# State of the art in autonomic computing and virtualization

Dan Marinescu, Reinhold Kröger

1. September 2007

Distributed Systems Lab
Wiesbaden University of Applied Sciences, 65197 Wiesbaden, Germany
tel: +49-(0)-611-9495219, fax: +49-(0)-611-9495289
web: wwwvs.informatik.fh-wiesbaden.de

DOPSY group
Labor für Verteilte Systeme
Distributed Systems Lab

# 1 Introduction

A hot topic in the early 70's, virtualization has grown again in importance over the past years. In both research and industry sectors, important progress has been made developing new technologies for both the desktop and the server market. In the same period of time, a new vision has emerged in the IT world: the vision of autonomic computing. This implies systems managing themselves in order to achieve global goals defined by the system administrator. This technical report summarizes the period 01.03.2007 - 21.08.2007 of the "Selbstmanagement virtueller Machinen" project conducted at the University of Applied Sciences Wiesbaden under the supervision of Prof. Dr. Reinhold Kröger. The report describes the state of the art in both autonomic computing and virtualization, as a starting point towards the development of autonomic computing techniques for virtual machine-based environments.

This report is organized as follows. Section 2 introduces virtualization by first describing its origins followed by a classification of the different virtualization techniques and finally a case study. Then section 3 deals with management aspects of virtual machine-based environments. In Section 4, the vision of autonomic computing is introduced, including a survey of research work in the field of self-management of virtual machines. Finally, the article is concluded by Section 5.

# 2 Virtualization

## 2.1 History

The first computing systems developed were large and expensive to operate. Due to an increasing demand for usage, these computers evolved first into batch processing (1950) and then into time-sharing systems (1965) that allowed multiple applications (possibly owned by different users) to run simultaneously. Yet these applications were not always perfect, with one faulty application being able to crash the entire system. It was obvious that to increase the reliability of the system, these applications had to be isolated one from another. The first approach was to use one application per computing system. However, besides being a very expensive solution, this has also proved to be wasteful: no time-sharing was used and so computing systems were not used at their full capacity. The tagging of memory blocks was then introduced for time-sharing systems. In the same time, in the field of software development, the instruction-by-instruction simulation of one computer system X on a different system G was a known technique [Gol74]. This was used to develop software for a computing system on which the programmer had no access (sometimes because the system was still under construction). Researchers soon realized that, when $X = G$, more copies of the hardware-software interface of the machine G can run simultaneously on G. This way, each user can run its application (and even its OS of choice) in an isolated environment, called virtual machine.

While simulation could be used to isolate applications running on the same hardware, it also had the drawback of slowing down the machine by a factor of 20 to 1. So researchers focused on improving the performance of the simulation software, which led to the appearance of the virtual machine monitor (VMM) in the late 1960s. Maybe the best known system of that time using a VMM is VM/370 developed by IBM.

The 1970s where flourishing for the virtualization technology, but then the 1980s and 1990s brought both a drop in hardware prices as well as the emergence of multitasking operating systems. The drop in hardware prices meant that it was no longer necessary to assure a high CPU utilization. While in the 1970s, computer architectures were developed with virtualization in mind [PG74], the replacement of mainframes to minicomputers and the PCs gave birth to new computer architectures that didn't offer the necessary hardware support for VMMs [RG05]. As such, by the late 1980s, VMMs became history.

In the 1990s, VMMs where brought back to light by the researchers at the Stanford University in a research project that led to the birth of VMware Inc., the first supplier of virtualization technology for commodity hardware. Nowadays, virtualization is again a hype, this time used to reduce management costs by replacing a bunch of low-utilized server boxes by a single-server system, with almost every big name in the IT industry

being involved in a virtualization project. With VMware leading the market, vendors like SWsoft and XenSource all offer competitive alternatives. And not only that virtualization products are developed for the IA-32 architecture, but also the architecture itself is being extended by both Intel (VT) and AMD (SVM) to support virtualization.

## 2.2 Basic concepts

Virtualization is commonly defined as a technology that introduces a software abstraction layer between the hardware and the operating system and applications running on top of it. This abstraction layer is called *virtual machine monitor* (VMM) or *hypervisor* and basically hides the physical resources of the computing system from the operating system (OS). Since the hardware resources are directly controlled by the VMM and not by the OS, it is possible to run multiple (possibly different) OSs in parallel on the same hardware. As a result, the hardware platform is partitioned into one or more logical units called virtual machines (VMs).

The following requirements for a VMM have been defined [PG74]:

- **Equivalence**: Running an application inside a virtual machine must be *equivalent* to running the same application on the underlying hardware.
- **Control**: The VMM must control and synchronize the access of VMs to hardware resources.
- **Isolation**: VMs must be isolated from each other with the purpose of ensuring stability (the crash of a VM should not affect another VMs), security (a possibly compromised VM shouldn't grant access to other VMs) and data consistency.
- **Performance**: The performance overhead caused by virtualization should be minimal, close to "bare metal" performance.
- **Encapsulation**: VMs must exist in form of a file or a directory of files which allows easy migration or cloning of the VM.

## 2.3 Classification

From an architectural point of view, the different virtualization approaches can be categorized into:

- Full Virtualization
- OS-Layer Virtualization
- Hardware-Layer Virtualization

For the purpose of this report the focus will be only on the Intel x86 architecture.

### 2.3.1 Full Virtualization

In this approach, the VMM is also called virtual machine manager and runs on top of a host operating system, commonly as an application in userspace. The result is that, in the VMs, the applications and the guest OS run on top of a virtual hardware provided by the VMM. This architecture can be observed in Figure 1.

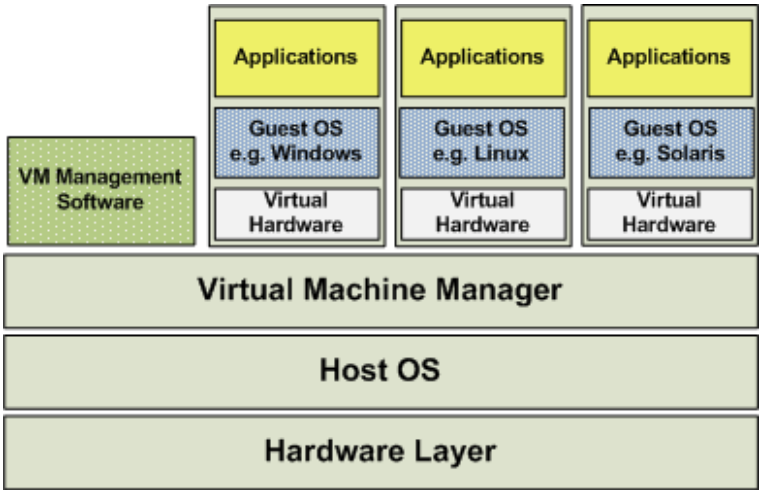


Figure 1: Full Virtualization

VMware Workstation[1], Parallels [2] and Virtual PC[3] are commercial desktop virtualization products that use this approach, with VMware Workstation being the most popular. VMware refers to this approach as Hosted Virtual Machine Architecture, with VMware Workstation installing like a normal application on top of the host OS. According to [SVL01], the application portion (VMApp) uses a driver loaded in the host OS (VMDriver) to establish a privileged VMM that runs directly on the hardware. The VMDriver then facilitates the transfer between the host world and the VMM world. While the CPU virtualization is handled by the VMM, when an I/O in the guest OS occurs, the VMM switches to the host world, where the VMApp performs the I/O on behalf of the virtual machine.

The main advantage of this approach is that it very easy to use. A common user can install a software product like VMware Workstation just like any other software product on its OS of choice. Inside VMware Workstation, a guest OS can be installed and used just like

[1]http://www.vmware.com/products/ws/
[2]http://www.parallels.com/
[3]http://www.microsoft.com/windows/products/winfamily/virtualpc/default.mspx

it would be running directly on hardware. The main disadvantage of this approach is the poor performance, which can be up to 30% less than when running directly on hardware. Since usability is more important than performance when it comes to the desktop market, it is easy to understand why products using this approach have been so successful in the desktop sector. However, when it comes to servers, performance plays a key role, with ease of use and installation being less relevant. This is why the server market requires another virtualization solution.

### 2.3.2 OS-Layer Virtualization

Also known as Single Kernel Image (SKI) or container-based virtualization, this concept implements virtualization by running more instances of the same OS in parallel. This means that not the hardware but the host OS is the one being virtualized. The resulting VMs all use the same virtualized OS image. This architecture is presented in Figure 2. Here, the virtualized OS image is called virtualization layer.

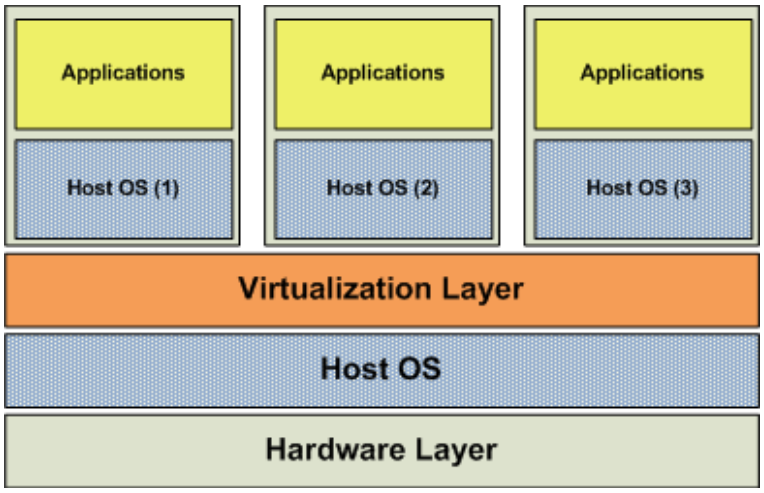


Figure 2: OS-Layer Virtualization

Among products that use this approach are Virtuozzo[4] and its open source variant OpenVZ[5], Solaris Container[6], BSD Jails[7] and Linux VServer[8]. All these products are commonly used in web hosting, high performance computing (HPC) clusters and grid

[4]http://www.swsoft.com/en/products/virtuozzo/
[5]http://openvz.org/
[6]http://www.sun.com/bigadmin/content/zones/
[7]http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/jails.html
[8]http://linux-vserver.org/Welcome_to_Linux-VServer.org

computing. This thin architecture eases the administration of the system, allowing system administrators to assign resources such as memory, CPU guarantees and disk space both when creating a VM as well as dynamically at runtime. When compared to other server virtualization solutions, OS-layer virtualization tends to be more efficient and fails only by little to provide the same isolation [SPF+07]. Yet this approach has one but big drawback: since the VMs use the same kernel as the host OS, the guest OS must be the same as the host OS (and such, it is not possible to run e.g. Windows on top of Linux).

### 2.3.3 Hardware-Layer Virtualization

This approach is commonly used on the server market due to its high virtual machine isolation and performance. Here, the VMM runs directly on hardware, controlling and synchronizing the access of the guest OSs to the hardware resources. Figure 3 depicts this architecture.

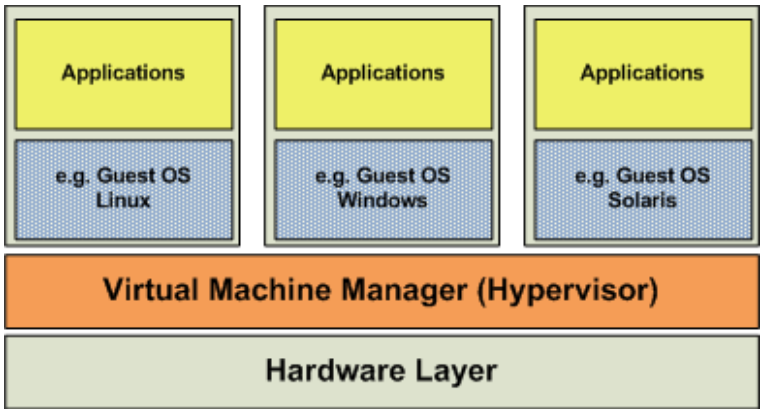


Figure 3: Hardware-Layer Virtualization

VMWare ESX Server[9] and Xen[10] are the main two competing VMMs that use this approach. Since the x86 architecture was not developed with virtualization in mind, and thus it is not what Popek and Goldberg would refer to as a virtualizable architecture [PG74], new techniques were developed to implement CPU virtualization. Paravirtualization is the technique used by Xen which provides a virtual machine interface representing a slightly modified copy of the underlying hardware, where the nonvirtualizable portions of the x86 original instruction set are replaced with their easily virtualized equivalents. This means that the guest operating systems running on top of this interface must be ported to use the slightly modified instruction set [RG05]. On the other hand, the applications running on

[9] http://www.vmware.com/products/vi/esx/
[10] http://xensource.com/

top of the guest OS don't require any modifications. Although the cost of porting an OS to Xen is relatively low [BDF+03], this is still a big disadvantage of the paravirtualization approach. The approach used by VMWare ESX avoids this problem by performing on-the-fly binary translation of privileged code (kernel code) which replaces the nonvirtualizable instructions with a code that can be run directly on the CPU [AA06]. A caching system is then used to increase the performance.

## 2.4 Case study: Xen

Xen started as an open-source project at the University of Cambridge (UK) and while its code base is still open-source, it has now grown up into a commercial product marketed by XenSource. This subsection provides a description of how Xen approaches virtualization.

### 2.4.1 The Virtual Machine Interface

As previously mentioned, Xen uses a technique called paravirtualization. The difference between paravirtualization and full system virtualization is that, while full system virtualization provides an identical copy of the underlying hardware, paravirtualization provides a slightly different copy of the hardware. This modified copy of the hardware is referred to as the virtual machine interface and consists of three main subsystems: memory management, CPU and device I/O. The following paragraphs describe how Xen virtualizes these subsystems.

**Virtualizing Memory** is considered to be the most difficult part of paravirtualizing the x86 architecture [BDF+03], both in terms of mechanisms required in the hypervisor, as well as modifications required to port a guest OS. The approach used by Xen is to provide the guest OS with direct read-only access to the hardware page tables, while page table updates are passed to Xen via a hypercall. A hypercall represents a synchronous communication mechanism between a domain (virtual machine) and Xen. It is then the job of the hypervisor is validate these updates to ensure safety. Since entering the hypervisor through hypercalls is an obvious performance overhead compared to direct updates, a guest OS may queue updates before applying an entire batch with a single hypercall. To avoid a TLB flush when entering and leaving the hypervisor, the top 64MB of each address space are reserved to Xen and cannot be accessed by any guest OS. However, this is no problem for the applications running on the guest OS, since none of the common x86 application binary interfaces (ABIs) use this address region.

**Virtualizing CPU** implies inserting a hypervisor below the OS and thus violates the common assumption that the OS is the most privileged part of the system. To guarantee domain isolation, the guest OS must run at a lower privilege level. Since the x86 architecture provides four distinct privilege levels (referred to as ring 0-3), the code of a Xen guest

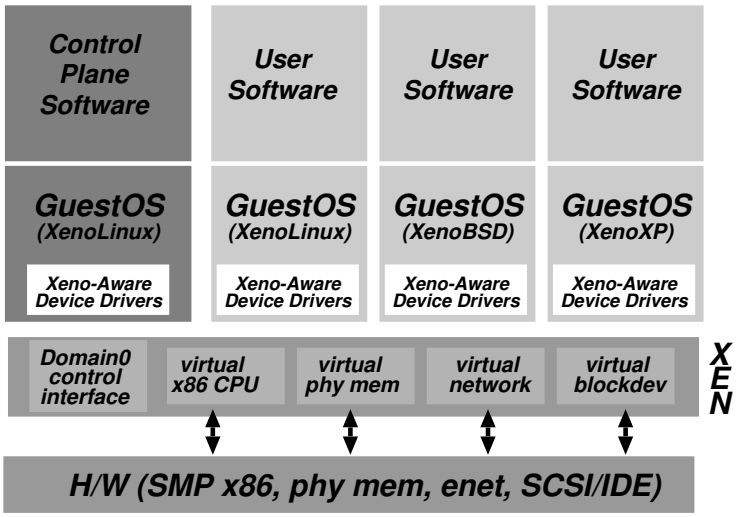


Figure 4: The structure of a machine running Xen [BDF+03]

OS, originally running in ring 0, must be ported to run in ring 1, while Xen becomes the most privileged entity of the system by running in ring 0. This is possible because none of the modern OSes use rings 1 and 2, with applications executing in ring 3. Because the ported guest OS runs in ring 1, privileged instructions cannot run directly on the CPU and need to be paravirtualized and executed by Xen. Since more domains can run concurrently on the same CPU, they need to be scheduled. For this purposes, Xen uses the Borrowed Virtual Time (BVT) scheduling algorithm [DC99].

**Virtualizing Device I/O** is achieved by transferring I/O data to and from a domain via Domain0 and Xen by using shared memory, asynchronous buffer-descriptor rings [BDF+03]. This mechanism is used to pass buffer information vertically trough the system and allows the hypervisor to perform validation checks. An event notification mechanism is used to deliver asynchronous notifications from a device to a domain thus replacing device interrupts.

### 2.4.2 Control and Management

When Xen was developed, one of the goals was to separate mechanism from policy. As a result, the hypervisor itself provides only the basic control operations, exporting them by means of a control interface to an authorized domain called *Domain0*. This domain is created at boot time and is the place where the application management software can be hosted. This architecture is presented in Figure 4. Through the control interface, Domain0 is able to create and terminate other domains, adjust scheduling parameters,

memory allocation and create virtual network interfaces and block devices.

# 3 Managing virtual machines

## 3.1 Management

For the purpose of this report, only the management capabilities of Xen and VMWare ESX are described. The first part describes the management interfaces provided by the two VMMs, while the latter presents existent management software, including third-party software, that uses these management interfaces.

### 3.1.1 Management interfaces

**Xen** has recently finished its 1.0 version of the Xen-API[11] and included it in the Xen 3.1 release. This management API is aiming at integrating systems running Xen with enterprise management solutions. Figure 5 shows the Xen management architecture. On top of the Xen hypervisor runs Xend, the Xen daemon, which controls and supervises the execution of the unprivileged domains. One of the requirements of the Xen-API was to allow remote access and control. The result was the Xen-RPC layer which consists of a set of RPCs that use a wire format based on XML-RPC and runs on top Xend. Based on Xen-RPC, bindings for various languages have been developed or are in the process of being developed. Bindings for Python, C (libvirt) and an Apache XML-RPC-based binding for Java have already been made available.

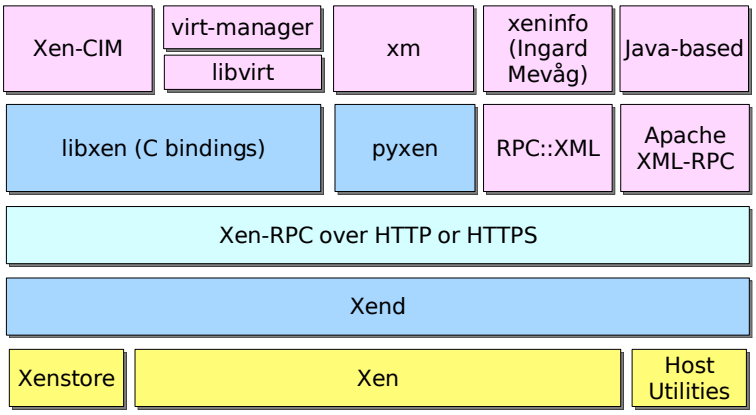


Figure 5: The Xen Management Architecture [Mel07]

Although the Xen-API already reached version 1.0, there integration with enterprise management solutions still requires the development of a CIM integration interface. The

[11] http://wiki.xensource.com/xenwiki/XenApi

Xen-CIM project[12] works on an open source implementation of a DMTF[13] CIM provider for Xen which uses the Xen-API. The DMTF System Virtualization, Partitioning, and Clustering Working Group (SVPC WG) is a standardization body which is currently working on defining models for virtualization and resource allocation which are not directly bound to any product.

**VMware** has already developed several management APIs. For managing a VMware infrastructure, two APIs have been developed: the VMware infrastructure SDK[14] provides developers of third-party tools a Web Service interface to the Virtual Center, which controls the ESX Server. The other infrastructure API is the CIM SDK[15] which allows managing the ESX Server via a CIM interface. Besides these APIs, a guest SDK[16] can be used to monitor guests running inside an ESX Server. The management capabilities of ESX Server clearly overpower the ones of Xen, at least for the time being.

### 3.1.2 Management software

In the following paragraphs, several management software products used for the management of virtual machine-based environments are described. Like before, the focus is only on management tools for Xen and VMware ESX. Note that only the products offering a larger amount of features have been selected.

**Enomalism** [17] is an open source, web-based management tool for Xen virtual machines. The feature list of Enomalism includes:

- Monitoring interface
- Resource management (like for example dynamic memory allocation)
- Live migration
- Disk management tools
- SSH client and VNC virtual desktop
- Virtual Appliance Management (package management)
- An open API for integration with third party products

[12] http://wiki.xensource.com/xenwiki/XenCim
[13] http://www.dmtf.org/
[14] http://www.vmware.com/support/developer/vc-sdk/
[15] http://www.vmware.com/support/developer/cim-sdk/
[16] http://www.vmware.com/support/developer/guest-sdk/
[17] http://www.enomalism.com/


Figure 6: Screenshot of the Enomalism Management Console

**Virt-manager** [18] is the open source virtual machine manager developed by Red Hat. Although originally designated to manage Xen systems, Virt-manager can also be used to manage Qemu[19] and KVM[20] guests. This is possible because Virt-manager was build on top of libvirt[21], a virtual machine management API. The list of features is quite similar with the ones of Enomalism, with few notable exceptions like the lack disk management tools or support for live migration.

**Virtual Iron** [22] is a commercial virtualization product that is based on the Xen hypervisor. The Virtual Iron Virtualization Manager includes a web-based GUI for the management of a virtual environment. The main management feature of Virtual Iron is its policy-driven resource and workload management capability. This allows the automatic management of resources and makes the difference between the Virtual Iron Virtualization Manager and products like Enomalism and Red Hats Virt-Manager. On the other hand, the fact that it is not open-source and its free version is stripped of some features makes this product less attractive as it would otherwise be.

**VirtualIQ** [23] is another commercial product that provides a feature set comparable with Virtual Iron. The main difference between the two is that while Virtual Iron runs on top of Xen, Virtual IQ runs inside of a virtual machine and is platform agnostic, being able to run on top of Microsoft Virtual Server, VMware ESX and Xen.

**Novell** [24] has been intensifying its work lately in the field of virtualization. As such, its release of the SUSE Linux Enterprise Server 10 SP1 includes a broad support for managing and provisioning Xen virtual machines. Also its management product, Novell ZENworks[25], has been extended with virtualization management capabilities. The Novell ZENworks Orchestrator addresses the automatic life-cycle management of virtual machines.

**Xen Enterprise** [26] is a commercial product developed by XenSource. It is build on top of Xen and adds features like easy installation, increased performance and a management console. The management capabilities of XenEnterprise are mainly focused on monitoring, life-cycle management and resource management of CPU, memory, disk and network I/O for QoS.

[18] http://virt-manager.et.redhat.com/
[19] http://fabrice.bellard.free.fr/qemu/
[20] http://kvm.qumranet.com
[21] http://libvirt.org/
[22] http://www.virtualiron.com/products/virtualization.cfm
[23] http://www.toutvirtual.com/solutions/solutions.php
[24] http://www.novell.com/linux/virtualization/
[25] http://www.novell.com/products/zenworks/
[26] http://xensource.com/products/xen_enterprise/index.html

**VMware's Virtual Center** [27] was designed to manage a VMware virtual infrastructure. Its feature list includes support for provisioning, monitoring, automation through task scheduling and alerting, automated optimization and live migration. Like before, VMware seems to be one step in front of its competitors.

[27] http://www.vmware.com/products/vi/vc/

# 4 Autonomic computing

## 4.1 The vision of autonomic computing

In 2001, IBM released a document [Hor01] presenting its vision of autonomic computing. In this article, the author observed that the ever increasing complexity of computing systems will cross the borders of human capabilities, leading to systems that are too complex to be administrated by human beings. System administrators will no longer be able to install, configure, maintain and optimize these computing systems of the future. The solution to this problem is developing a technology that allows computing systems to manage themselves. This solution is referred to as autonomic computing.

**Autonomic computing** means that computing systems can manage themselves based on high-level objectives set by system administrators. As such, system administrators are not completely replaced, but instead need to shift their field of work to the higher levels, like for example developing policies with whom the autonomic system complies. According to [KC03], IBM deliberately chose a term with biological connotation: the autonomic nervous system governs functions like heart rate and body temperature without the interference of the brain. Drawing a parallel between the soon to be achieved complexity of a computing system and the complexity of the human body, it is obvious that, just like the brain alone could not deal with managing the entire complexity of the human body, the system administrator will not be able to manage alone such a complex computing system. As such, autonomic computing should do the same for computing systems as the autonomic nervous system does for the human body.

## 4.2 Taxonomy

Since autonomic computing is a rather new field of research, there is a lack of unity with respect to terms and definitions. What IBM calls autonomic computing is more or less what Intel refers to as proactive computing [Ten00], whereas by organic computing the German research community addresses the same problems but focuses more on hardware aspects [MSMW04]. This sometimes leads to a conflict in terms and definitions. For the purpose of this survey the IBM taxonomy will be used as a backbone and extended with complementary terms and definitions where necessary.

### 4.2.1 Definitions

**Self-management** is considered to be the essence of autonomic computing and represents the process by which computing systems manage themselves. According to [KC03] there are four aspects of self-management:

- *Self-configuration*: the property of a computing system to configure itself automatically with respect to a high-level policy, thus sparing the system administrator from

installing and configuring the system manually. Self-configuration also implies that new components added to the system are incorporated seamlessly.

- *Self-optimization*: the property of a computing system to continuously seek to improve its operation by dynamically changing its parameters at runtime.
- *Self-healing*: the property of a computing system to detect, diagnose and repair local problems caused either by software of by hardware failures.
- *Self-protecting*: the ability of a computing system to defend itself in the face of an malicious attack or cascading failures. Besides these, self-protection also means that the system is capable of predicting possible problems based on logs and reports.

Yet these are not the only "self-X" properties found in computer science publications. Terms like self-organization, self-inspection, self-repairing, self-monitoring, self-testing and many others have all been used in academic research. These are all referred to as "self-X" properties. In [FZ05], the authors provide a term frequency statistic of each of these "self-X" proprieties over a range of computer science publications. Along self-management, another frequently appearing term is self-organization.

**Self-organization** is a highly interdisciplinary term. Definitions for self-organization have been provided by biologists, physicists, chemists, philosophers and recently computer scientists. The latter define self-organization as the capability of a computing system to adapt itself to a dynamic environment without outside control. The presence of a central coordination component in such systems is generally excluded. Self-organization is considered to be a special propriety of a self-managing system [MWJ+07].

### 4.2.2 Autonomic elements

In their vision, Kephart and Chess see autonomic systems as an interactive collection of autonomic elements [KC03]. This basic component of the system is capable of managing both its internal behavior, as well as its relationship with other autonomic elements. Together they collaborate to achieve the goals set by either a higher-ranking autonomic element or by a human system administrator.

From a structural point of view, an autonomic element is composed of one or more managed elements and only one autonomic manager that controls the managed elements. This structure is shown in Figure 7. The autonomic manager monitors the managed element, analyzes the obtained information and, based on its knowledge base, constructs and executes plans that modify the behavior of the managed element. The roadmap towards autonomic computing sees the development and attachment of autonomic managers to the already existing manageable elements.

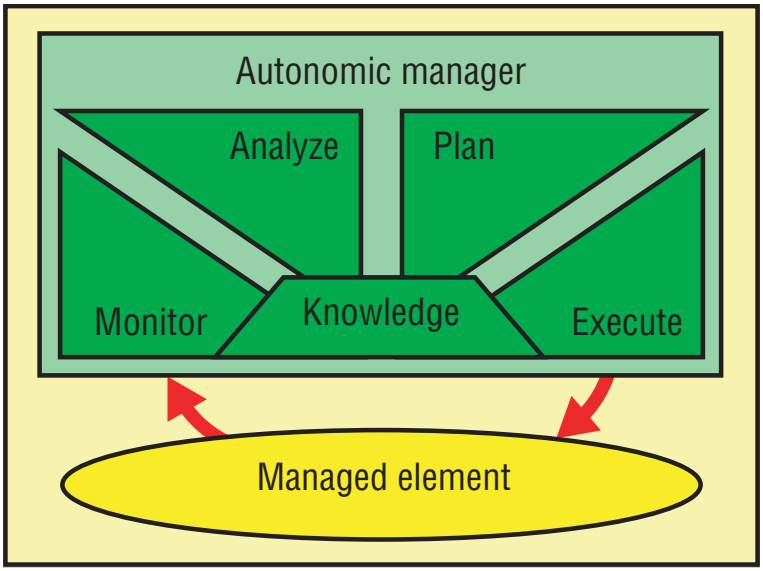


Figure 7: The structure of an autonomic element [KC03]

### 4.3 General approaches to autonomic computing

Over the past years, researchers looked for techniques that can be used in developing autonomic computing systems. This subsection describes different approaches used in autonomic computing.

**Control theory** has been frequently used in autonomic computing systems [Hel04], [AKN04]. While planning and optimization can be accomplished using other approaches, the change in conditions that appear at runtime cannot be planned and requires a real-time approach. Control theory has been proven to work in achieving this purpose, although modifications to the classical control theory used in engineering were required.

**Agent-based computing** [Jen00], is a field that has been well studied in the past and fits well to the autonomic elements paradigm. However, most of the agent-based theories have focused on single-agent learning theories, which does not fit in the context of a complex autonomic system where agents adapt to other agents [Tia03]. As such, the theory of agent-based computing needs to be extended to fit the autonomic computing, multi-agent scenario.

**Biology inspired** approaches have proven to be successful in the past. Neural networks and genetic algorithms found lots of applications in computer science. These techniques can also be applied to autonomic computing. Genetic algorithms, for example, can be

used to achieve self-optimization. Besides these classic techniques, new ones have been developed [BBD+06]. Stigmergy, for example, is a method of indirect communication between entities using the surrounding environment. The term was introduced by Pierre-Paul Grasse in 1959 after studying the behavior of termites, which communicated to each other by leaving pheromone trails. This method of indirect communication can be used to develop self-organizing computing systems [BC04]. Another technique inspired by biology is cell-oriented programming. In [GED02], the authors developed a biologically inspired programming model for self-healing computing systems.

**Behavior analysis** represents the art of studying the behavior of a system in order to develop a statistical model that can be used to predict future behavior and thus to detect and prevent possible problems. Negotiation theory and microeconomics are as well possible sources of inspiration in implementing autonomic computing systems.

### 4.4 Self-management of virtual machine-based environments

It has been previously argued that a complex computing system is hard and sometimes impossible to be managed by a human system administrator. The system needs to be able to manage itself, with no or little interference from the system administrator. Unfortunately, virtual machine-based environments nowadays are manually managed by system administrators using GUI-based management tools that do not provide any automation capabilities. Yet virtualization itself can be used to split a complex system in multiple homogenous virtual machines which can be easily managed and physically moved around the system, thus simplifying the autonomic task. The lack of such solutions shows that the self-management of virtual machine-based environments is a field of research where there is still a lot of work to be done. This subsection presents a survey of preliminary academic work related to the self-management of virtual-machine based envrionment.

**Policy-based** approaches have been a popular way to manage computing systems. Virtualization-based computing systems are no exception. In [RRX+06], Ruth et al. developed a system called VIOLIN, based on a nanoHUB [28] infrastructure, which uses the Xen hypervisor for virtualization. Each physical machine of the system hosts one or more virtual machines and has one *monitor daemon*. The *monitor daemon* monitors the CPU utilization, memory and virtual machine resource allocation of each physical machine. Besides monitoring, the *monitor daemon* is also able to allocate resources to local virtual machines. Results from all the monitoring daemons are gathered by an *adaption manager*, which then computes a global view of available and allocated resources. Furthermore, the *adaption manager* can control virtual machine resource allocation, in this case memory and CPU allocation, through the monitoring daemons. Thus, the *adaption manager* is an

[28] http://www.nanohub.org

intelligent agent which acts on behalf of a *resource reallocation policy* defined by a system administrator. The *resource reallocation policy* uses a heuristic algorithm that aims at increasing the performance of the system but doesn't intent to provide an optimal resource allocation to virtual machines. Migrating virtual machines between physical machines is also used to optimize resource allocation. Results obtained by the authors show an increase of performance and resource utilization. Another policy-based approach has been used by Grit et al. [GIYC]. Here, the authors used Shirako[29], a Java-based toolkit for resource leasing services for what they refer to as autonomic orchestration. Autonomic orchestration involves both provisioning and managing a system or in this case a virtual environment. The paper analyzes the challenges of developing policies for on-demand and adaptive allocation of resources in a shared (virtual) infrastructure and proposes extensions to the Shirako toolkit for the self-management of Xen VMs.

**Control theory** has also been used for the dynamic allocation of resources in virtual environments. In [WZPS07], Padala et al. have used classical control theory to develop a feedback-driven resource control system that works at the VM level. Also this system uses Xen as a hypervisor but focuses only on CPU scheduling. The controller uses a two-layer architecture: a *utilization controller* per virtual machine, which computes the CPU entitlement of the VM, and an *arbiter controller* which ultimately determines if the CPU entitlements for all virtual machines can be satisfied and if not uses a QoS differentiation mechanism to determine which CPU entitlements will be satisfied. Another paper that uses control theory for managing virtual environments is [ZBG+05]. Here, the authors introduce the concept of *friendly virtual machines*. Their approach is that the VMs themselves should be adaptable. To implement this, the authors developed control-theoretic models for application adaption using Additive-Increase/Multiplicative-Decrease (AMID) rules. Furthermore, they modified a User-Mode Linux (UML) instance to test this approach with respect to efficiency and fairness. Although the results obtained were good, because only one UML instance was used it remains unclear how this approach works for an infrastructure where resources are distributed. In the summer semester of 2007, Mark Bommert worked on a project at the University of Applied Sciences Wiesbaden under the supervision of Prof. Dr. Reinhold Krger. In this project, a proportional controller was used to dynamically allocate memory to virtual machines running on top of Xen.

**Utility functions** have been previously used in autonomic systems [WTKD04]. Menascè et al. used autonomic computing techniques in [MB06] for dynamic CPU allocation for virtual machines. The dynamic CPU allocation is achieved by optimizing a utility function. No actual virtualization technology was used to implement this approach, yet the authors obtained promising results by means of simulation.

[29] http://www.cs.duke.edu/nicl/cereus/shirako.html

**Time series analysis** has been used for decades to forecast the behavior of a system. In [BKB07], Bobroff et al. developed a mechanisms for the dynamic migration of VMs. Their algorithm uses measurements of historical data to forecast future demands by means of time series analysis. Furthermore, depending on future demands, the algorithm uses **bin packing heuristics** to migrates VMs to other physical machines to provide Service Level Agreement (SLAs) probabilistic guarantees.

# 5 Conclusion

The purpose of this report was to introduce the reader to the autonomic computing and virtualization concepts. Both of them are of major importance for the IT-sector and could be successfully combined to reach the ideals of the autonomic computing vision. Researchers have already started applying autonomic computing techniques to manage virtual environments. Yet this work is only at the beginning, researchers basically testing techniques known to work on a new problem. There is still a long way to a fully autonomic virtual machine-based environment. Work needs to be done with respect to the architecture of such a system because existent work only uses simple architectures to test their approach, while autonomic computing targets complex environments. Future work should also see how different techniques can be combined, for example a control-theory-based approach for a real-time reaction to an SLA violation and an optimization technique for optimizing the overall performance and resource utilization of the system.

## References


[AA06] Adams, Keith ; Agesen, Ole: A comparison of software and hardware techniques for x86 virtualization. In: *ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*. ACM Press. – ISBN 1595934510, 2–13

[AKN04] Abdelwahed, Sherif ; Kandasamy, Nagarajan ; Neema, Sandeep: A control-based framework for self-managing distributed computing systems. In: *WOSS '04: Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*. ACM Press. – ISBN 1581139896, 3–7

[BBD+06] Balasubramaniam, Sasitharan ; Barrett, Keara ; Donnelly, William ; Meer, Sven van d. ; Strassner, John: Bio-inspired Policy Based Management (bioPBM) for Autonomic Communication Systems. In: *POLICY '06: Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06)*. IEEE Computer Society. – ISBN 0769525989, 3–12

[BC04] Barron, Peter ; Cahill, Vinny: Using Stigmergy to Co-Ordinate Pervasive Computing Environments. In: *WMCSA '04: Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'04)*. IEEE Computer Society, 62–71

[BDF+03] Barham, Paul ; Dragovic, Boris ; Fraser, Keir ; Hand, Steven ; Harris, Tim ; Ho, Alex ; Neugebauer, Rolf ; Pratt, Ian ; Warfield, Andrew: Xen and the art of virtualization. In: *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. ACM Press. – ISBN 1581137575, 164–177

[BKB07] Bobroff, Norman ; Kochut, Andrzej ; Beaty, Kirk: Dynamic Placement of Virtual Machines for Managing SLA Violations, 119–128

[DC99] Duda, Kenneth J. ; Cheriton, David R.: Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In: *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles* Bd. 33. ACM Press. – ISSN 0163–5980, 261–276

[FZ05] Fromm, Jochen ; Zapf, Michael: Selbst-Eigenschaften in verteilten Systemen. In: *Praxis der Informationsverarbeitung und Kommunikation (PIK)* 2005 (2005), oct, Nr. 4, 189-198. `http://www.vs.uni-kassel.de/publications/2005/FZ05`

[GED02] George, Selvin ; Evans, David ; Davidson, Lance: A biologically inspired programming model for self-healing systems. In: *WOSS '02: Proceedings of the first workshop on Self-healing systems*. ACM Press. – ISBN 1581136099, 102–104

[GIYC] Grit, Laura ; Irwin, David ; Yumerefendi, Aydan ; Chase, Jeff: Virtual Machine Hosting for Networked Clusters: Building the Foundations for Autonomic Orchestration

[Gol74] Goldberg, Robert P.: Survey of Virtual Machine Research. In: *IEEE Computer Magazine* 7 (1974), June, S. 34–45

[Hel04] Hellerstein, J. L.: Self-managing systems: a control theory foundation. In: *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, 708+

[Hor01] Horn, Paul: *Autonomic computing: IBM's Perspective on the State of Information Technology*. 2001

[Jen00] Jennings, N. R.: On agent-based software engineering. In: *Artificial Intelligence* 117 (200), S. 277–296

[KC03] Kephart, J. O. ; Chess, D. M.: The Vision of Autonomic Computing. In: *Computer* 36 (2003), S. 41–50

[MB06] Menasce, Daniel A. ; Bennani, Mohamed N.: Autonomic Virtualized Environments. In: *ICAS '06: Proceedings of the International Conference on Autonomic and Autonomous Systems*. Washington, DC, USA : IEEE Computer Society, 2006. – ISBN 0–7695–2653–5, S. 28

[Mel07] Mellor, Ewan: *The Xen-API*. April 2007

[MSMW04] Mller-Schloer, Christian ; Malsburg, Christoph von d. ; Wrt, Rolf P.: Organic Computing. In: *Informatik-Spektrum* 27 (2004), Nr. 4, 332–336. `http://dx.doi.org/10.1007/s00287-004-0409-6`. – DOI 10.1007/s00287–004–0409–6

[MWJ+07] Mühl, Gero ; Werner, Matthias ; Jaeger, Michael A. ; Herrmann, Klaus ; Parzyjegla, Helge: On the Definitions of Self-Managing and Self-Organizing Systems. In: Braun, T. (Hrsg.) ; Carle, G. (Hrsg.) ; Stiller, B. (Hrsg.): *KiVS 2007 Workshop: Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme (SAKS 2007)*. VDE Verlag. – ISBN 978–3–8007–2980–7, 291–301

[PG74] Popek, Gerald J. ; Goldberg, Robert P.: Formal requirements for virtualizable third generation architectures. In: *Commun. ACM* 17 (1974), July, Nr. 7, 412–421. http://dx.doi.org/10.1145/361011.361073. – DOI 10.1145/361011.361073. – ISSN 0001–0782

[RG05] Rosenblum, M. ; Garfinkel, T.: Virtual machine monitors: current technology and future trends. In: *Computer* 38 (2005), Nr. 5, 39–47. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1430630

[RRX+06] Ruth, P. ; Rhee, Junghwan ; Xu, Dongyan ; Kennell, R. ; Goasguen, S.: Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure. In: *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, 5–14

[SPF+07] Soltesz, Stephen ; Poetzl, Herbert ; Fiuczynski, Marc E. ; Bavier, Andy ; Peterson, Larry: Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In: *EuroSys '07: Proceedings of the 2007 conference on EuroSys*. ACM Press. – ISSN 0163–5980, 275–287

[SVL01] Sugerman, Jeremy ; Venkitachalam, Ganesh ; Lim, Beng-Hong: Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In: *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*. USENIX Association. – ISBN 188044609X, 1–14

[Ten00] Tennenhouse, David: Proactive computing. In: *Commun. ACM* 43 (2000), May, Nr. 5, 43–50. http://dx.doi.org/10.1145/332833.332837. – DOI 10.1145/332833.332837. – ISSN 0001–0782

[Tia03] Tianfield, H.: Multi-agent based autonomic architecture for network management, 462–469

[WTKD04] Walsh, W. E. ; Tesauro, G. ; Kephart, J. O. ; Das, R.: Utility functions in autonomic systems. In: *Autonomic Computing, 2004. Proceedings. International Conference on*, 70–77

[WZPS07] Wang, Zhikui ; Zhu, Xiaoyun ; Padala, Pradeep ; Singhal, Sharad: Capacity and Performance Overhead in Dynamic Resource Allocation to Virtual Containers. (2007)

[ZBG+05] Zhang, Yuting ; Bestavros, Azer ; Guirguis, Mina ; Matta, Ibrahim ; West, Richard: Friendly virtual machines: leveraging a feedback-control model for application adaptation. In: *VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*. ACM Press. – ISBN 1595930477, 2–12