seconds, which has a 1/1000 chance of reaching a valid (one capable of being infected) destination. Also, the LDs in the simulation had an epoch of 10 seconds, i.e., each LD, every 10 seconds, picks a node at random and shares its belief (on or off) with it.

All of the GD models that we explored (except for the PosCount model) required estimates of the true and FP rates of individual LDs. To make the comparison fair, we used the same parameters across all the LDs (DBNs account for the TP/FP rates in a principled fashion and, presumably, would do better than the other detectors if heterogeneous detectors were used).[‡]

[‡] A TP rate of 0.6 and a FP rate of 0.2 were used for the simulations, the results of which are presented here.

Figure 4 shows the results for all GDs when the LDs pass one message per 10-second epoch. The sweet spot on this curve is in the lower-left corner: this implies a low FP rate is achieved where a global alarm is triggered before many hosts are infected. As can be seen from the figure, both DBN models clearly outperformed the baseline GD models. The PosCount detector at an FP rate of 100 per week will only raise a detection after the entire network is infected. The CuSum detector is able to operate at our target FP rate of 1 per week, but it detects at a much higher infection percentage (> 20%) than the DBN models. The CP-DBN, which is not designed to detect an epidemic spread, can still achieve the 1 FP per week while allowing only an 8% infection, while the E-DBN model that was specifically designed for this scenario can detect with an infected-host percentage of about 4%. The top line shows the results that are obtained as you sweep through the LD thresholds with no corroboration. Clearly, the E-DBN detector outperforms the rest, and even the CP-DBN detector performs better than the simpler models.

In the next section, we describe how such a detection framework can trigger a containment response from control points deployed in the enterprise.

ADAPTIVE FEEDBACK

With policy enforcement points such as those that reside on trusted, self-defending platforms, and intelligent intrusion detection systems as discussed above, it is important to have an overarching architecture that correlates distributed information, local decisions, and individual device actions so that we have a closed-loop for autonomic management. In this section we discuss the management building block of security autonomics for the enterprise–the adaptive policy management architecture. As shown in Figure 5, the main components of this adaptive, self-management architecture include a Manager of Managers (MOM), Intermediate Device Managers (IDM), a Policy Enforcement Point (PEP), and a Policy Feedback Point (PFP). The role of the MOM is to provide autonomic and centralized management of enterprise security policies, including translating business-driven policies into device-specific controls and pushing them to specific devices through IDMs. A main distinction of this architecture from the standard policy-based network management architecture [15] is the introduction of PFP and the control feedback loop. The PFP collects and processes intrusions, security alerts, violations, and other abnormal behaviors from a variety of systems (e.g., intrusion detection systems, system logs, etc.), and it sends such data as control feedback to the MOM. With such feedback information the MOM then determines the necessary control updates, which can either lead to automatic actions pushed to the network, or the feedback data and recommended actions can also be used to help the network administrator make corresponding human decisions on control updates.

The benefit of using the centralized MOM with adaptive feedback for automated response to a network condition is that it provides coverage for the entire network on a holistic level. This makes the autonomic response more intelligent than a "per security enforcement point" or even a "per security device type" detection and mitigation approach. These traditional per-device or per-device-type models, where each individual device only has knowledge of the local network events and will make a policy decision based on this limited information, has a potential to cause co-relation and conflicts, as there is no information sharing between the various enforcement points on the policies (both static and dynamic) and network events. Our approach ensures that all events are co-related and the action taken is applied at the most effective control point, which in our prototype example, was the network enforcement point closest to the source. Since the MOM has the knowledge of the capabilities for each enforcement point, it is able to make an intelligent decision on the placement of these dynamic controls.

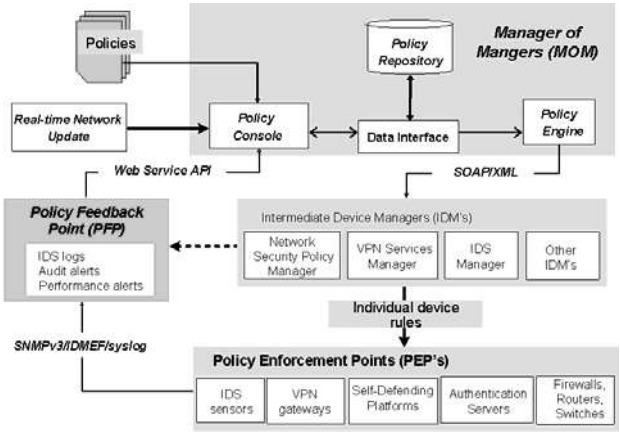

Figure 5: Conceptual architecture

Another distinguishing component of this architecture is the capability-based policy specification [14], which enables high-level policies to be implemented transparently on end-devices and common policies to be pushed from a central location to various network devices from different vendors. Compared to most existing policy specification models [16], our policy schema is a consistent and extensible data model for network security policy representation. The important notion of this schema is the ability to specify heterogeneous devices in terms of their capabilities. This approach allows the overall data model to be extensible, since newer devices can be added by describing their capability data models. This approach provides a platform that allows consistent security policy specifications and standard device capability specifications to be developed. Advantages of this policy

specification include security policy specification independent of device differences, which allows for extensibility and algorithmic mapping; capability knowledge, which allows conflict resolution and threat analysis during security policy definition; data model that includes network and end-point nodes, which reduces the chances of lapses in security; a combined data model, which allows for co-related feedback events from the network and reduces the administrative (human) overhead of hand mapping a high-level security policy down to a heterogeneous set of devices, each with their own configuration methods and syntax.

Market Survey on Security Policy Management

We implemented a prototype of the above architecture with representative "real-world" operational enterprise IT use-cases to demonstrate the benefits of this architecture. We studied 15 commercial products/solutions from a broad variety of vendors, including the market leaders in network and security management, based on a survey by the Burton Group [17]. Based on our study we broadly categorized these products into three types: (1) vertical solutions with several desirable capabilities, but focused on single vendor devices and lacking support for management and integration in a multi-vendor enterprise environment; (2) multi-vendor network configuration and device management systems, most of which were designed for management, provisioning, automation for network configuration and Quality of Service (QoS), but not for security management; and (3) what we believe were the first-generation autonomic management solutions: these products have many of the required capabilities and can be evolved to cover for some of the missing capabilities necessary in today's enterprises. We selected two products from the third category for further operational validation against our use-cases, which are described in the following section.

ENTERPRISE USE CASES AND TEST RESULTS

We developed and tested our adaptive management architecture against several representative use cases to validate the architecture for a large enterprise network that consists of diverse security enforcement points with varying security capabilities, and to validate the usability of such architecture in highly complex, heterogeneous, multi-device, multi-protocol networks. The following are use cases most relevant to autonomic management:

- *Degraded mode of operation*. This use case is to demonstrate how a network policy reacts when the network is forced to move into a degraded mode of operation due to unexpected change (e.g., a denial of service attack).
- *Dynamic policies with feedback from the network*. The purpose of this use case is to demonstrate the ability to present real-time information driving adaptive change in the network configuration to secure against a threat detected by our distributed detection systems.
- *Automatic detection, resolution, and verification of policy conflicts*. Other related use cases we developed include complex policy enforcement, high-level policy definition and abstraction, domains of constant policies, and visualization.

Figure 6 is a diagram of the lab network used to conduct the use case tests. The lab was set up so as to mirror a large-scale, heterogeneous IT production environment as close as possible. The setup includes typical network security products and technologies such as firewalls, network intrusion detection and prevention systems (NIDS/NIPS), routers, and switches. The network topology for the lab also mirrors a typical enterprise network with different zones (Internet, demilitarized zone <DMZ>, and Intranet). We used a Security Event Management System (SEMS) as the central repository of all network events. SEMS stores event streams from various sources such as IDSs, firewall logs, router logs, etc. in a database and performs a network-level holistic co-relation and aggregation of the events to generate real-time alerts for the entire network versus the per-device approach. These network-wide co-related alerts are used to trigger a policy update via the MOM to the appropriate network control point(s). In our tests, we were able to demonstrate the adaptive feedback concept using these active components. To simulate a network degradation, we injected a stream of abnormal network traffic using (denial-of-service-like) UDP-based malware attacks. The SEMS was able to detect these events, co-relate them, and send the alert to the MOM console. In response to this event, the MOM console automatically created the dynamic policy update as a response to the threat. In this example, the control update was to block the source IP address that was generating the attack traffic; this update was pushed to the network enforcement point closest to the source of the attack. We successfully verified this with multiple scenarios where the attack traffic was blocked automatically.

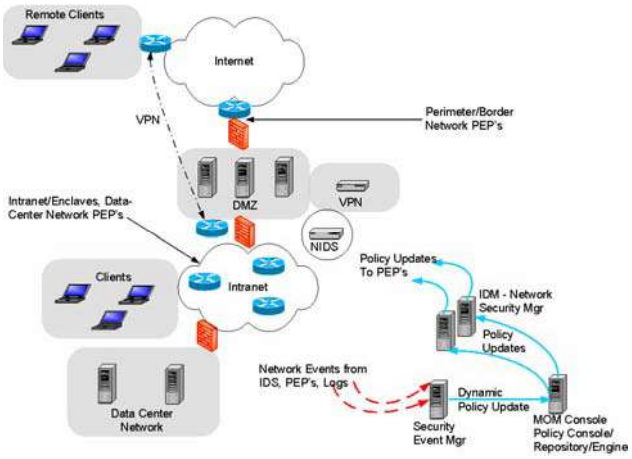

Figure 6: Lab network

CONCLUSION

Today's enterprise networks are extremely complex entities, containing a very large number of hosts and spread over many locations. Intrusion attempts due to self-propagating code are becoming an increasingly urgent problem, in part due to the homogeneous makeup of the Internet. Recent advances in anomaly-based IDSs have made use of the quickly spreading nature of these attacks to identify them with high sensitivity and at low FP rates. However, slowly propagating attacks are much more difficult to detect because they are cloaked under the veil of normal network traffic, yet can be just as dangerous due to their exponential spread pattern. We describe a framework where hosts running local IDS instances can corroborate the likelihood of an attack in an autonomous, decentralized fashion, by gossiping their local beliefs to other participating hosts. Securing such a network and ensuring the correct operation of its component elements is an extremely challenging task. A large part of the complexity lies in the sheer heterogeneity of enforcement points, each of which must be configured and managed in slightly different ways. The state of the art is quite lacking: enterprise policy management is still largely a manual, *localized* process, lacking a higher level, network-wide view. There is an urgent need for a framework to unify the disparate components to allow for a more autonomic operation and maintenance of the network.

In this paper, we described three building blocks that move us closer to realizing the ultimate goal, that of "autonomic operation" of the enterprise. First, we described the notion of *self defending platforms*, which enable an end-host to detect program-level anomalies and unauthorized modifications. Next, we described the concept of *distributed detection* and inference, where end-hosts collaborate among themselves to reason about the state of the entire network. Finally, we discussed an *adaptive policy management* architecture that can supplement the previously discussed capabilities. At a high level, the policy framework can support and complement the other two building blocks. It can do this by providing a channel for anomalies signaled by these building blocks to percolate up to entities that have a system-wide view of the network and to translate remedial actions determined at the system level into actionable tasks at the lower-level building blocks. That is, they can serve as a feedback channel from higher-level entities to the host-based mechanisms. To demonstrate the efficacy of our framework, we use the example of a DoS attack on a synthetic network and show how it can be stopped by means of our feedback mechanism.

ACKNOWLEDGMENTS

We acknowledge our reviewers Prashant Dewan, Dennis Morgan, Kai Miao, Toby Kohlenberg and Georgios Theocharous. We also acknowledge Greg Kime, Ravi Sahita, and Michael Sparks for the development of the adaptive feedback management architecture.

REFERENCES

[1] Intel® Virtualization Technology Specification for the IA-32 Intel® Architecture at http://www.intel.com/design/pentium4/manuals/index_new.htm

[2] *IA-32 Software Developers Manual*.

[3] J. E. Smith and R. Uhlig, "Virtual Machines: Architectures, Implementations, and Applications," *Tutorial 1, HOTCHIPS* 17, August 2005 at http://www.hotchips.org/archives/hc17/1_Sun/HC17.T1P2.pdf*

[4] *The Secure Hash Algorithm (SHA-1)*, National Institute of Standards and Technology, FIPS PUB 180-1, "Secure Hash Standard," U.S. Department of Commerce, April 1995.

[5] *Trusted Computing Group–Trusted Network Connect* at https://www.trustedcomputinggroup.org/groups/network/*

[6] *IETF Network Endpoint Assessment BOF* at http://www1.ietf.org/mail-archive/web/nea/current/index.html*

[7] T. Schluessler, H. Khosravi, G. Nagabhushan, R. Sahita, U. Savagaonkar, "Runtime Integrity and Presence Verification for Software Agents," at http://www.intel.com/technology/magazine/research/runtime-integrity-1205.htm

[8] *FBI Computer Crime Survey, 2005* at http://www.fbi.gov/page2/jan06/computer_crime_survey011806.htm*

[9] Bailey, M.; Cooke, E.; Jahanian, F.; Nazario, J.; and Watson, D., "The internet motion sensor: A distributed blackhole," in *Proceedings of the* Network and Distributed System Security Symposium Conference Proceedings, Feburary 2005.

[10] Nojiri, D., Rowe, J. and Levitt, K., "Cooperative response strategies for large scale attack mitigation," in *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition*, April 2003.

[11] Anagnostakis, K. Greenwald, M., Ioannidis, S., Keromytis, A. and Li, D., "A cooperative immunization system for an untrusting internet," in *Proceedings of the 11th IEEE International Conference on Networks* (ICON), 2003.

[12] Agu Rajab, M., Monrose, F., and Terzis, A., "On the effectiveness of distributed worm monitoring," Technical report, John Hopkins University, 2005.

[13] Dean, T. and Kanazawa, K., "A model for reasoning about persistence and causation," *Computational Intelligence*, 5(3):142-150, 1989.

[14] H. Li, R. Sahita, G. Kime, J. Noel, and S. Yadav, "Policy Enabled Network Security with Adaptive Feedback Loop and Capability-Based Data Mode," *EURESCOM*, October, 2003.

[15] D. Verma, *Policy-Based Networking, Architecture and Algorithms*, *New Riders Publishing,* 1st edition, November 2000.

[16] G. Stone, B. Lundy, and G. Xie, "Network Policy Languages–a survey and a new approach," *US DOD, IEEE Network*, January/February 2001.

[17] P. Schacter, "Distributed Security and the Disappearing Perimeter," *Burton Group Network Strategy Report*, V1, 23 APR 2002.

[18] Intel® Active Management Technology white paper, http://www.intel.com/go/iamt/

AUTHORS' BIOGRAPHIES

John Mark Agosta is part of the Intel Research DDI network intrusion detection project. His work involves developing probabilistic models for intelligent diagnosis and management. Previously he worked on automated response to customer inquiries for Customer Relationship Management (CRM) software while working at Edify Corporation. From 1998-2000 he was Chief Technical Officer (CTO) for Knowledge Industries, where he built Bayes networks for medical, avionics, and automobile clients. From 1992 to 1998 he worked as a research engineer at SRI International. He built models for electric utility generator alarm filtering and computer network intrusion detection. He also worked in automated planning for emergency response and USAF air campaign planning. Agosta received his Ph.D. degree in the Engineering-Economic Systems Department (now Management Science and Engineering) of Stanford University in 1991. His thesis topic was on an application of Bayes networks to visual recognition. His e-mail is john.m.agosta at intel.com.

Jaideep Chandrashekar received a B.E. degree from Bangalore University, India, in 1997, and a Ph.D. in Computer Science from the University of Minnesota in January 2006. He has been with Intel Research/CTL since then. His research interests include computer networks and distributed systems, especially Internet technologies, network routing, and computer security. He currently serves on the program committee for Infocom 2007. His e-mail is jaideep.chandrashekar at intel.com.

Denver H. Dash received his B.S. degree in Physics from Case Western Reserve University in Cleveland and his M.S. degree in Physics from the University of Pittsburgh. He received his Ph.D. degree in Intelligent Systems from the University of Pittsburgh in May 2003, specializing in causal models and Bayesian methods in machine learning. Prior to joining Intel Research he was a postdoctoral research fellow at the RODS lab at the Center for Bioinformatics at the University of Pittsburgh. There he worked on biosurveillance and outbreak detection using probabilistic graphical models. He joined Intel Research in September of 2003, where he has since received a CTG Divisional award for applying machine learning techniques to reduce sort-test costs. He serves on the program committees of UAI, AAAI, ICML, and he conducts reviews for most major international conferences/journals in Artificial Intelligence and Machine Learning. His e-mail is denver.h.dash at intel.com.

Manish Dave is a staff network engineer with Intel's Information Technology Group. He is lead engineer and designer for the Internet connectivity and external network connectivity for Intel. He has over ten years of network engineering experience and network security experience. His e-mail is manish.dave at intel.com.

David Durham joined Intel in 1995 and is currently a principal engineer in Intel's Corporate Technology Group. He has a passion for research into protecting computers from viruses and network-based attacks. He manages the research group responsible for developing new network security capabilities that are going into Intel's platforms. Since joining Intel, David was responsible for developing policy-based network management standards, traffic engineering products, and for creating platform-based network security solutions. He is the author of a book entitled *Inside the Internet's Resource Reservation Protocol: Foundations for Quality of Service* published by

John Wiley and Sons Inc.; he is the co-author of several Internet standards-track RFCs, and he has represented Intel externally in various standards bodies at the working group chair level. David has 50 patent applications pending and has been issued 6 patents. He holds B.S. and M.S. degrees in Computer Engineering from Rensselaer Polytechnic Institute. His e-mail is david.durham at intel.com.

Hormuzd Khosravi is a senior network software engineer in the Communication Technology Lab at Intel. His current focus is on integration of Intel's platform security and manageability technologies with different network architectures and standards for improving enterprise security. He joined Intel in 1999, has worked on several networking projects including Control Plane Platform Development Kit (CP PDK) that shipped as part of the Intel IXA SDK product. He was involved in defining and developing industry standards for modular communication platforms in forums such as the IETF ForCES working group and Network Processing Forum. Hormuzd received his M.S. degree in Computer Engineering from Rutgers University. His e-mail is hormuzd.m.khosravi at intel.com.

Hong Li is a senior researcher with the IT Research team of Intel's Information Technology Group, leading research in the area of trustworthy and survivable systems. She also led the development of several IT security strategies and architectures. She is a 2004-2005 Santa Fe Institute Business Network Fellow. Prior to Intel, Hong worked for Science Applications International Corporation (SAIC) and ECUTEL, and before starting her industry career, Hong was a research associate at the U.S. Naval Research Laboratory. Hong holds a Ph.D. degree from Penn State University and a B.S. degree from Xi'an Jiaotong University, China, both in Electrical Engineering. Her e-mail is hong.c.li at intel.com.

Stacy Purcell graduated from the Georgia Institute of Technology with a BS-CS degree in 1992. He joined Intel Corporation in Folsom, CA immediately after graduating where he has been employed in several roles including system administrator, network engineer, and manager over the course of the last 11 years. His e-mail is stacy.p.purcell at intel.com.

Sanjay Rungta is a principal engineer with Intel's Information Technology group. He received his B.S.E.E. degree from Western New England College and his M.S. degree from Purdue University in 1991 and 1993, respectively. He is the lead architect and designer for the Local Area Network for Intel. He has over 13 years of network engineering experience with three years of experience in Internet Web hosting. He holds one United States patent and four pending in the area of Network Engineering. His e-mail is sanjay.rungta at intel.com.

Ravi Sahita is a senior network software engineer in the Communications Technology Lab at Intel. His primary focus is on platform and network security/manageability. He is interested in platform approaches to address network security issues, such as software integrity and other counter measures against network-based attacks. Ravi has contributed to industry standards for network management and to the Intel NetStructure® Policy Manager and Common Open Policy Services (COPS) SDK products. He is also a contributing member of the Internet Engineering Task Force (IETF) and the Trusted Computing Group (TCG). Ravi received a B.E. degree in Computer Engineering from the University of Bombay, and an M.S. degree in Computer Science from Iowa State University. His e-mail is ravi.sahita at intel.com.

Uday Savagaonkar received an MTech degree from the Indian Institute of Technology, Mumbai, India in July 1998 and a Ph.D degree from Purdue University, West Lafayette, IN, in December 2002. Currently he is with the Communications Technology Lab at Intel Corporation, Hillsboro, OR. His research interests include network pricing, Markov games, networked platform security, and platform virtualization. His e-mail is uday.r.savagaonkar at intel.com.

Eve M. Schooler is a senior research scientist in the Corporate Technology Group (CTG) and the project lead for the Distributed Detection and Inference (DDI) research project, which focuses on distributed network anomaly detection in large-scale enterprise networks. Eve obtained a B.S. degree from Yale University, an M.S. degree from UCLA and a Ph.D degree from Caltech, all in Computer Science. Prior to Intel, she held positions at Apollo Computers, Information Sciences Institute (ISI) and AT&T Labs-Research. Her e-mail is eve.m.schooler at intel.com.

† Intel® Active Management Technology requires the computer to have additional hardware and software, connection with a power source, and a network connection. Check with your PC manufacturer for details.

Δ Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM) and, for some uses, certain platform software enabled for it. Functionality, performance or other benefits will vary depending on hardware and software configurations and may require a BIOS update. Software applications may not be compatible with all operating systems. Please check with your application vendor.

THIS PAGE INTENTIONALLY LEFT BLANK

Machine Learning for Adaptive Power Management

Georgios Theocharous, Corporate Technology Group, Intel Corporation
Shie Mannor, Department of Electrical and Computer Engineering, McGill University
Nilesh Shah, Corporate Technology Group, Intel Corporation
Prashant Gandhi, Corporate Technology Group, Intel Corporation
Branislav Kveton, Department of Computer Science, University of Pittsburgh
Sajid Siddiqi, School of Computer Science, Carnegie Mellon University
Chih-Han Yu, Department of Computer Science, Harvard University

Index words: power management, machine learning

ABSTRACT

Adaptive Power Management (APM) systems for laptops decide when to place a component into various power-saving states given the user activity. The long-term goal of an APM system is to maximize the battery life while minimizing the annoyance to the user. The state-of-the-art in commercial solutions is timeout policies. These policies switch components into their low power states based on thresholds of periods of inactivity. Unfortunately, such methods not only waste power during the periods of inactivity, but also needlessly annoy the user when they turn off components at inappropriate times. Research in APM, on the other hand, has focused more on modeling system dynamics and not on usage patterns. We propose a system that learns when to turn off components based on different user patterns. We describe the challenges of building such a system and progressively explore a range of solutions. We experiment with a direct approach that predicts when to turn off a component given usage features (e.g., historical keyboard and mouse activity, active application, history of network traffic, and history of CPU utilization). To improve performance of the direct approach we partition data based on the context and train the learning algorithms separately for each context. Context could be past idleness or any partitioning of the data that improves performance. We then propose a model-based approach that captures the temporal dynamic of system and user state, the cost of power and user annoyance, as well as the effect of power-saving actions on the user and system. Our direct approach is validated on a large data corpus collected from multiple real users where results show a considerable improvement over traditional timeout methods.

INTRODUCTION

Adaptive Power Management (APM) attempts to increase the perceived battery life of a laptop by turning off certain components when their services are not going to be needed. Of course, it is not known in advance which services will be needed and when. Moreover, the perceived impact of having certain services not available is much greater than the perceived impact of others. The goal of APM is to extend the battery life as much as possible with an acceptable perceived loss of performance. APM, as opposed to non-adaptive power management, attempts to make better decisions with time and to adapt to individual users.

APM is an example of an autonomic system. The autonomics concept is derived from the human body's autonomic nervous system that regulates individual organ function, and for the most part is not subject to voluntary control. An autonomic computing system controls the functioning of computer applications or systems without explicit user input. The goal of autonomic computing is to create systems that are self-managing, self-healing, and self-protecting. Autonomic computing promises to reduce expenditures associated with operations, maintenance, and support, and to significantly improve the end user's experience.

Autonomic systems ensure smooth and uninterrupted operation by monitoring the system state, diagnosing problems, identifying user context, and executing actions to fix the problems. For example, an APM system monitors the user and laptop state, diagnoses the user mood or context, and changes power states such that the user is not noticeably impacted. Autonomic systems need to naturally reason about uncertainty in both perception

and action. In an APM system, perception uncertainty arises from the fact that the user context cannot be directly observed from sensors, such as keyboard and mouse activity, or the currently active application. Actions that turn off components also generate uncertainty. They create uncertainty in terms of the time it takes to turn a component on and off and in terms of the effect it would have on the user's context. For example, placing the machine on standby does not always take the same amount of time and sometimes the Operating System (OS) does not even allow it. If the user context is for example a measure of idleness, then going to standby, could either increase idleness if it succeeds or decrease it if it fails, hence the uncertainty. If the user context is a measure of the users' mode of operation (e.g., working, browsing for fun) then going to standby could again change the context in an unpredictable manner.

Modeling real-world phenomena as stochastic processes and sequential decision-making under uncertainty (see Figure 1) are widely used paradigms in artificial intelligence. The process of estimating these stochastic process models and the computation of optimal policies fall into the category of machine learning. Machine learning refers to the problem of constructing computer programs that automatically improve with experience. The simplest class of machine learning algorithms are *classifiers,* which are functions that learn (from examples) to map input patterns to output classes. For the APM domain, our algorithms learn to map laptop usage patterns (at the OS level) into power management actions as shown in Figure 2.

In this paper, we first summarize past approaches to the APM problem and then describe a machine-learning approach that maps user patterns to power-saving sections. Our approach outperforms current commercial solutions such as timeout policies. We then propose a more general model-based approach that relies on building stochastic process models. We finally discuss the potential application of such models to other domains.

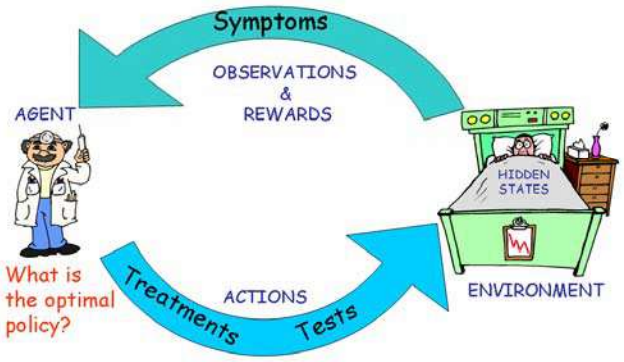

Figure 1: A classic sequential decision-making task under uncertainty. The doctor performs treatments and diagnostic tests on the patient, which change in a stochastic manner the internal state of the patient. The doctor then observes symptoms. The optimal policy is one where in the long run the patient is made well.

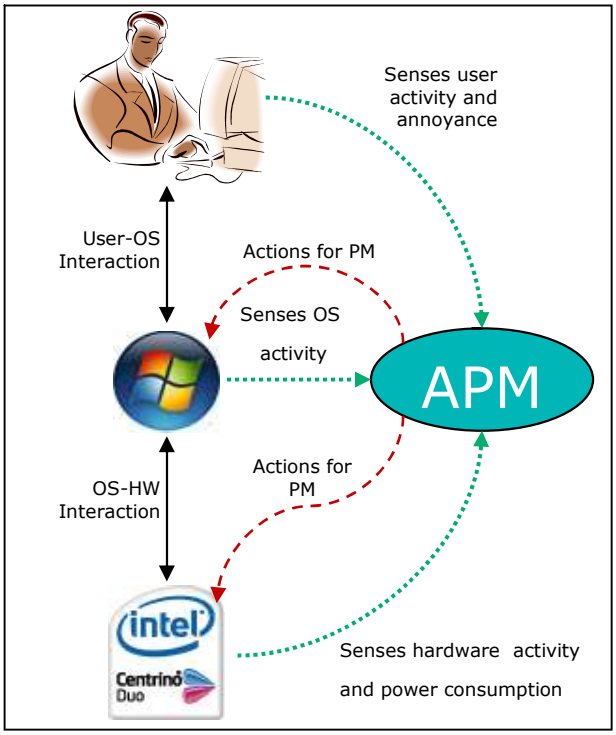

Figure 2: APM takes as input user and system activity and issues power management actions to the OS and to the hardware

ADAPTIVE POWER MANAGEMENT

An APM system maps the computer system (OS and hardware) activity and the user activity into power management actions as shown in Figure 2. The goal of such a system is to maximize battery life while minimizing the impact on the user. In this section, we describe the state-of-the-art in APM solutions and our proposition for explicit consideration of the user's context and perception.

Past Approaches

Currently used power management techniques are based on *timeout policies*. These policies shut down a component after it is not used for some predefined time. The aggressiveness of the scheme is reflected by the length of the timeout before a specific component is shut down. For example, Windows* OS has several built-in power management schemes that allow the user to choose between different levels of aggressiveness. The user is expected to switch manually between these schemes. Timeout policies are fairly simple and robust. They are, however, non-adaptive and may be too fast or too slow to react.

Applying machine-learning techniques to the APM problem is in its infancy. In [16] a simple predictive approach for deciding if to turn a component on or off was investigated. The authors attempt to predict the length of the idle period and claim that typically a short idle time is followed by a long active time and vice-versa (i.e., the prediction is based on the recent history). The decision rule eventually recommended that the component be shut down if it was used for less than some threshold value. They explored two approaches: a threshold value that is determined using regression and a threshold value that is manually obtained from data. In both approaches, the threshold parameters were obtained based on data in a non-adaptive manner. The approach of [7] is to predict the future delay as an exponentially weighted sum of recent delays. Our work uses learning techniques that focus more on the adaptive and dynamic aspects of power management.

Several papers considered policy optimization in the context of power management ([1], [14], [15]). According to this approach, a single Markov Decision Process (MDP) or a Semi-Markov Decision Process (SMDP) is constructed and solved using linear programming. An MDP is a stochastic process model of the state of the world and costs of actions (see Figure 6). An SMDP is an MDP where the next state does not only depend on the current state but also on how long the current state has been active. The Markov property, which states that the current state is sufficient for predicting the next state, is violated, hence the semi-Markov terminology. The Markov models used in these works are extremely simple and their state space is mostly an active/not active indication. State transitions are estimated from data and the optimization is done off-line. A somewhat more complex stochastic optimization model is presented in [12]. The model used in this work is a non-stationary process where there are several modes. For each mode, the process is a Markov decision process. In general, these models are not realistic because they assume that the state variables can be directly observed from sensors. Our work differs from this line of work in that we explicitly take into consideration the user activity (usually not directly observable) and also consider the perceived performance.

User-Based APM

The papers mentioned above attempt to use dynamic models estimated from data. To the best of our knowledge, they are all based on relatively small and simulated amounts of data. Most importantly, these works do not model user state in their decision making, which we believe to be essential.

Monitoring user activity (i.e., context) from sensor observations is a well-studied problem, e.g., see [6]. However, there are relatively few instances where activity monitoring is linked to decision making, which is the goal of APM. In this paper, we focus on making the actual decision: which component to turn off and when. With this goal in mind, we introduce a novel representation of context, namely the idleness duration variable, to describe the state of the user. Though simple, this is an effective way to characterize context for this decision problem in the power management domain. We also address the issue of how the user perceives the performance degradation (a measure we call *annoyance*).

Reducing the power consumption to the bare minimum can be easily done by moving to standby mode on every occasion. This, however, is not very useful since a power management system applying such a policy will not be very usable. It is clear that a "good" policy should minimize power consumption. Therefore, one has to consider the tradeoff between the power savings and the perceived performance degradation, i.e., the annoyance. In order to examine this tradeoff, we look at tradeoff-type plots, where one axis shows the power saving and the other shows the level of annoyance. With no power savings, all the components are on all the time and the annoyance is assumed to equal zero. As the annoyance increases, the power savings may increase. Still, the power savings will never be more than the minimal power consumption level needed to perform the required operations. Figure 3 demonstrates a power-saving annoyance curve. Different users may choose different points on the power-saving annoyance curve according to their desired tradeoff.

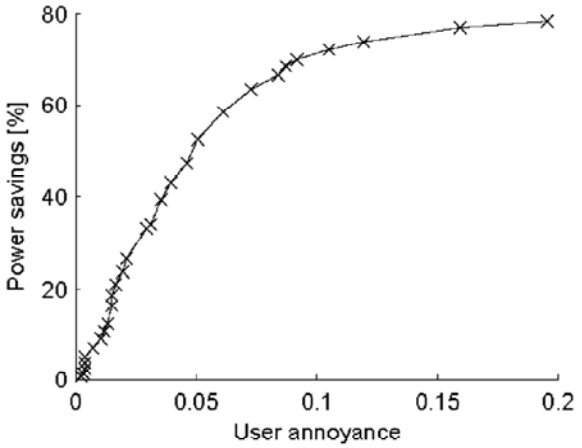

Figure 3: A tradeoff curve for power savings versus annoyance. The power savings cannot pass a certain threshold as shown with the converging curve to be approximately 80%. This curve was produced from a real trace, where the points on the curve represent various timeout policies. The furthest top right point represents a timeout parameter of 1 second and the furthest lower left, a timeout parameter of 600 seconds.

Quantifying the power savings is straightforward and it can be done in Watts per second, or the proportion of time a component is off. In our experiments, we considered four components: turning off the LCD, turning off the WLAN, running the CPU in low frequency, and placing the laptop in standby mode. When the laptop is turned on or in standby mode, it consumes 18.5 and 0.7 Watts per second, respectively. Turning off the LCD, or the WLAN, or switching the CPU into its low-frequency mode, results in the consumption of 14, 17.5, and 15 Watts per second, respectively.

Quantifying the annoyance is trickier. Essentially, the problem is that annoyance is a subjective measure and different users may feel a different level of annoyance for the same sequence of actions. Based on interviews with users, we decided to measure the annoyance as follows. If the system decided by mistake to turn off the CPU, WLAN, and LCD, it leads to annoyance of 1, 3, and 7, respectively. Moving to a standby mode by mistake leads to annoyance of 10. We know if turning off a component was a mistake since we can detect if this component is needed or not after it was turned off. For example, if we turn the LCD off and after a few seconds the user opens a new application and clicks the mouse we can safely assume that turning the LCD off was a mistake. The different annoyance coefficients represent a collective and subjective estimate of how much more annoying is turning off one component compared to turning off another.

THE DIRECT APPROACH

In our direct policy approach, we trained a separate classifier for each action. The input is the sensor measurements and features, and the output is whether to turn the component on or off.

The sensors that were sampled every second were active application, keyboard and mouse activity, CPU load, and network traffic. Out of these sensors we constructed various features, such as sums, decayed sums, averages, and gradients of sensor readings over different past windows.

The labels for the classification problems were generated by heuristic laws after going through the trace of sensor readings. We used several ad-hoc rules to determine the labels, such as, if turning WLAN off and packets are sent within 30 seconds, determines the action to turn WLAN off to be a mistake. This labeling scheme gives a "target" policy that is guaranteed not to annoy the user because it will never turn off components when needed in the next fixed look-ahead.

As a yardstick for a learning algorithm, we used logistic regression, k-nearest-neighbors and the C4.5 decision tree. These algorithms are simple to use and produced more or less similar results. Since these algorithms are standard, we do not describe them here. We refer the reader to [5]. Many classifiers (such as logistic regression) have some threshold function. For example, if the outcome is probabilistic between the two classes, then the threshold is the minimum probability for deciding the first class. By varying the threshold, we can make the resulting classifier more or less aggressive leading to different points on the annoyance-power-saving curve.

The second yardstick we used is timeout-based policies. The decision to turn off a component in such a policy is according to the time passed since it was last used–we turn it off if it was not used for more than some predefined threshold. We can control the level of annoyance by varying the threshold (the larger the threshold the less aggressive the timeout policy and the less annoyance we will observe).

Context-Based Policy Learning

For context-based policy learning, we explicitly take into consideration the user context variable. The idea is to separate the data for each value of the context variable and then train a separate classifier for each data partition. We then choose the decision threshold for each classifier such that the overall power savings are maximized. As we show in our experiments, this approach performs better than naïve classifiers trained on all data.

In general, user context is not directly observable and represents the user's internal state, mood, or intentions. In our experiments we discovered that the time since the component was last active is a good feature for predicting the actions. Therefore, we defined context to be the time since the component was last active. We partitioned these durations into 30 categories, where each category represents a value for the context variable. Value 1 means in the previous time step the component was active, and value 30 means the component was idle for the last 600 steps. We chose the rest of the 28 thresholds heuristically across the 600 seconds. We observed that the longer a component is idle the more likely it will be idle (which means it should be easier to predict the action); therefore, for larger idle times we made the partition density coarser than that for shorter idle times. Empirically, the number 30 gave us reasonable performance. For smaller numbers the predictions were not as good, and for larger numbers we were over fitting.

In order to determine the thresholds for each context classifier we used an optimization algorithm. The algorithm starts by setting the least annoying thresholds for all classifiers. At each step, we increase the threshold that corresponds to the maximal power savings over annoyance increase ratio subject to a global annoyance constraint. Identifying the threshold and its new value is computationally simple because we only enumerate a single parameter and freeze the remaining ones. This optimization method proved efficient and it always converged to a local optimum. Figure 4 shows a graphical representation of our algorithm.

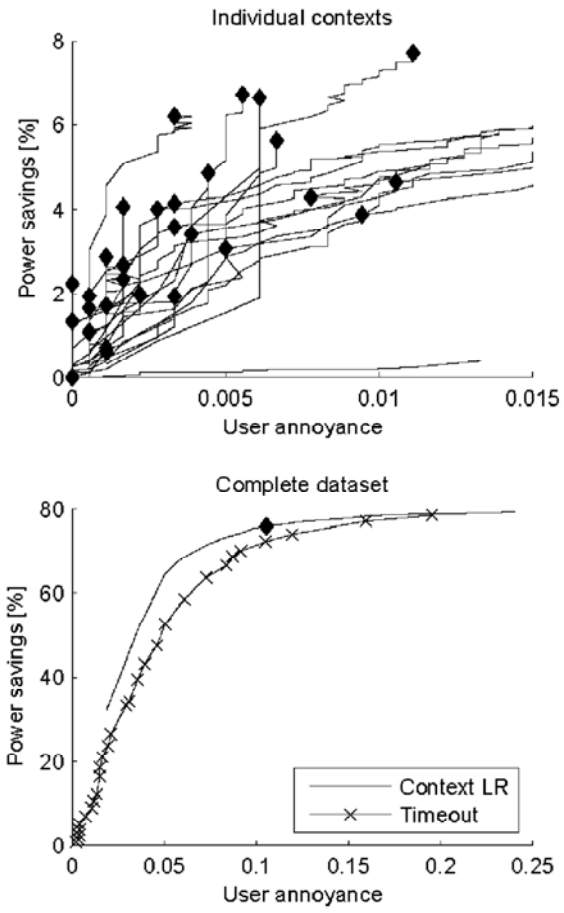

Figure 4: The top graph shows power savings/user annoyance curves corresponding to the 30 individual contexts. The black diamonds represent the decision points returned by the heuristic optimization method subject to the annoyance constraint of 0.1. The bottom graph compares power savings/user annoyance curves for the optimized mixture classifier (black line) and timeout policies (black line with cross markers). The black diamond denotes the global annoyance constraint for 0.1.

Experiments

We compare the quality APM policies with naïve classifiers and with timeout policies. The different policies were evaluated on 42 traces, which were collected for 7 users, and they represent the cumulative experience of 210 usage hours. The data are obtained from T42 laptops by using an application that we developed. We present results for standby policies only. These results generalize to the power management of the LCD, CPU, and WLAN. In order to make a fair comparison, we compare the

performance of the various policies for the same annoyance level.

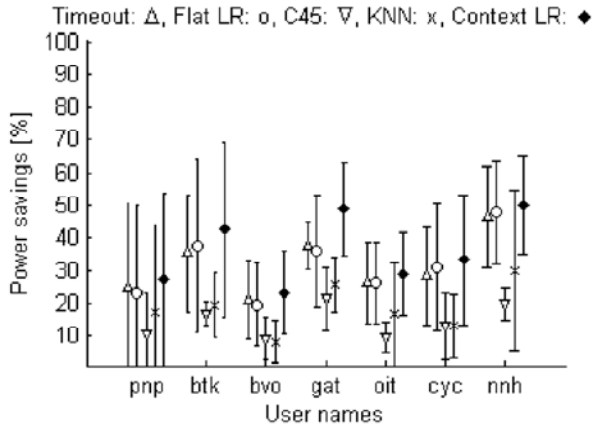

Figure 5: The graph shows expected power savings and their standard deviations for various classification techniques. The target annoyance level is 0.05.

Based on Figure 5, we know that the context-based LR outperforms the baselines methods. This observation justifies the benefits of training classifiers for individual contexts, which are constructed based on our domain knowledge. An obvious question that arises is why we save more power for some users than for others. This phenomenon can be explained from the distribution of the idle times for each user. Simply, if the user spends more time in longer inactivity periods, we save more power. This demonstrates the difference between users. Different users may have a very different behavior and the APM policy has to adapt to the specific user.

DISCUSSION

So far we have experimented and described a direct approach to the problem of APM. We believe that significant performance improvements might be possible by using stochastic process models of the domain. These models capture the temporal dynamics of user and system state as well as the annoyance and power costs. Having learning and planning algorithms could potentially be beneficial to other domains as we describe at the end of this section.

Model-Based Approach

The model-based approach decouples the decision-making process from the problem of learning and estimating the model of the environment. Learning a model refers to the process of defining/discovering domain variables and how they relate to each other. Using the model refers to the process of computing decisions given the model. Next we describe three models of increasing expressiveness as well as increasing difficulty in learning them and using them.

Markov Decision Processes

The MDP model facilitates reasoning in domains where actions change the states stochastically and where there is usually a delayed reward signal (e.g., winning in backgammon is attributed to possibly many moves along the play and not only to the final move). Figure 6 gives a graphical MDP representation.

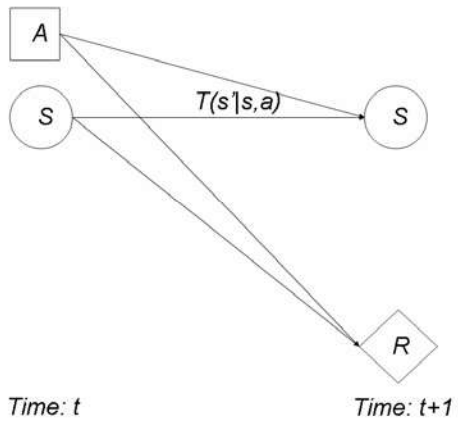

Figure 6: This is a graphical representation of an MDP model. The left column represents what happens at time t and the right column what happens at t+1. Arrows represent dependence.

In an MDP, there exists an underlying state that encompasses all the information needed to represent the world. This state is a summary of the relevant systems and models. Given the state of the system the future is independent of the past. This means that the true state captures all the information needed to describe the system. This last property is often referred to as the *Markov property*. The agent can act in response to the observed state. The result of the action depends on the state of the world and may be uncertain.

The objective of the agent is to maximize his long-term cumulative reward (typically, the reward of interest is the discounted future reward where the near future is more important than the more remote future). An immediate reward is received when exercising an action. This immediate reward depends on the action made by the agent and the true state. The immediate reward that is received in a particular state following a certain action may also be random.

Formally, MDPs are described as a 4-tuple: $<S,A,T,R>$ as shown in Figure 6. S denotes a finite set of states and A denotes a finite set of actions. $T(s'|s,a)$ denotes the transition probability from state s to state s' under action a. R(s,a) is the reward for taking action a in state s. In Figure 6 we see the dependencies between the different

elements that define the MDP: the current state (S) and action (A) determine the next state (S') according to the transition probability (T); the current state (S) and action (A) determine the reward (R).

The main advantage of the MDP model is its simplicity. On the one hand, learning the parameters and computing an optimal policy can be done efficiently by several different algorithms (e.g., [2]). On the other hand, the simplicity of the MDP model comes with a price: it cannot capture unobservable dynamics, and its success hinges on the assumption that the state of the system can be estimated with no errors.

Dynamic Bayesian Networks

A Dynamic Bayesian Network (DBN) describes a stochastic process where some of the variables are observed and some are not. Figure 7 shows a graphical representation of a special case of DBNs where there is a single variable called a Hidden Markov Model (HMM). In a DBN the agent reasons about the state of the process indirectly though the observed variables. For example, the doctor in Figure 1 makes an inference about the internal state of the patients through symptoms.

An HMM is described as a 4-tuple: $<S,T,Z,O>$ as shown in Figure 7. S denotes a finite set of states, and Z denotes a finite set of observations. T(s'|s) denotes the transition probability from state s to state s'. O(z|s) is the probability of observing the observation z in state s. In the figure we see the dependencies between the different elements that define the HMM: the current state (S) determines the next state (S') according to the transition probability (T); the observation (Z) depends on the current state (S) through the observation probability (O).

The advantage of DBNs is their ability to capture complex dynamics that are not completely observable. The disadvantage of DBNs is their lack of support for decision making. Estimating the DBN parameters can be done relatively efficiently using algorithms such as the Expectation-Maximization algorithm.

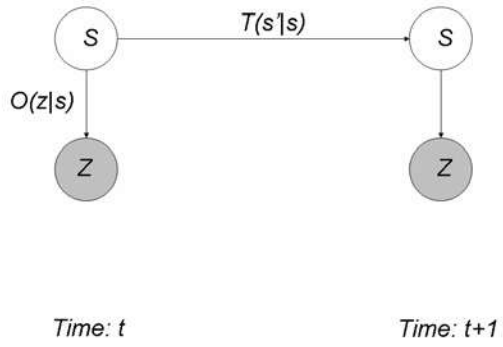

Figure 7: This is a graphical representation of a simple dynamic Bayesian networks called a Hidden Markov Model

POMDPs

The Partially Observable Markov Decision Process (POMDP) model is the combination of HMMs/ DBNs and MDPs. The idea behind the POMDP model is to combine the strengths of HMMs (capturing dynamics that depend on unobserved states) and MDPs (taking the decision aspect into account) without significantly sacrificing computational efficiency. The model is depicted in Figure 8, where it can be seen that the POMDP contains elements from both MDPs and HMMs. POMDPs are a natural conceptual framework for modeling sequential decision tasks as illustrated in Figure 1. A POMDP model describes sequential decision tasks under uncertainty. A control policy in a POMDP computes an action after every observation such that in the long-run (discounted or average) the expected utility is maximized. Uncertainty, therefore, has two sources in POMDPs. First, the true state of the world is usually unknown (the doctor does not know what the patient has exactly). Second, even if the state is known, some actions may have uncertain consequences (different treatments may work differently for patients with the same condition).

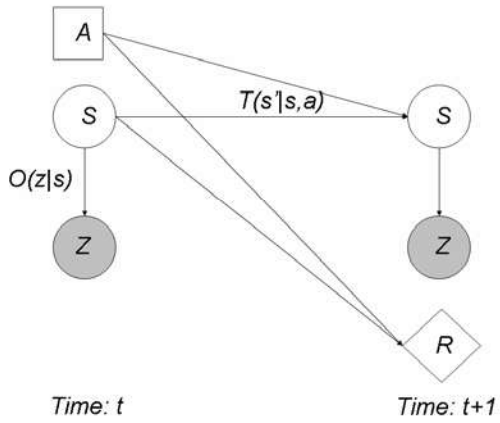

Figure 8: This is a graphical representation of a POMDP model. It is created by combining the MDP and HMM models.

A POMDP policy computes actions at every time step. Due to the fact that the system state is not observed, the POMDP policy maps actions to all possible probability distributions over the states, otherwise called *belief states B*. The belief state *b(s)* represents the agent's current belief that the true state is *s*. The goal of the policy is to maximize the long-term reward. The POMDP policy can be computed by solving the Bellman Equation [2]. Figure 9 illustrates the POMDP execution system. For the sake of simplicity we do not discuss how belief update is done or how the Bellman Equation is solved. A good introduction to POMDPs can be found in [8].

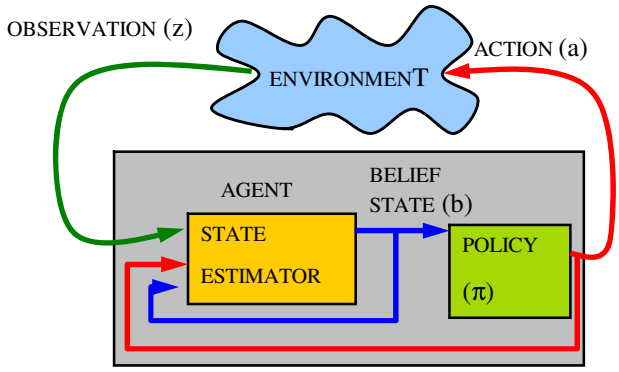

Figure 9: This figure represents the POMDP execution system. The state estimator module takes the previous belief states, the action taken and observation received, and produces a new belief state. The Policy module takes as input the belief states and outputs an action.

Some of the strengths and weaknesses when using the POMDP model are these:

- *Computing the policy*. In general, computing the optimal policy is intractable [11],[10]. Still, reasonable approximations can be made by following some simple heuristics. There are many algorithms for finding an adequate policy that may not necessarily be optimal.
- *Controlling the level of abstraction*. The POMDP model allows choosing different levels of abstraction leading to models that have more or less states depending on the POMDP designer preferences. This leads to a tradeoff between the amount of information that has to be supplied on the model (a large number of states would require a lot of information on the model) and the potential accuracy of the model (a large number of states would enable a higher accuracy).
- *Reasoning in terms of information*. Reasoning in terms of information allows the designer to distinguish between actions that maximize utility and actions that provide information on the state.
- *Handling model uncertainty*. Lack of knowledge regarding the specifics of a model can be naturally incorporated into the POMDP model. Similarly to DBNs, the model parameters can often be learned from data, using algorithms such as the Expectation-Maximization algorithm. This allows for the building of a rough initial guess to start with and refining it later, based on the observed data.

The Model-Based Approach to Adaptive Power Management

Among the different models presented above, the POMDP model is the only one that is rich enough to capture the two main aspects of APM. First, the world model includes a human user and a complex computer system that cannot be assumed to be perfectly observable. Second, APM management involves making decisions on which components to turn on and off.

There are many possible ways to define a POMDP for APM. The suggested POMDP model is shown in Figure 10. Casting this model in the language of POMDP we have this: the actions (A) are to turn on/off (for each component); the state space (S) is a combination of system state and user state; the possible observations are the various sensors/features we described before; the transition probability (T) and the observation probability (O) are both unknown a-priori and must be learned from data.

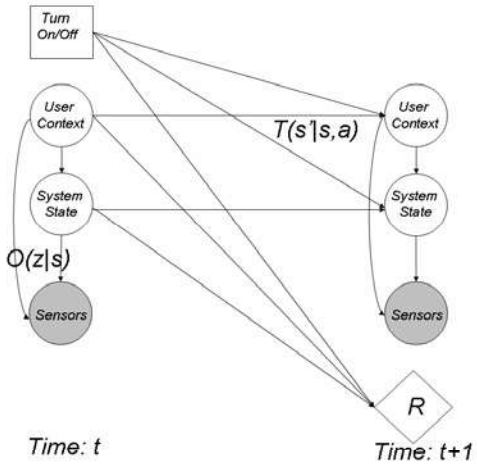

Figure 10: Our model has user, system, and sensor variables. The user variables are not directly observable and represent user context or activity. The system state indicates whether the component is idle or active. The sensor variables are keyboard and mouse activity, application running, CPU load, and network traffic. The reward function R represents power cost as well as cost incurred by annoying the user.

The main problem in constructing the POMDP for APM is how to construct the state space. This is a thorny problem at the core of applying POMDP methodology to any real-world problem. It is obvious that a model that fully describes the system or the user context (i.e., the way the user is interacting with the system) would be too complex to describe, and would certainly not lead to any useful computations. We therefore have to balance the complexity of such a model with our need to be able to obtain a useful APM policy.

We emphasize that learning a good POMDP model in general involves answering questions such as "what is the user's context," "how does that context relate to system state," "how does context change over time," and "what is the level of user annoyance." These questions are not trivial in the field of artificial intelligence. Still, we believe we can obtain partial answers to these questions based, in part, on heuristics or even arbitrary decisions. Once we have such answers we can offer a solution to the POMDP.

Our ongoing research focuses on finding an adequate model to represent a realistic system and then solving that model. We are currently studying several aspects of the model. First, we are developing a more complex model for user context (other than past idleness). In particular, we look at automatic context construction (that is generated by automatically clustering sequences). Second, we are looking at different ways to measure annoyance. Instead of using an annoyance measure that is the same for all users, we consider learning the annoyance from the user, based on individual feedback, and even prompt the user for explicit feedback. Third, we are studying the statistics of the duration between changes of the values of the system and user variables. Fourth, we consider the initial period where the system is initializing. The problem here is that when the system starts running the amount of information that is available is small and the decisions are potentially sub-optimal. In this initial stage there might be a lot of value in exploring actions that can provide a lot of information on the user. The problem of balancing between information gathering and using the information to perform optimally is known as the exploration-exploitation dilemma. As was shown in [13], model-based approaches may be unsatisfactory in their initialization phase.

Other Domains

There are many real-world problems that can be framed as sequential decision-making tasks under uncertainty. Stochastic models such as POMDPs offer a good framework for such tasks because they encompass the idea that there are underlying hidden states (user and system) and the idea that the system needs to constantly take actions under uncertainty to satisfy some long-term goals and to maximize some performance criterion.

Explicit reasoning about the user state is imperative in many systems that have significant interaction with the user. For example, in the power management domain, if a user does not mind interruptions we can afford to have more aggressive and less accurate power management policies, where accuracy is in terms of predicting whether a component can be turned off. The user state could be summarized by variables that are related to user activity (physical and mental) and utility preferences. Learning mental user models is not an inconceivable idea and has previously been explored in opponent modeling in games such as in [3] and [4].

The idea of user adaptive autonomic systems such as the APM systems could be extended to domains that go beyond platforms. Instead of an isolated computer platform we could call it an "environment" which has both user states and task states. The environment produces observations and utilities. The autonomic agents' task is to maximize their long-term utility as explained in Figure 1. Figure 11 and Figure 12 show some examples of similar domains previously studied by researchers. In each figure, we describe the domain and how we would frame it as a POMDP problem.

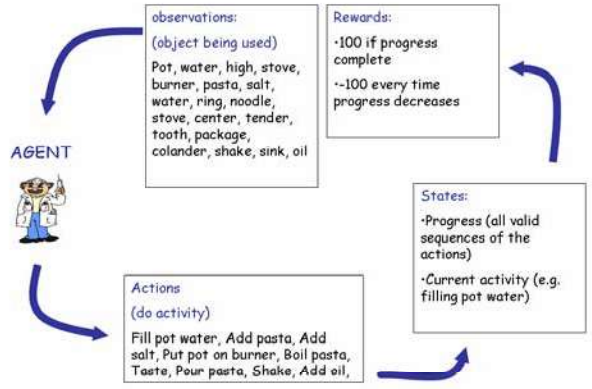

Figure 11: Example of a prompting scenario where an elder person boils pasta. The system monitors the person, diagnoses his/her state and prompts him/her appropriately to complete the task. This is a non-platform autonomic system where the state variables capture task and user states. This problem was studied by the authors in collaboration with the Intel Lablet in Seattle.

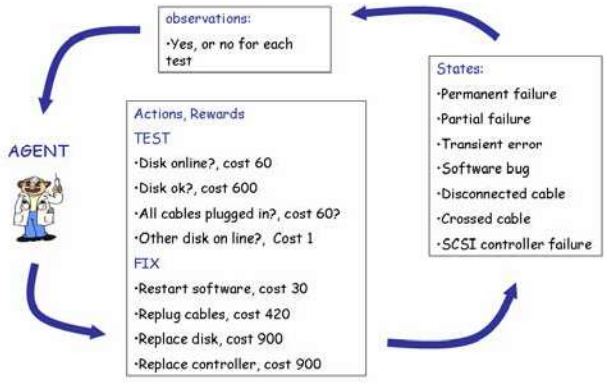

Figure 12: Example of an autonomic system for diagnosing a failed disk on a server and repairing it. It was modeled as a POMDP in [9]. This is a classic platform autonomic system where the user is not explicitly considered.

There are many other similar problems of current interest to Intel that can naturally fit the model-based decision-making framework. Among these are the following:

- *Identity capable platforms*. In this domain the idea is to develop trusted access across a variety of devices, networks, and services. This can be achieved by identity-based policies where identity could be the user itself, or some mode of operation. This is obviously a system that needs to explicitly reason about the user in order to achieve some long-term goals, such as ensuring a trusted network or certain services.
- *Server power management*. In this domain power management has to be done over an entire server farm. The system state is captured by sensors, which monitor the work load, the temperature, and the different processes that run on each server. The actions manage the system by assigning workloads dynamically to servers. The reward function defines the objective of the system by providing a positive reward when the power (and heat) distributions are as balanced as possible and jobs get executed without delays; otherwise, the reward is negative.
- *Virus detections on networks*. In this domain an agent has to predict whether or not there is a worm attack based on local network traffic patterns. The state here represents the network and traffic patterns. There are no actions here but rather the agent has to decide whether or not the network is currently under attack. The reward is positive when attacks are correctly identified and negative otherwise.
- *Memory power management*. In this domain, the agent needs to predict whether blocks of RAM can be turned off without impacting the performance that is perceived by the user. The system and user state here are similar to the APM domain. Actions are whether to turn on or off blocks of RAM. The reward function represents user's annoyance and power costs similar to the APM problem.
- *Awareness CPU architecture*. In this domain the agent needs to forecast the next upcoming CPU state based on usage patterns. This is again a very similar problem to the OS power management problem except that it needs to be done at the millisecond level.
- *Multi-device transparent power sharing*. In this domain machines need to automatically share power. Given potential usage, benefit, and activity of each device, power should be divided according to how the users' tasks are accomplished and should not take into account individual device power levels.

CONCLUSION

In this paper, we described how a machine-learning methodology could be applied to APM. Despite the simplifying assumptions we made about observability and temporal dynamics, the performance of our approach is better than naïve classification techniques and commercially available methods. Our success is partially due to the fact that we modeled the user context, which is

a fundamental concept that arises in any autonomic domain. We are currently working on more complex models and on methods for automatic discovery of context and learning of temporal dynamics.

We believe that stochastic models such as MDPs, DBNs, and POMDPs are promising techniques for mitigating the complexity and handling uncertainty for autonomic systems. The POMDP model in particular facilitates reasoning about uncertainty and information; it allows the application designer to control the level of abstraction needed and to choose the appropriate tradeoff between model accuracy and feasibility. Yet, the POMDP model can be solved, at least sub-optimally, using relatively simple methods. Thus, the main challenge in applying the POMDP model in autonomics is not computational; rather it is a modeling challenge, i.e., creating models that are simple yet effective for the application at hand. We are currently working on using stochastic models, and POMDPs in particular, for APM.

Handling uncertainty and hidden information is at the core of many autonomic systems. Whether the designer of the autonomic system chooses a direct approach or whether a probabilistic model is preferred, the explicit consideration of the user is crucial for the success of many such systems. We believe that many autonomic systems of interest would benefit from addressing uncertainty and unobservable dynamics directly.

ACKNOWLEDGMENTS

We thank Leslie Pack Kaelbling from MIT for helpful discussions and our reviewers for valuable suggestions that significantly improved this paper.

REFERENCES

[1] Benini, L., Bogliolo, A., Paleologo, G. A., and De Micheli, G., "Policy Optimization for Dynamic Power Management," *IEEE Transactions on Computer-Aided Design*, Vol. 18, Issue 6, pp. 813–833, 1999.

[2] Bertsekas, D.P., "Dynamic Programming and Optimal Control," *Athena Scientific*, 3rd edition, 2005.

[3] Billings, D., Papp, D., Schaeffer, J., and Szafron, D., "Opponent Modeling in Poker," in *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 493–499, 1998.

[4] Carmel, D. and Markovitch, S., "Incorporating Opponent Models into Adversary Search," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 120–125, 1996.

[5] Duda, R. O., Hart, P. E., and Stork, D. G., *Pattern Classification* (2nd ed.), New York: John Wiley & Sons, Inc., 2001.

[6] Duong, T., Bui, H., Phung, D., and Venkatesh S., "Activity Recognition and Abnormality Detection with the Switching Hidden Semi-Markov Model," *International Conference on Computer Vision & Pattern Recognition*, 2005.

[7] Hwang, C. H. and Wu, A., "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation," in *Proceedings of the International Conference on Computer Aided Design*, Morgan Kaufmann, San Francisco, CA, pp. 28–32, 1997.

[8] Kaelbling, L. P., Littman, M. L. and Cassandara A. R., "Planning and Acting in Partially Observable Stochastic Domains," *Artificial Intelligence*, 101, pp. 99–134, 1998.

[9] Littman, M., Nguyen, T., Hirsh, H., Fenson, E. M., and Howard, R., "Cost-Sensitive Fault Remediation for Autonomic Computing," *International Joint Conference on Artificial Intelligence*, 2003.

[10] Madani, O., Hanks, S., and Gordon, A., "On the Undecidability of Probabilistic Planning and Infinite Horizon Partially Observable Markov Decision Processes," in *Proceedings of the Seventeenth National Conference in Artificial Intelligence*, pp. 409–416 1999.

[11] Papadimitriou, C. and Tsitsiklis, J.,"The Complexity of Markov Decision Processes," *Mathematics of Operation Research*, Vol. 12, Issue 3, 1987.

[12] Ren, Z. and Krogh, B.H., "Hierarchical Adaptive Dynamic Power Management," *IEEE Transactions on Computer*, Vol. 54, Issue 4, pp. 409–420, 2005.

[13] Shani, G., Brafman, R., and Heckerman, D. "An MDP-based recommender system," in *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pp. 453–460, 2002.

[14] Simunic, T., Benini, L., Glynn, P., and De Micheli, G., "Event-Driven Power Management," *IEEE Transactions on CAD*, Vol. 20, Issue 21, pp. 840–857, 2001.

[15] Simunic, T., "Dynamic Management of Power Consumption," in R. Graybill and R. Melhem, editors, *Power Aware Computing*, Chapter 6, Kluwer Academic, 2002.

[16] Srivastava, M., Chandrakasan, A., and Brodersen, R., "Predictive System Shutdown and Other Energy Efficient Programmable Computation," *IEEE*

Transaction on VLSI Systems, Vol. 4, Issue 1, pp. 42–55, 1996.

AUTHORS' BIOGRAPHIES

Georgios Theocharous received his Ph.D. degree in Computer Science in 2002 from Michigan State University. From 2002 to 2004 he was a post-doctoral associate at the Computer Science and Artificial Intelligence Lab at M.I.T., and in October 2004 he joined Intel as a research scientist. His research interests include computational models of learning and planning under uncertainty and their applications to the real world. Specific models include reinforcement learning, completely and partially observable Markov decision processes (POMDPs), semi-Markov decision processes, hierarchical POMDPs, and dynamic Bayesian nets. His e-mail is georgios.theocharous at intel.com.

Shie Mannor received a Ph.D. degree in Electrical Engineering from the Technion-Israel Institute of Technology in 2002. From 2002 to 2004, he was a postdoctoral associate at the Laboratory for Information and Decision Systems at M.I.T. Since 2004 he has been an Assistant Professor of Electrical and Computer Engineering at McGill University. He was a Fulbright scholar in 2002, and he is currently a Canada Research Chair in Machine Learning. His research interests include machine learning, planning and control under uncertainty, multi-agent systems, and he has a particular interest in applications of machine learning in networks and information technology. His e-mail is shie.mannor at mcgill.ca.

Nilesh N. Shah is the principal investigator of the User Activity-Based Adaptive Power Management research project. His research is positioned at the intersection of mobile platforms, power management, machine learning, and activity inference. Shah has held several positions within Intel. Most recently, he managed an Advanced Platform Development team within the Mobility Group. He played an expatriate role as manager, helping to build the team and provide the capability for Intel's Communications Group in Shanghai, China, towards the development of network processors, WLAN/WiMAX chipsets, Ethernet, and optical switches. He joined Intel in 1998 as part of the Chipset Development group after graduating from Purdue University with a Master's degree in Electrical Engineering. His e-mail is nilesh.n.shah at intel.com.

Prashant Gandhi received his M.S. degree in Electrical Engineering in 2005 from Santa Clara University. He joined Intel as a software integrator in February 2006. His interests include mobile power management and power optimized software. Specifically he is interested in investigating the tradeoffs between performance and power for multi-threaded applications optimized for multi-core processors. His e-mail is prashant.gandhi at intel.com.

Branislav Kveton is a Ph.D. student in the Intelligent Systems Program at the University of Pittsburgh. After defending his dissertation in the Fall of 2006, he will join the Corporate Technology Group (CTG) at Intel Corporation as a full-time employee. His major interests are solving large-scale stochastic decision problems and real-world anomaly detection. His long-term goal is to keep bridging the gap between theory and the complexity of real-world problems. His e-mail is bransislav.kveton at intel.com.

Sajid Mahmood Siddiqi is a Ph.D. student in Robotics in the School of Computer Science in Carnegie Mellon University, where he received his M.S. degree in Robotics in 2005. His Ph.D advisors are Geoffrey J. Gordon and Andrew W. Moore. He received a B.S. degree in Computer Science and a B.A. degree in Mathematics and Economics from the University of Southern California in 2003. His research focuses on probabilistic and statistical methods for modeling uncertainty, particularly in time series data. In the summer of 2006, Sajid was an intern at Intel Research working with the Adaptive Power Management team, where his mentor was Georgios Theocharous and his manager was Nilesh Shah. His e-mail is siddiqi at cs.cmu.edu.

Chih-Han Yu has been a Ph.D. student in Computer Science at Harvard University since 2005. From 2003 to 2005, he worked on his M.S. degree and conducted research in the Artificial Intelligence Lab of Stanford University. Prior to that, he received his B.S. degree from the National Taiwan University, Taipei, Taiwan. In the summer of 2006, he was a research intern in Intel Corporation. His research interests primarily lie in machine learning and artificial intelligence. In particular, he is interested in applying reinforcement learning and graphical model techniques to operating systems, distributed systems, and robotics. His e-mail is chyu at fas.harvard.edu.

THIS PAGE INTENTIONALLY LEFT BLANK

A Self-Managing Framework for Health Monitoring

Amit Baxi, Corporate Technology Group, Intel Corporation
Nagaraju Kodalapura, Corporate Technology Group, Intel Corporation

Index words: Body Area Network, Zigbee, patient monitoring, autonomic, self-management, power management, fail-safe mechanisms, biometric authentication

ABSTRACT

Existing medical devices and equipment monitor abnormal physiological conditions and trigger audio-visual alarms. However, these devices require high levels of interaction by a doctor since they do not have any self-managing capabilities to reduce user intervention or to autonomically handle alerts.

Recent advances in low-power wireless communication technologies have led to the development of small form-factor, wireless, body-wearable biomedical devices for health monitoring [1]. This enables patient mobility and comfort and enables continuity of care from hospital to home. In this paper we propose a self-managing framework for health monitoring using body-wearable bio-devices [2] that can reduce the doctor's intervention for patient management. This will enable doctors to effectively manage a greater number of patients and reduce the number of errors inherent in paper-based processes.

We illustrate three usage models where this self-managing framework can be applied: remotely monitoring patients at home, monitoring fetal well-being in a maternity ward, and monitoring critical patients in an Intensive Care Unit (ICU).

INTRODUCTION

There has been a significant development in bio-medical instrumentation over the last two decades. A number of high-tech medical diagnostic and therapeutic systems are used by doctors for diagnostics, for monitoring critical patients, and for delivering treatment. Such devices detect abnormal physiological states, and they trigger audio-visual alarms when the limits are crossed. However, they require frequent intervention by doctors to manage these alarms.

If such patient monitoring systems could be equipped with some intelligence and self-managing capabilities, the systems would be able to autonomically handle several alarm conditions for efficient operation, would require less intervention by doctors and would reduce errors related to patient management.

In this paper we describe a self-managed generic framework for digital health monitoring that can be applied in different use cases. The self-managing capabilities of such a framework are useful to efficiently manage several tasks that usually require user intervention.

We believe that such a system design will enable medical device manufacturers to design their biomedical front-end devices to be compatible with a generic framework and this will drive standardization of hardware and software interfaces for medical devices.

Moreover, this framework utilizes the advances in Personal Computer (PC) and server technology to provide reliable, generic, cost-effective and efficiently self-managed patient monitoring solutions.

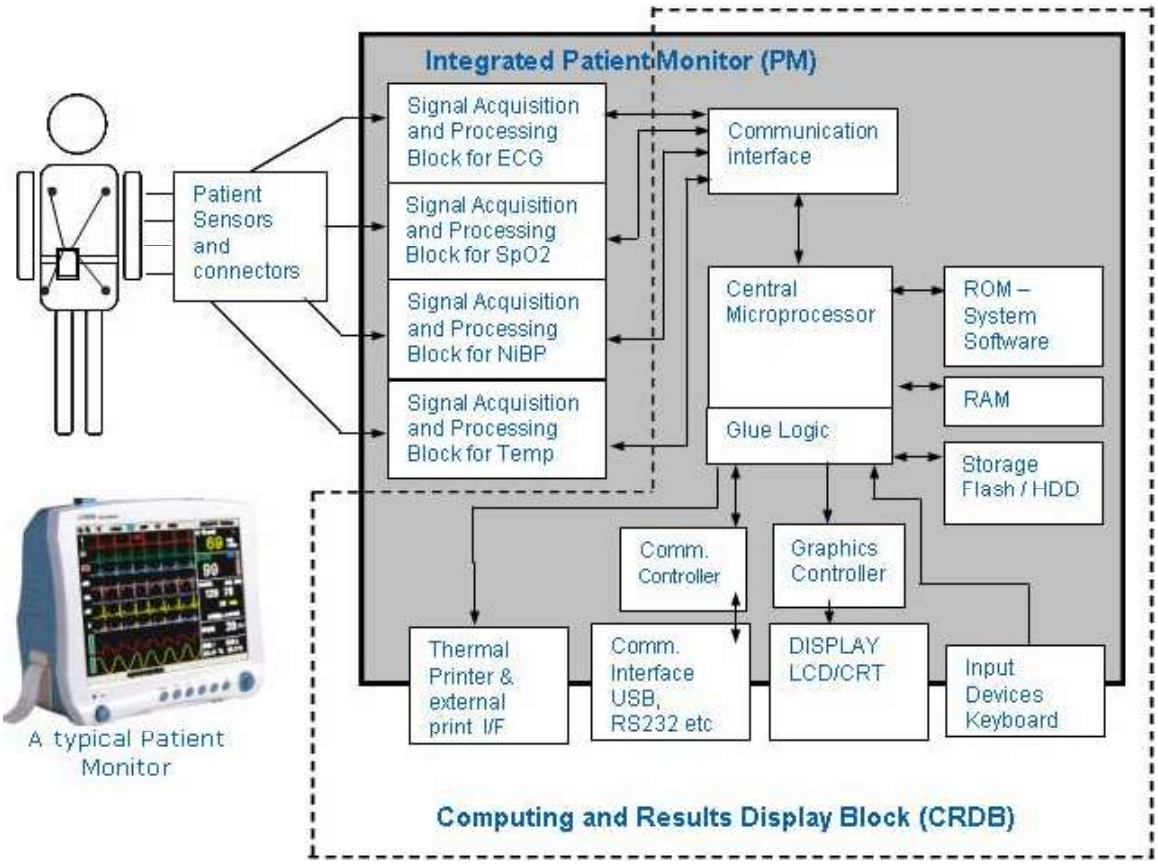

Figure 1: A conventional multi-parameter patient monitor

OVERVIEW OF EXISTING PATIENT MONITORING SOLUTIONS

Patients in the Intensive Care Unit (ICU) are usually wired to a bedside patient monitor known as a Multi-Parameter Patient Monitor (MPM). The MPM monitors one or more of the patient's vital physiological signals such as Electrocardiogram (ECG), Blood Oxygen Saturation (SpO2), Non-Invasive Blood Pressure (NiBP), Invasive Blood Pressures (IBP), Respiration, Temperature, Airway Gases, etc. Patient Monitoring Systems (PMS) are usually dedicated embedded systems, each with its own sensors, biomedical front-end hardware, LCD display, keypad, integrated thermal printer, communication interfaces such as RS232 or USB, and battery backup. These systems are quite expensive and thus contribute to the overall high costs of healthcare. These dedicated embedded systems have proprietary hardware and software. The biomedical Original Equipment Manufacturers (OEMs) avoid the use of general-purpose PC platforms for critical applications since the PC's hardware, software, and operating system (OS) are not optimized and reliable enough for medical applications. A typical MPM is shown in Figure 1.

Patient monitors have biomedical signal processing algorithms for detecting abnormal physiological conditions such as cardiac arrhythmias, apnea, low blood pressure, etc., and the monitor usually gives an audio-visual alert when such a condition is detected. The doctor usually intervenes when an alert is generated and either silences the system or takes corrective action to treat the patient. Apart from alerts generated by physiological conditions, other alerts are also generated to notify system status or malfunctions such as a low battery condition, sensor coming off the patient, etc. Since a doctor managing an ICU has to respond to alerts from several patients, it becomes an uphill task for the doctor to manage alerts, if the patient monitoring system does not have the capability to self-manage such alerts.

Our proposed framework addresses these issues by embedding intelligence at various levels in the system, in order that the system can manage itself and only require intervention by a doctor for critical physiological alerts.

As seen from Figure 1, the Computing and Results Display Block (CRDB) in a typical MPM is quite similar to that of a general-purpose PC. Hence, it makes sense to functionally separate the CRDB from the integrated PMS

and use a single PC to cater to the computing and results display requirements of multiple patients.

A SELF-MANAGING FRAMEWORK FOR HEALTH MONITORING

Architecture

The evolution of low-power wireless communication technologies like Zigbee (IEEE 802.15.4), low-power Bluetooth*, etc. has enabled the development of small, body-wearable, wireless sensors for patient monitoring.

As shown in Figure 2, these sensors can be configured to form a Wireless Body Area Network (WBAN) [3] and enable monitoring of multiple bio-parameters (such as ECG, Pulse Oxygen saturation, Blood Pressure, etc.) of multiple patients at a central location. Each body-wearable sensor, known as a Bio-Front End device (BFE), is composed of a sensor for bio-signal sensing, the related front-end hardware, a low-power microcontroller for data acquisition and a wireless transceiver for data transfer to the receiver. There can be multiple BFE devices connected to a patient for monitoring multiple parameters. An aggregator (AGG) device worn by the patient performs the function of receiving the wirelessly transmitted data from multiple BFE devices connected to a patient and transmits the aggregated data to a backend PC or server. The AGG device can be a device like a Portable Digital Assistant (PDA) or a scaled down version of that without a Liquid Crystal Display (LCD). The AGG and the BFE devices form a localized WBAN, with each BFE device having a unique device ID.

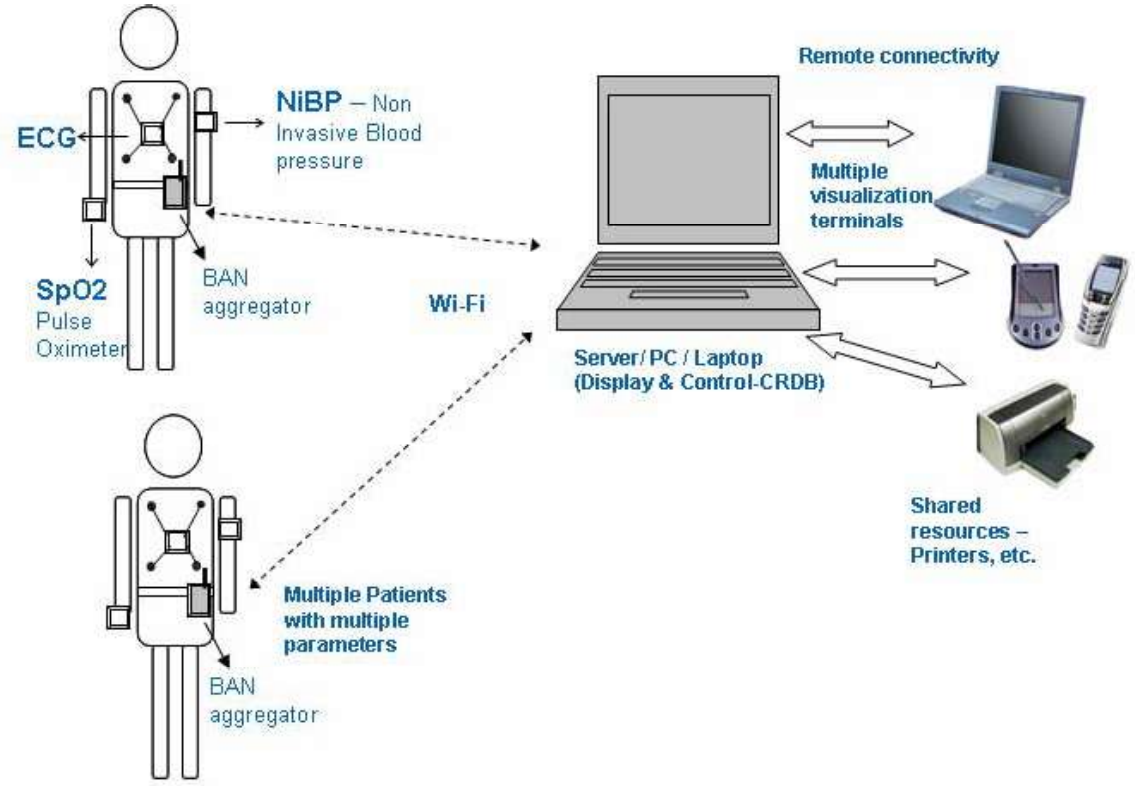

Figure 2: Architecture of a body-wearable, wireless health monitoring platform

Following are the advantages of this architectural framework:

- The functional partitioning of the PMS allows a single PC to cater to the computing and results display requirements of multiple patients, thereby reducing the cost of the overall system.
- There is better resource utilization since a single resource (like a network printer) can be shared across multiple patients.
- It enables patient mobility and comfort.
- It allows central monitoring of multiple patients from a single location.

- It allows the entire patient-monitoring framework (consisting of WBANs of multiple patients) to be managed from a central location.
- Such a framework does not require the doctor to be in close proximity to the patient or to the monitoring equipment, since the patient's physiological data can be made available to the doctor anytime and anywhere on his hand-held communication device.

System Components in the Health Monitoring Framework

There are three major components in the health monitoring framework as shown in Figure 3:

- The BFE device.
- The AGG.
- The backend computational platform such as a PC or server.

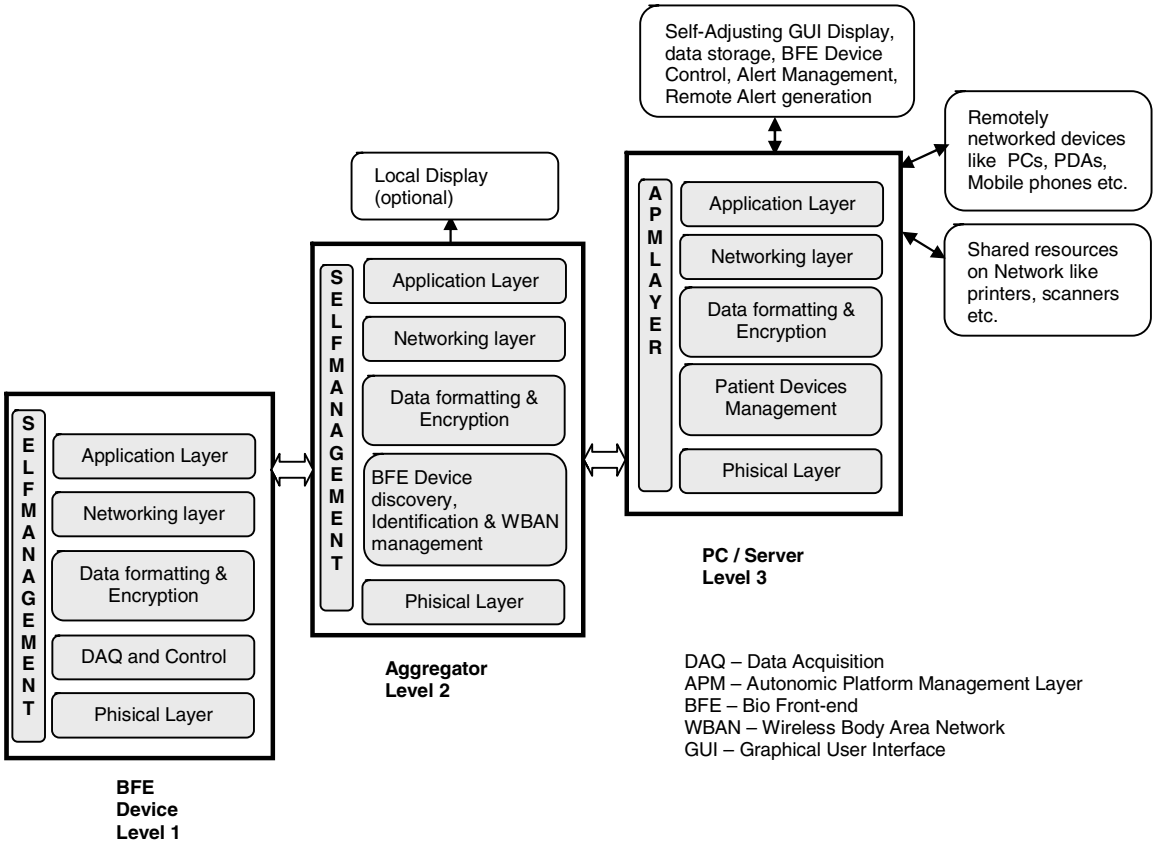

Figure 3: System components of a self-managing framework for health monitoring

The Bio-Front End

The BFE device, which is worn by the patient, consists of the sensor, the front-end hardware and firmware for processing the sensor signals, a low-power microcontroller and a low-power wireless transceiver device for communication. The communication technology between the BFE and the AGG device can be based on a low-power wireless standard like the 802.15.4 (Zigbee) or low-power Bluetooth in order to maximize the battery life. The type of sensor and the front-end hardware is specific to the physiological signal being measured. For example, if the BFE device is designed to monitor ECG, the sensors would be pre-gelled electrodes, whereas for pulse oximetry, the sensor would be an optical finger probe. The microcontroller controls the overall functionality of the device and performs the functions such as data acquisition, device control, data formatting, packet forming, wireless transmission and execution of the commands sent by the AGG device.

The Aggregator

The AGG device is also worn by the patient and it serves to aggregate the data from multiple BFE devices connected to a patient and sends these data wirelessly to a PC for further processing. The AGG uses low-power wireless technology like Zigbee to communicate with the BFE devices and Wi-Fi or other appropriate communication technology to communicate with the PC,

since the required data rate and range is larger than that for a body-area network. The AGG device also serves as the intermediate messaging link between the BFE device and the PC.

The Personal Computer or Server

The PC is used to collect data from multiple AGG devices connected to multiple patients and process these data. The PC also serves to display the physiological waveforms and parameters of several patients on the screen, does further signal processing and computation, performs feature extraction, stores data, and communicates with other terminals over the Internet protocol. The PC also keeps the doctor updated on the patient status by sending the patient information on his mobile phone or PDA. It also manages resources such as printers, scanners, and other devices, which are shared across multiple patients.

SELF-MANAGING THE HEALTH MONITORING FRAMEWORK

The flow of events in self-managing a device in the framework is shown in Figure 4. Self-management involves sensing or monitoring parameters from the device or the system to be managed and then analyzing the parameters to find out whether a corrective action is required. Intelligent algorithms and decision trees are used to decide if a management action is required and then to arrive at the most effective decision to act on the inputs. The decision is then executed by the execution engine in a feedback loop to effectively manage the device or the system without the need for user intervention.

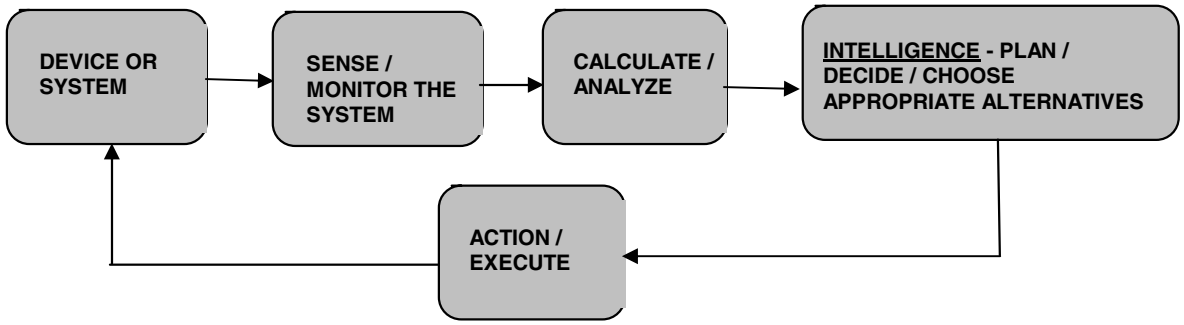

Figure 4: Flow of events in self managing a system

The proposed health-monitoring framework can be efficiently self-managed by building limited self-managing capabilities in each building block and shifting major intelligence and decision-making capabilities to the PC.

The PC makes the control decisions based on status inputs from the connected BFE and AGG devices and then instructs the devices to perform certain self-management functions.

Following are the advantages of this self-management approach:

- BFE devices may not have high computational capability. Hence, they can send their status information to the PC and use the processing power of the PC to make optimal self-management decisions.
- The PC is the central hub of information gathering and it is aware of the status of all devices in the network. Hence, the PC is better equipped to make optimum decisions by considering the status of the entire platform as a whole, rather than the status of a single device.

The BFE and the AGG devices should have limited self-managing capability in order to manage themselves in case the communication link to the PC fails. Such devices can have fail-safe mechanisms and recovery algorithms to put the device in a safe state in the absence of PC connectivity.

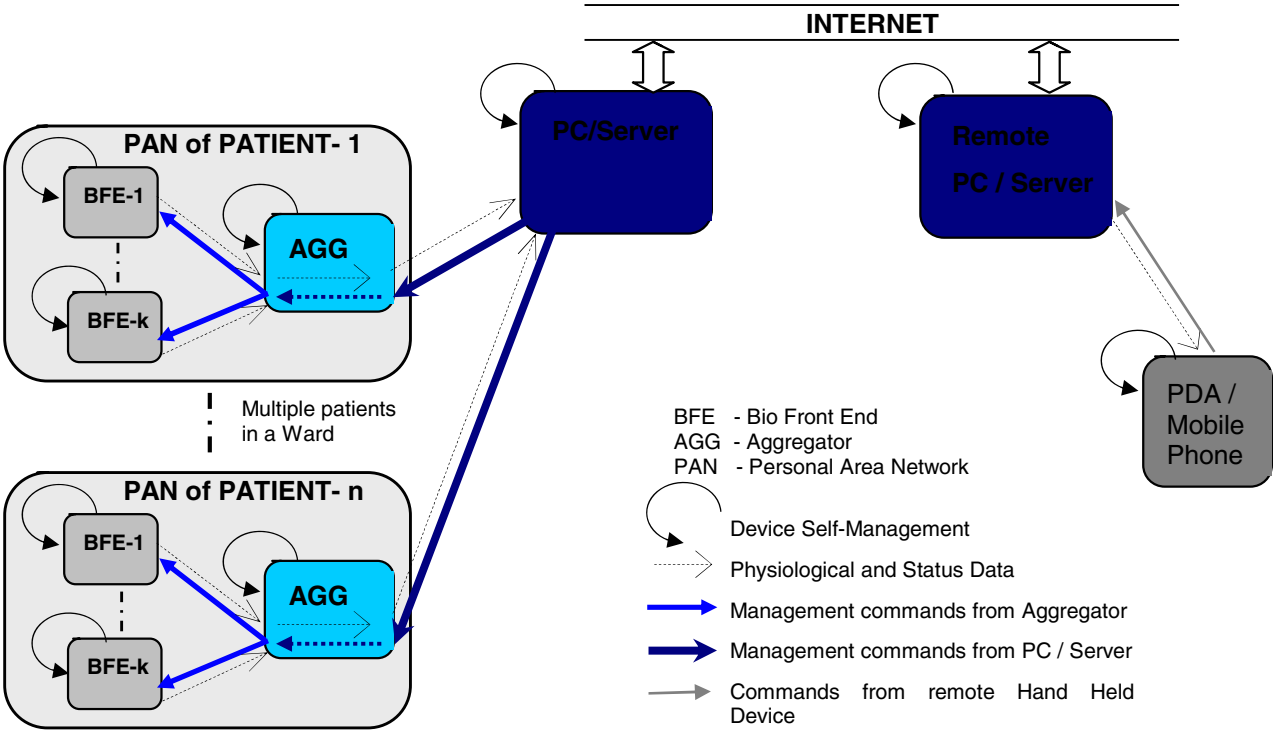

Figure 5: Self managing framework for health monitoring

Figure 5 shows a self-managing framework for health monitoring. Each BFE device connected to the patient self-manages itself to some extent and can also be managed by its WBAN AGG device and by the back-end PC/server. The management scope of a BFE device is limited to itself. The management scope of an AGG device is limited to itself and also to manage all the BFE devices in the WBAN of that particular patient. The PC is the central hub of information gathering and does the system-level management, making decisions based on inputs from all the devices in the system. The management scope of PC extends to itself, all AGG devices in its network and all the associated BFE devices. The PC also interacts with other servers that are remotely connected via the Internet link. The remotely connected hand-held devices and servers can be used to manually manage the BFE and AGG devices connected to the network after secure authentication. The PC manages and communicates with the BFE devices in the network via the intermediate AGG devices.

The self-managing capabilities of different blocks of the framework are described below.

Self-Managing the BFE Device

The self-managing capabilities of the BFE device may vary according to the type of the device. An ECG BFE device may self-manage itself differently than a SpO2 BFE device. Typically, a BFE device can exhibit the following self-management capabilities:

- *Device to patient binding*. This involves binding all the BFE devices and the AGG device connected to a patient to a unique patient-ID and a unique Personal Area Network (PAN) identification (ID) number. The PAN-ID of each WBAN is closely coupled to the Patient-ID (i.e., the patient registration number assigned to the patient by the hospital). The unique Patient-ID and the patient history may be programmed in a small Radio Frequency ID (RFID) wrist strap attached to the patient, when the patient is admitted to the hospital. The BFE devices may have RFID reading capability to read the Patient-ID and be associated with the unique Patient-ID, when they are brought into close contact with the patient's RFID wrist strap. Once each BFE device and AGG device gets associated with the Patient-ID, all the connected devices can form a unique PAN-ID for the patient.

Alternatively, each BFE device may have the capability to autonomically identify a patient based on some biometric technique. Data packets sent by the BFE devices to the AGG device and to the PC are tagged with the unique Patient-ID in order to bind them to the patient's individual database.

- *Network association.* All the BFE devices and the AGG device connected to a patient are programmed to have a unique PAN-ID for their network. As mentioned above, the PAN-ID can be programmed into the BFE devices and AGG device before connecting them to a patient. When the BFE device is turned ON, it looks for beacons sent by the AGG device to check the AGG device's PAN-ID and gets itself associated with the network if it matches with its own PAN-ID. The AGG in turn identifies the newly joined BFE device, determines the type (i.e., ECG, SpO2 etc.) of the BFE device and intimates the same to the PC. The PC dynamically adjusts the Graphical User Interface (GUI) on its screen to effectively optimize the screen space to display the signals from the newly networked BFE device.
- *Self-configuration*. When a BFE device is turned ON it automatically configures itself to a default state. It also checks the attached sensors and re-configures itself for its mode of operation. For example, for an ECG BFE device, there are two modes of operation: diagnostic and monitoring. In diagnostic mode a 10 lead cable is used to sense the ECG signals and the ECG signals are digitized at 500 samples per second per lead. However, in the monitoring mode, a 3 or 5 lead ECG cable is used to sense the ECG signals and the ECG signals are digitized at a sampling rate of 200 samples per second per lead. At start-up, the BFE device can identify the type of ECG cable connected at its input and automatically adjust its mode of operation, the number of channels to be sampled, the sampling rate per channel, and it can reserve the bandwidth required for the wireless data transfer.
- *Self-calibration and self-check*. At power ON or when requested by the PC, the BFE device can also execute self-calibration and self-check routines to dynamically adjust its settings for maximum accuracy. For example, an ECG device may input a fixed reference voltage to its Analog to Digital Converter (ADC), convert it to a digital representation, and then adjust its gain and offset errors to compensate for the errors in the ADC. Similarly, a SpO2 device may calibrate its optical sensor characteristics, whereas, a NiBP BFE device may calibrate its air pressure sensor and self-check its air release solenoid valves and the air pump for proper operation.
- *Power management*. The BFE device can dynamically turn OFF the idle peripherals when not in use. For example, the BFE device microcontroller can turn OFF its communication ports, data acquisition, and control peripherals when not in use in a particular mode of operation. Battery capacity monitoring can be done at periodic intervals by the BFE device and the battery capacity information can be sent to the PC via the AGG device.

  The BFE device can reduce the clock frequency of its microcontroller and related peripherals if too much computing is not required by the BFE device sensor. Very low data rate BFE devices such as body temperature sensors can enter sleep state between periodic measurements, whereas, the medium data rate devices such as SpO2 may turn OFF the wireless transmitter's RF circuitry between data transmissions in order to conserve battery power.

  The BFE device may either execute certain power management functions autonomically or execute them when instructed by the PC. For example, when the battery voltage falls below a certain threshold value, the PC may instruct an ECG BFE device to reduce the ECG sampling rate, and the PC may interpolate the sub-sampled ECG data for proper visual representation. Alternatively, depending on the status of other sensors attached to the patient, the PC may instruct the ECG BFE device to reduce the number of ECG signals to be sampled and reduce the amount of wireless data transmission to conserve battery power.

  Near the end of battery capacity, the PC and the BFE device may issue audio and visual alerts to inform the hospital staff to change the battery. The BFE device also intimates to the PC its decision to turn OFF to a safe state when the battery capacity reaches 0%.
- *Fail-safe mechanisms*. It is very important for the BFE device to have intelligent algorithms to handle alerts in case of malfunctions and put the BFE device in a safe state to avoid patient injury.

  Some important malfunctions could be microcontroller/hardware malfunction, sensor malfunction and communication failure.
- *Hardware malfunction*. These types of malfunctions can be induced by failure of electronic components or can be due to effects of electromagnetic interference (EMI). EMI can cause the BFE device microcontroller to enter a runaway state that can be very dangerous. This can be mitigated by ensuring that, in the case of malfunction, a watchdog timer running inside the microcontroller device either resets

the microcontroller or puts the microcontroller and associated hardware in a known safe state.

In the case of BFE devices such as a NiBP device, the blood pressure is measured by occluding the brachial artery by inflating a cuff tied to the patient's arm. If the air inside the cuff is not released due to hardware malfunction, the blood flow to the patient's hand stops and may cause serious tissue damage. Hence, it is very important to have either a redundant microcontroller or some standby mechanical devices (such as normally open solenoid air release valves) to ensure air release even in case of power or hardware failure.

- *Sensor malfunction*. The BFE device also needs to handle situations when the sensors connected to the BFE device malfunction or when they get disconnected from a patient. For example, for an ECG BFE device, the number of electrodes connected to a patient may vary from three to ten. The BFE device needs to continuously monitor the impedance of each electrode and generate an alert in case the electrode gets disconnected from the patient. The BFE device sends the sensor status to a PC periodically, and the PC can generate an alert in case of malfunction. Apart from generating an alert it may also be necessary to stop displaying the waveform of the disconnected ECG lead to avoid the doctor misinterpreting the noise picked up by the disconnected electrode as a valid ECG signal. The ECG BFE device may, however, continue to monitor the ECG signals from other electrodes.

  In another case, an air leak may be present in the cuff tied to the patient's arm for blood pressure measurement. The NiBP BFE device needs to monitor the rate of increase in cuff pressure once the air pump starts pumping air into the cuff. If the cuff pressure does not increase at the required rate an air-leak alert is transmitted to the PC.

- *Communication failure and communication errors*. The BFE device also needs to handle situations when the sensors and its hardware components are functioning normally but when the communication link to the AGG device or the PC fails. In such cases the BFE device is neither able to send the physiological data and its status to the PC, nor is it able to receive autonomic control messages from the PC. In such a scenario, the BFE device may save the patient's data in its local memory for transmission to the PC when the communication link is re-established. If the BFE device does not have sufficient memory to store data for long periods of disconnection, it may save the status of the last few minutes using a circular memory buffer. In case of communication failure, the BFE device keeps on looking for the periodic beacons sent by the AGG device in order to check the PAN-ID and re-establish the network connection.

  Communication errors may be introduced in the communication link between the BFE device, the AGG device and the PC due to the presence of other networks operating on the same wireless frequency. A typical example can be the interference of Wi-Fi (802.11b) in a Zigbee (802.15.4) network since they both operate in the same frequency band of 2.4 GHz. In another scenario, two patients may be connected with BFE devices that use the same frequency channel for communication to their respective AGG device. When such patients come in close proximity, their networks conflict with each other causing errors in communication. The BFE-AGG body area network needs to have a robust interference detection and mitigation algorithm by which a conflicting BAN would dynamically sense and change its operating frequency channel in case of interference from a similar network.

  In case the BAN is not able to mitigate the interference, robust error detection techniques should prevent the BFE device from executing wrong commands received from either the AGG device or the PC.

Self-Managing the Aggregator Device

The AGG device serves as a communication link between the WBAN formed by the BFE devices and the backend PC/server. The AGG device has a low-power wireless interface like 802.15.4 or a low-power Bluetooth to network the BFE devices and also a long range, high data rate interface like Wi-Fi to interface with the back-end PC. The AGG device may or may not have a local LCD display and control keypad. A typical example for an AGG device could be a PDA. The self-managing capabilities of the AGG device may be similar to that of the BFE devices.

The AGG device manages the BAN and allows BFE devices to join or leave the BAN seamlessly. Automatic BFE device discovery, device type identification, and BAN networking are the responsibility of the AGG device. The AGG should have sufficient capabilities to manage the BAN efficiently even in the absence of the communication link to the PC and have sufficient memory for storage of patients' physiological and status data over extended periods of operation.

Another important function of the AGG device is data encryption/decryption for network security.

Autonomic Platform Management (APM) by the PC

The PC or server is the central hub of data collection from multiple WBANs of multiple patients. The PC aggregates data from multiple patients, processes the data, stores them in specific formats and also makes the data available to multiple remote terminals for display. The PC also makes the patients' data available to the doctor at any place and at any time by transmitting important patient data to hand-held devices like mobile phones and PDAs. The PC also has the capability to issue control commands to any BFE device in its network to change its operating parameters. The PC also manages the shared resources like printers, fax equipment, etc., which are connected to it.

Since the PC is the central hub of information aggregation from BANs, user inputs from GUIs and information from remote terminals, it is better equipped to make self-management decisions for the entire health-monitoring framework. Also, since the PC has tremendous computation capacity it is better suited to run complex decision trees to arrive at the best management decision.

A single PC can be used to cater to the processing needs of multiple patients in a ward of the hospital. Multiple wards of the hospital can be connected to a server to further aggregate data from the entire hospital and to send/receive messages to hand-held devices.

The self-managing capabilities of the PC/server, shown by the APM layer in Figure 3, can be as follows.

- *Network management.* The PC forms a network of the AGG devices and dynamically changes the network topology as patients get transported in and out of the wards for tests and operative procedures. The PC is also aware of the devices in the BAN of each AGG device. The PC enables automatic device discovery, device association and disassociation algorithms, and automatic device-to-patient binding. Each patient is associated with a unique PAN-ID, and all devices in the BAN of the same patient have the same PAN-ID. The PC binds the real-time data of each patient to the patient's respective backend database.

  Apart from managing the networking of AGG and BFE devices, the PC also manages the shared resources on its network like printers, scanners, fax machines, etc.

- *BFE device control*. Since critical patients have different monitoring requirements as compared to non-critical patients in recovery, the PC can re-configure the BFE devices in different operating modes and change the device settings depending on the level of monitoring required.

  The PC is better equipped to handle self-management functionality of the health-monitoring platform as a whole. The PC may run complex bio-signal processing algorithms and based on the results may instruct the BFE device to execute certain self-management tasks. For example, the PC may analyze the ECG signal for the presence of baseline drift and may send a message to the ECG BFE device to restore the ECG baseline to ground by re-charging an AC coupling capacitor in the BFE device hardware. Similarly, if very small amplitude ECG signals are being sensed by the electrodes, it can instruct the BFE device to increase the amplification of the BFE device's programmable gain amplifier hardware.

  For a NiBP BFE device the PC may instruct the device to take blood pressure measurements either at programmed intervals or autonomically when certain thresholds are exceeded or under manual control.

  The PC can also perform autonomic power management of the devices connected on its network. The PC can turn ON or turn OFF the BFE devices or keep them in SLEEP state. The PC can wake the sleeping devices at appropriate times, instruct them to take a measurement and put them to SLEEP again. Also, the PC can extend the battery life of a device by dynamically changing the device parameters and settings. For example, the PC can instruct an ECG BFE device to reduce the number of leads to be sampled or reduce the sampling rate per lead, depending on the quality of ECG required.

- *Autonomic GUIs*. The PC can autonomically adjust the screen space to display a number of vital signal waveforms and numerical data from a number of patients. The screen can be automatically partitioned and re-adjusted to accommodate the data from new devices that join the network, and the screen space can be made free and utilized in a better manner when the BFE devices leave the network. When the screen space is limited, and displaying all parameters from all patients is not possible, the PC should give weight to critical parameters such as ECG readings.

- *Managing alerts*. The PC is the central computing resource for the entire health-monitoring framework and needs to process alerts coming from almost all devices attached to the network.

  Broadly, the alerts can be classified as alerts generated by the patient's physiological status, device status, and communication errors.

- *Alerts due to patient's physiological status*. The PC processes the physiological signals and parameter data from the patient and checks if any of the

programmed limits are exceeded. For example, the PC computes the heart rate by identifying and counting the number of 'R' waves in the patient's ECG per minute. In case the heart rate crosses the programmed limits, the PC may request the NiBP BFE device to take frequent blood pressure measurements and may also power ON a defibrillator machine in case the patient needs to be administered an electrical shock to restore normal heart rhythm. The PC can also intelligently identify irregular heart rhythms called arrhythmias and keep a log of these with a time stamp. In some critical conditions, the PC can automatically send the abnormal physiological data to a doctor's mobile phone/PDA for diagnosis and also trigger audio-visual alarms locally.

- *Device status alerts*. The PC monitors the status of the BFE and AGG devices and manages the alerts generated due to low battery, sensor failure or disconnection, component failure, calibration errors, etc. For example, if the patient condition is not critical, under low battery conditions the PC may instruct an SpO2 BFE device to intermittently monitor the blood oxygen saturation at 1-minute intervals, rather than monitoring it continuously. This allows the LEDs inside the SpO2 finger probe and the associated circuitry to sleep between measurements, thereby extending the battery life. For sensor disconnection the PC can raise appropriate audio-visual alarms and/or page the hospital staff on duty.

  In the case of component failure or calibration errors the PC can compensate for the error by software correction or instruct a device like the NiBP to measure blood pressure using the redundant pressure sensor. Once an error is detected and flagged by the PC, the PC also monitors whether the error has been corrected and removes the error flag accordingly. Alternatively, the user may manually acknowledge the error and put the PC in a mute state for a certain amount of time within which the user is expected to correct the error.

- *Fail-safe mechanisms*. Since the PC is the central computing resource for the health-monitoring platform, it is necessary to build reliability into the PC platform. Reliable hardware design, a reliable low-latency hard real-time OS, a reliable networking stack and application software are necessary to build a medical-grade PC. The PC should be immune to the EMI generated by other medical equipment in its vicinity and should also have low EMI emissions. The PC should have a battery backup to remain operative in case the mains power supply fails. The PC should have the capability of self-monitoring, preventive maintenance, and have redundant processing cores and configurable logic to heal itself in case of component failure. The storage media and the Internet broadband link should be robust. The aggregated patient data and the network status information should also be copied in a central archive so that in case of malfunction, a standby PC should be able to take over the monitoring functionality of the failed PC without loss of data.

USAGE MODELS

This self-managed digital health-monitoring framework can be applied in several use cases, some of which are described below:

Use Case 1: Remotely Monitoring Patients at Home

Home monitoring involves monitoring of non-critical patients, patients in the recovery stage after being discharged from hospital, proactive health monitoring, or monitoring the health of the elderly within their homes or care facilities.

Joe, 68 years old, has recovered from open heart surgery and has been discharged from the hospital. Lately, Joe has become forgetful and forgets even basic tasks like taking his medicines on time. Joe's children are working and are concerned about Joe's health when they are at work. Dr. Smith wants to remotely monitor Joe's recovery as Joe performs his daily activities to ensure that his recovering cardiovascular system is not subject to a sudden stress. Joe is fitted with a wireless ECG device, a NiBP device, and a Pulse Oximetry device for monitoring his blood oxygen saturation. The devices connected to Joe are miniature, lightweight, and they do not interfere with his daily activities.

Joe has a Wi-Fi-enabled desktop PC in his living room which is also used by Joe to watch TV. The PC continuously monitors Joe's vital parameters in the background and sends data to the hospital in real-time. Wireless webcams fitted in different rooms of his home also send streaming video data to the PC. Joe's PC is connected to the Internet using a broadband link.

While sitting at his laptop in the hospital, Dr. Smith connects to Joe's PC and runs the pre-designed protocol for remotely monitoring open heart surgery patients like Joe. Dr. Smith runs the protocol and then leaves for an urgent operation. The protocol remotely and autonomically programs Joe's ECG device to continuously monitor a 3-lead ECG, the NiBP device to take a blood pressure reading once every hour, and the Oximetry device to monitor only in critical situations. The protocol automatically sends a reminder audio-visual message on Joe's PC screen asking him to undergo the

morning exercise on his home treadmill and also programs the treadmill to limit the maximum speed to 2 miles per hour and limit the exercise time to 10 minutes. The reminder message alerts Joe in the midst of his TV program. Joe starts walking on the treadmill and during his exercise session, his heart rate increases to 130 beats per minute (bpm). The increased heart rate is monitored by the PC, and the PC triggers the NiBP device to take frequent blood pressure measurements and it also automatically stops the treadmill as a precautionary measure to prevent excessive stress. The increased heart rate condition also turns on the Pulse Oximetry device to monitor Joe's oxygen saturation. When Joe's heart rate returns to normal after his exercise session, the NiBP device again increases the BP measurement interval to 1 hour while the Oximetry device turns OFF. Since Dr. Smith had set the mobile phone alert limit for heart rate to 150 bpm, BP limits to 150/100 mmHg, and the SpO2 limit to 94%, he was not alerted on his mobile phone since Joe's vital signs were within the limits. When Dr. Smith returns from his operation he examines the stored vital parameters of Joe and he is happy with Joe's increased stress-handling capability. Meanwhile, Joe's son is also able to remotely keep an eye on Joe from his office by monitoring the streaming webcams and Joe's vital parameters. Dr. Smith has remotely also fed the medicine schedule on Joe's PC. Joe's PC flashes audio-visual messages on its screen in a timely manner to remind Joe to take his medicines.

Use Case 2: Wireless Fetal Monitoring During Labor

Judy is pregnant and has been admitted to the hospital for delivery. Dr. Willy expects a normal delivery but decides to monitor Judy's Uterine Contractions (UC) and Fetal Heart Rate (FHR). The fetal heat rate variability is an important parameter to asses fetal well being and to assess whether the fetus would be able to sustain the stress of uterine contractions during delivery. Dr. Willy fits Judy with a wireless ultrasound sensor and a uterine pressure sensor by means of a belt around her abdomen. The ultrasound sensor monitors the FHR while the pressure sensor monitors Judy's UCs and sends the data wirelessly to the central server of the labor ward, where a number of patients are being monitored simultaneously. Judy is still not in labor and is able to take frequent walks around her room and in the ward, while still being closely monitored. Judy does not feel the subtle UCs, indicating the start of labor. However, minute uterine pressure changes are picked up by the pressure sensor and are evident in the graphical tracing at the central monitoring terminal. There is a sudden and steep drop in the FHR which goes unnoticed by the busy hospital staff. However, the FHR analysis software on the server detects the sudden drop in FHR, and the software automatically sends alert messages and FHR tracings to Dr. Willy and the chief nurse on their mobile phones. Dr. Willy is out of hospital when he receives the message on his mobile phone. Dr. Willy immediately rings up the chief nurse and instructs her to be prepared for an emergency cesarean operation by the time he reaches the hospital. Judy undergoes an emergency cesarean operation and delivers a healthy baby. During delivery Dr. Willy notices the umbilical cord entangled around the fetus's neck resulting in partial suffocation and thereby decreasing the FHR. Dr. Willy is thankful that the timely alerts by the central monitoring system saved the baby's life.

Use Case 3: Critical Patient Monitoring in an Intensive Care Unit

Dr. Bill is a resident doctor currently managing the 16-bed ICU in a hospital. A patient, Jack, operated on via angioplasty, is brought to Dr. Bill's ICU. Dr. Bill tags Jack's admission time in ICU by using a RFID reader to read Jack's RFID wrist strap and also wirelessly transfers a softcopy of Jack's medical record file from inside the wrist strap to the central station for study. Dr. Bill rubs the wireless patient monitoring devices on Jack's RFID wrist strap and connects them to his body. The wireless monitoring devices read Jack's Patient-ID from his wrist-strap to form a BAN and get associated to Jack's backend database on the central station. The computer screen of the ICU's central monitoring station automatically re-adjusts itself to display, monitor, and store Jack's vital signs along with the parameters of other patients. Dr. Bill prepares a monitoring schedule and a drug administration schedule for Jack using the central monitoring application software. The monitoring and drug administration profile for Jack is immediately transmitted to the hand-held devices given to nurses on duty in the ICU. The central station performs periodic diagnostic tests using the attached wireless devices and also sends timely reminders to the nurses to administer drugs to patients, depending on individual patient profiles. Once a nurse administers a drug to a patient she mutes the generated reminder and logs the event, which also gets logged in the central station. The central monitor also runs intelligent algorithms on patient's vital physiological signals and generates alerts, sends alerts to doctors' mobile phones, and prints important events on locally connected printers. Dr. Bill is relieved that the central monitoring station manages most of his mundane tasks and helps him in effectively managing large numbers of patients.

CHALLENGES AND OPPORTUNITIES

There are several challenges which need to be overcome in order to realize such a digital health-monitoring framework:

Platform reliability. Ensuring the reliability of the entire platform is the key to the success of the proposed architecture. Reliability has to be built right from the electronic component level to the OS and application software level. It is necessary to have fail-safe and backup mechanisms to ensure that patient monitoring is not interrupted when parts of the network fail.

Robust wireless communication. Wireless communication is the backbone of the proposed health-monitoring framework. Robust mechanisms should be developed to mitigate interference issues that result when several wireless networks co-exist. Robust error correction and error detection algorithms should be developed to build the same reliability as that of a wired link in the wireless interfaces.

Standardization and interoperability. The medical device OEMs need to agree on a common set of communication protocols and standard hardware interfaces. Devices from different manufacturers should be able to plug into the system seamlessly.

Infrastructure and ubiquity. The health-monitoring framework needs wireless communication infrastructure like Wi-Fi hot spots, routers, switches, etc. which are limited to a hospital or home network. Emerging technologies such as WiMAX can address the problem of seamless wireless connectivity throughout the cities and villages.

Intelligent algorithms. The PC needs to run intelligent algorithms to make self-managing decisions. The algorithms should be continuously evolving and patient centric. Machine learning, artificial intelligence, and prediction algorithms may require tweaking and clinical trials until they reliably self-manage a health-monitoring framework.

Development of miniature, ultra-low power sensors and battery technology. BFE device sensors and hardware need further miniaturization to the level of a small system-on-chip, and the power needs to be optimized to the microwatt level so that the sensors can operate by using ambient light as the power source. Battery technology such as the lithium-polymer and moldable lithylene battery technology have to evolve further to combine high capacity, small form factor, and light weight.

CONCLUSION

We believe that a self-managed wireless health-monitoring framework can significantly improve the quality of healthcare while providing patient comfort, mobility, and continuity of care. Such a framework does not require the doctor to be in close proximity to the patient; however, it still provides the same quality of care. The doctor is enabled to monitor more patients effectively. The number of errors related to paper-based processes is reduced significantly in an autonomically managed framework. The proposed framework leads to better resource utilization, better resource sharing, reduces doctor's intervention, and hence makes healthcare more affordable.

Such an open standards-based health-monitoring platform would motivate more standards-based hardware and software designs and shift the biomedical OEMs from proprietary hardware-centric platforms to standards-based software-centric general-purpose PC platforms.

ACKNOWLEDGMENTS

The authors recognize the valuable suggestions given by the reviewers for improving the quality and content of this paper.

REFERENCES

[1] David Culler, Deborah Estrin, Mani Shrivastava, "Overview of Sensor Networks," *IEEE Computer Society*, August 2004, pp. 41–49.

[2] J. A. Stankovic, Q. Cao, "Wireless Sensor Networks for In-Home Healthcare: Potential and Challenges," *Department of Computer Science, University of Virginia.*

[3] Chee-Yee Chong, Srikanta P. Kumar, "Sensor Networks: Evolution, Opportunities and Challenges," in *Proceedings of the IEEE*, Vol. 91, No. 8, August 2003, pp. 1247–1256.

AUTHORS' BIOGRAPHIES

Amit Baxi is a biomedical engineer working as an R&D Engineer in the Corporate Technology Group, Intel Corporation. He has been in Intel for more than a year and is responsible for architecting and building the hardware and software bio-medical components for Intel's healthcare platforms. He has more than 11 years experience in the design and development of medical embedded systems such as Cardiac Stress Test Systems, Defibrillators, Multiparameter Patient Monitoring Systems, Spirometers, ECG monitoring and diagnostic equipment etc. His e-mail is amit.s.baxi at intel.com.

Nagaraju Kodalapura is an electronics and communication engineer working as an R&D Engineer in the Health Platforms lab of the Corporate Technology Group, Intel Corporation. He has been with Intel for about six years. He is primarily responsible for the design and development of Wireless Embedded System Software for bio medical sensors for Intel's healthcare platforms. His expertise is in the area of wireless embedded systems, device drivers, debuggers, and simulators for multicore

processors. His e-mail is nagaraju.n.kodalapura at intel.com.

THIS PAGE INTENTIONALLY LEFT BLANK

For further information visit:

developer.intel.com/technology/itj/index.htm