Initial Monitoring of Virtual Networks

Deliverable D1.1

Autonomic Internet (AutoI) Project

FP7-ICT-2007-Call 1 - 216404

Abstract
This document describes the AutoI Virtualisation Plane and its interactions with other AutoI planes.

Document Properties

Properties[1]:

Document Number[2]:	D1.1
Document Title:	Initial Monitoring of Virtual Networks
Document responsible:	Andreas Berl, Andreas Fischer
Author(s) / Editor(s):	Andreas Berl, Andreas Fischer, Lefteris Mamatas, Alex Galis, Zohra Boudjemil, Steven Davy
Reviewed by:	Steven Davy (WIT), Alex Galis (UCL)
Dissemination Level[3]:	PU
Status of the Document:	Final Version
Version:	2.0
This document has been produced in the context of the Autonomic Internet (AutoI) Project. The AutoI Project is part of the European Community's Seven Framework Program for research and is as such funded by the European Commission. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors view.	

1 Input of Title, Date, Version, Target dissemination level, Status via *"File /Properties/Custom"* in the Word menu

2 Format: FP7-ICT-2007-CALL1-216404 AutoI/<Deliverable number>
Example: FP7-ICT-2007-CALL1-216404 AutoI/D6.1

3 Dissemination level as defined in the EU Contract:
PU = Public
PP = Distribution limited to 6th FP participant

Revision History

Revision	Date	Issued by	Description
V0.1	13/10/08	Andreas Fischer	1st draft ToC & Takeover from D6.1
V0.2	17/11/08	Andreas Fischer	Some smaller revisions
V0.3	25/11/08	Andreas Fischer	Incorporated UCL additions
V0.4	26/11/08	Andreas Fischer	Added section on vCPI
V0.5	27/11/08	Andreas Fischer	Added some text on security
V0.6	28/11/08	Andreas Fischer	Added UCL provided info on vSPI
V1.0	09/12/08	Andreas Fischer	First Review Version
V1.1	17/12/08	Andreas Fischer	Second Review Version
V1.2	17/12/08	Andreas Fischer	Incorporated last minute changes by UCL
V1.3	22/12/08	Alex Galis	Alex' review and comments
V1.4	08/01/09	Andreas Fischer	Pre-final version for submission
V1.5	12/01/09	Andreas Fischer	Final version for submission
V2.0	12/01/09	Andreas Fischer	Really final, last minor editing

Table of Contents

Executive Summary

The AUTOI project is developing a solution for the manageability of the Future Internet under the concept of five planes (OSKMV) - Virtualisation Plane (VP) [20], Management Plane (MP) [23], Knowledge Plane (KP) [23], Service Enablers Plane (SP) [24] and Orchestration Plane (OP) [21]. The overall architectural model of the AutoI project was depicted in D6.1 published in August 2008 and D4.1 published in December 2008.

This document is directly linked to address Objective 2 - the virtualisation of Communication resources and Objective 7 - control of virtual resource of the project work plan.

This document describes the AutoI Virtualisation Plane and its interactions with other AutoI planes. First a state of the art of virtualisation is given. Then methods of virtualisation are selected as basis for the resource virtualisation of the AutoI virtualisation plane. It is shown, how routers can be virtualised by using the method of system virtualisation. Virtual routers are described and also the virtual networks that can be created by using system virtualisation.

The interaction between the virtualisation plane and other AutoI planes is realised by the provision of two different interfaces, vCPI and vSPI. The vCPI allows a localized monitoring and management of virtual resources (e.g. management of virtual routers on a single hardware), whereas the vSPI is an interface that allows a monitoring and configuration of virtual resources that is based on a more global view of the system. Groups of virtual resources can be controlled via the vSPI, including components and whole topologies. Besides describing these two interfaces, this document also considers the programmability of virtualisation plane functions. Programmability is necessary to achieve design of the virtual plane that is flexible enough to adapt to unforeseen changes that may occur in the future.

Future work will be devoted to the final design of the virtualisation plane interfaces (vCPI and vSPI) that will to be prototyped using the JAVA programming language and XEN as virtual machine monitor (VMM). The programmability of virtual resources will be further developed and prototyped.

Abbreviations

AIM	AutoI Information Model
AMS	Autonomic Management System
API	Application Programming Interface
AutoI	Autonomic Internet Project
CIM	Common Information Model
CLI	Command Line Interface
CPU	Central Processing Unit
DMTF	Distributed Management Task Force
DNS	Domain Name System
DHT	Distributed Hash Tables
DOC	Distributed Orchestration Component
Dom0	XEN privileged domain
DomU	XEN unprivileged domain
DSL	Domain Specific Languages
DSM	Domain Specific Modelling
DTD	Document Type Definition
EE	Execution Environment
FI	Future Internet
IP	Internet Protocol
ISA	Instruction Set Architecture
ISP	Internet Service Provider
KP	Knowledge Plane
LAN	Local Area Network
MIB	Management Information Base
MP	Management Plane
NIC	Network Interface Card
OP	Orchestration Plane
OS	Operating System

OSKMV	Autonomic Internet Planes: Orchestration Plane (OP), Service Enablers Plane (SP), Knowledge Plane (KP), Management Plane (MP) and Virtualisation Plane (VP)
OVF	Open Virtualization Format
P2P	Peer-to-Peer
SP	Service Enablers Plane
SSH	Secure Shell
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
vCPI	Virtual Component Programming Interface
VE	Virtual Environment
VL	Virtual Link
VM	Virtual Machine
VMM	Virtual Machine Monitor
VN	Virtual Network
VoIP	Voice over IP
VP	Virtualisation Plane
VPN	Virtual Private Network
VR	Virtual Router
vSPI	Virtualisation System Programmability Interface
WLAN	Wireless Local Area Network
WPA	Wi-Fi Protected Access

1 Introduction

1.1 Motivation and Objectives of the Document

The primary purpose of this document is to describe the functionality of the Virtualisation Plane within the AutoI project. The Virtualisation Plane allows the AutoI infrastructure to be abstracted from the underlying hardware. Hardware routers are virtualised and a common interface is defined that allows clients to access virtual hardware without caring about hardware specific issues.
As such, two goals are followed in this document. On the one hand, the design decisions of the virtualisation plane are discussed, explaining which virtualisation solution is employed and why. Several solutions are presented and discussed with regard to their fitness for the AutoI project. On the other hand, first versions of the interfaces of this plane to other AutoI planes are presented and documented with a clear explanation of all relevant functionality.
This document represents an update of the chapter 3.2 of the deliverable D6.1 published in August 2008.

1.2 Structure of the Document

The rest of the document is structured as follows:

- In section 2 some background on virtualisation is given and current state of the art in virtualisation is discussed. Several virtualisation solutions are compared with each other with regard to the needs of the AutoI project.
- In section 3, the virtualisation approach taken in the AutoI Virtualisation Plane is explained, based upon the findings of section 2.
- In section 4 an introduction is given to the vCPI and the vSPI. These are the two main AutoI virtualisation interfaces.
- In section 5 the programmability of the Virtualisation Plane is discussed.
- In section 6 a detailed introduction is given to the vCPI, the interface dealing with management of single components.
- In section 7 a similar introduction is given to the vSPI, the interface that provides methods to manage the network as a whole.
- Finally in section 8 conclusions are drawn and future work is discussed.

2 Background and State-of-the-Art

2.1 Aggregation or splitting of resources

One possible way to classify different virtualisation techniques is by looking at the interface between the virtual environment and the real hardware used. This opens up two different ways of virtualisation: a n:1 aggregation of resources (clustering) or a 1:n splitting of resources (zoning, partitioning).
A virtual machine (VM) is a software implementation of a computer that executes programs like a real computer. To optimise resource utilisation, the computational and network resources within a site are partitioned into virtual environments (VEs), which are fully isolated runtime environments that abstract away the physical characteristics of the resource and enable sharing.
In the first case, access to several hardware instances is clustered to present one single virtual hardware instance for a virtual environment (VE) to use. Access to hardware is abstracted for the VE – it is not aware of the fact that it actually uses several hardware instances instead of just one. This technique is used for example in grid computing.
In the second case, access to a single hardware instance is multiplexed for several different VEs in order to separate and isolate each one of them from its neighbours. Each of the VEs is completely oblivious of the fact that there are other VEs competing with it for the physical resources.

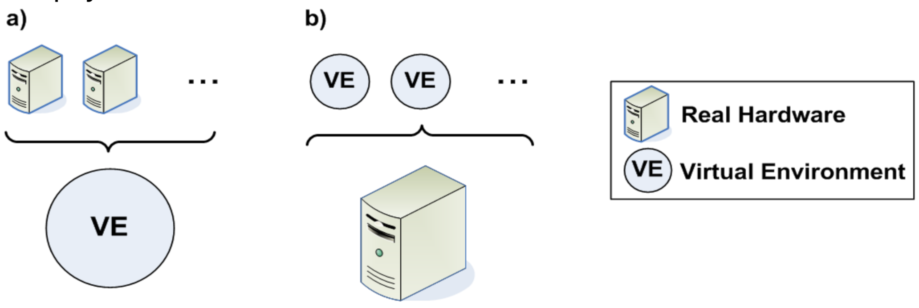


Figure 3.2.1:1 - Aggregation / Splitting of resources

2.2 Process or system virtualisation

Virtualisation techniques can also be classified according to whether they use process virtualisation or system virtualisation.
Process virtualisation is a technique that is actually quite common. It allows several different processes to be isolated from each other by virtualising access to resources for each process. Each process may run conceptually on a virtual CPU, using virtual memory, etc. This concept has been used for a long time in multiprocessing/multiprogramming environments where each process seems to have full control of its CPU and its memory, while in reality it competes with lots of other processes running on the same machine. Typically, the underlying operating system (OS) will mediate access to the physical resources, allocating memory and CPU time for each process as needed.
System virtualisation on the other hand takes this concept one step further. It virtualises a whole system, allowing several instances of an OS, or – possibly – even several different operating systems to be run at the same time on the same hardware. Each OS will have access to its own virtualised hardware, being isolated from all other OS instances.

2.3 Layers of system virtualisation

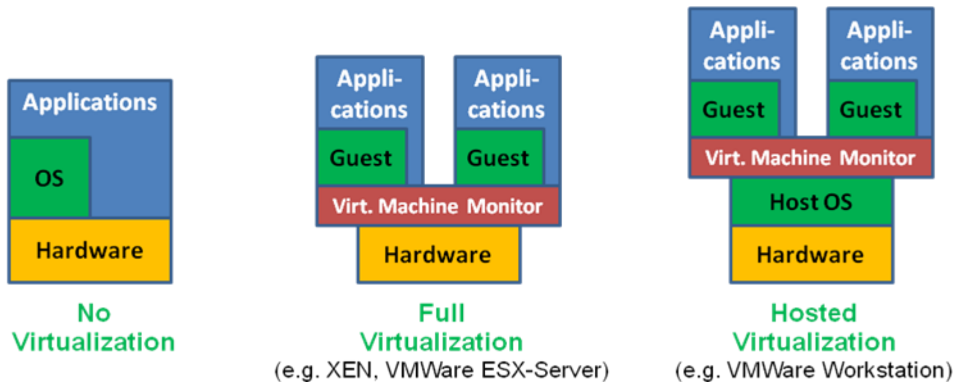


Figure 3.2.1:2 - Full Virtualisation / Hosted Virtualisation

A third way to classify virtualisation techniques is to look at the position of the virtual machine monitor (VMM) (see figure 3.2.1:2). The VMM is the virtualisation software that multiplexes several VEs onto a single hardware instance. In a system without virtualisation, there is obviously no VMM at all.

In a full virtualisation scenario, the VMM acts as kind of a small operating system, sitting directly on top of the hardware and mediating access to it for the hosted VEs. This is the approach used for example by XEN and VMWare ESX-Server [3, 6]. The advantage of this approach is situated in its speed because there is no additional layer to be crossed between VMM and the hardware. However this also comes with a price: the VMM has to communicate with the hardware in order to be able to handle requests from its hosted VEs. This may require the VMM to provide its own drivers and schedulers, possibly leading to the VMM becoming a full blown operating system of its own.

In the hosted virtualisation scenario, the VMM runs as a simple piece of client software within the operating system of the host. This is the approach used for example by VMWare Workstation [4]. In contrast to the previous approach, in this approach the VMM does not need its own drivers for all of the hardware, as it can use the facilities provided by the host OS. On the other hand, performance will suffer in this approach, as system calls have to be handed through yet another layer. Also, the VMM does not have special privileges regarding the resources allocated to it – indeed it will compete for the resources with other processes running on the host OS, possibly leading to a further degradation of service for the hosted VMs.

2.4 Instruction Set Architectures (ISA)

Yet another way to classify different virtualisation techniques is to differentiate by the way an Instruction Set Architecture (ISA) is offered for a virtual environment.

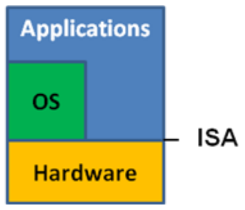


Figure 3.2.1:3 - Instruction Set Architecture

The ISA is the interface between software and hardware (see figure 3.2.1:3). It depends on the hardware architecture – different architectures, like x86, PowerPC or Sparc offer different sets of hardware operations for the operating system to use.
Virtual Machine Monitors (VMMs) have to provide an ISA that can be used by the hosted operating systems within their virtual context. There are basically two possibilities for the VMM: either to replicate the unchanged ISA from the host systems hardware architecture or to emulate a new ISA (possibly that of another, different hardware architecture) and translate system calls from the virtual machines into system calls that are compatible with the underlying hardware environment. In the first case, virtual machine guests are limited to the ISA provided by the hardware, while in the second case any combination of host architecture and guest architecture is possible, albeit – due to the translation process – only with a certain performance penalty.

2.5 Classical Hypervisors and System Emulators

A control program commonly known as "hypervisor" or "virtual machine monitor" (see figure 3.2.1:4) is run on a given hardware platform and simulates one or more computer environments (virtual machines). Classical hypervisors like XEN or VMWare will simply replicate the ISA of their host system for their virtual machines (VMs). Thus, the virtual hardware recognized by the VMs is the same as the real hardware the hypervisor runs on. This is the approach used by software packages like XEN or VMWare.

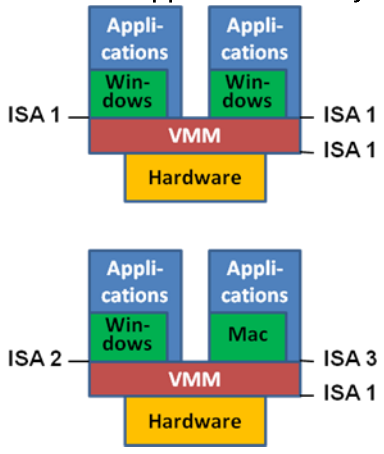


Figure 3.2.1:4 - Hypervisors / System Emulators

System Emulators on the other hand emulate their own hardware environment, dynamically translating between different ISAs. This opens up the possibility to run software that was compiled for a specific architecture on a different architecture. Moreover, this approach also allows running operating systems compiled for different architectures, side by side on the same machine, if the system emulator is able to emulate the different kinds of hardware.

2.6 XEN

2.6.1 Paravirtualisation

XEN introduces a new concept in virtualisation techniques: Paravirtualisation. The x86-hardware platform lacks specific support for virtualisation as there are some critical instructions that will not throw an exception when executed by unprivileged clients. Normally this would prohibit the use of a virtualisation model of [5]. XEN project introduced

Paravirtualisation [2]. This concept requires the guest operating system to be modified to replace the critical instructions with calls to the hypervisor, called (appropriately) “hypercalls”.
This modification allows XEN to act as a hypervisor (i.e. control program run on a given hardware platform and it simulates one or more computer environments / virtual machines). even without the necessary hardware support. A drawback is, that the guest operating system is no longer oblivious to the fact that it runs within a virtual machine. In fact, the modifications necessary in the guest operating system are quite intrusive, prohibiting the use of this technique with a closed source OS.
XEN allows guest operating systems to get direct access to the underlying physical hardware, possibly reserving some hardware for some VMs. This allows for example to assign each VM its own network interface card, physically shielded from cards of other VMs, without having to emulate the respective hardware.

2.6.2 XEN Trick

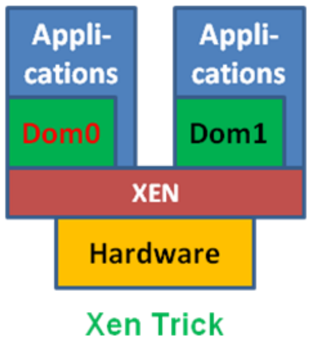


Figure 3.2.1:5 - Xen Trick - privileged (Dom0) and unprivileged (DomU) guests
In order to avoid having to implement full operating system functionality, XEN performs a little trick. The XEN hypervisor still manages the central elements, like CPU and memory access. More uncritical hardware however is managed with the help of one of its guests.
XEN therefore introduces the concept of privileged and unprivileged “Domains” (see figure 3.2.1:5). One of the guests will be the “Dom0” guest. XEN will communicate with that guest and use it to access all hardware not managed by XEN itself. All other guests (“DomU”) still can access the hardware, but their requests will be relayed by XEN through the Dom0 guest.
This approach requires the Dom0 guest to be aware of XEN and implement a control interface to XEN to be able to handle requests originating from other DomU guests. The Dom0 guest is also responsible for providing the ability for users to create and manage new virtual machines via the control interface [2].

2.7 Advantages and Problems of Virtualisation

2.7.1 Advantages

There are several obvious advantages to virtualisation. First, virtual machines offer a great way to implement defined test beds and development environments. By allowing freezing of the current state of a virtual machine it becomes easy to return to a defined starting point if problems occur. Moreover the repetition of predefined scenarios allows tests to be conducted while being certain that the underlying system stays the same in all scenarios.
Second, by having an abstraction layer between the real hardware available and the ISA offered to virtual machines it becomes possible to keep legacy software running in a virtual

environment, even if the original hardware requirements can no longer be met due to failing hardware or missing spare parts. Using a system emulator approach, a VMM can emulate the original legacy hardware on current platforms, providing even higher performance.
Third, virtualisation enables the consolidation of several servers onto one-hardware. Current hardware systems tend to be vastly oversized for their jobs. Combining these jobs allows the better utilization of hardware while still keeping a logical separation of different systems. This will also reduce the amount of hardware necessary, resulting in savings in both physical space and energy requirements.

Finally it is possible to adhere to high availability and security requirements. If a virtual machine experiences high system load, the VMM can simply assign more CPU time or system memory to it. If the real hardware is overloaded itself, the virtual machine can be migrated onto another physical system. By cloning systems it is possible to create "hot standby" environments, which can be used in disaster recovery measures. And last but not least the virtual machines are separated from each other providing compartmentalisation of different tasks for additional security.

2.7.2 Disadvantages

There are, however, also some problems with regard to virtualisation. Additional costs may add up as new hardware has to be bought to cover the increased hardware requirements of the host that will house the virtual machines. If virtual machines should be easily accessed from multiple locations, for example for migration of virtual machines, a significant amount of network storage has to be provided, possibly requiring additional network storage services and administration costs.
Also, as virtual machines are easy to create and to clone, it is likely that more (possibly unused) virtual machines will be created than strictly necessary, eating up additional resources.
Some security problems should not be overlooked. While virtualisation allows compartmentalisation of different tasks, increasing security in that area, it also opens up new or increased security problems in other areas.
First, a Trojan horse could use virtualisation techniques to hide itself from scanners, and indeed two such attempts – called Blue Pill and Subvirt [7, 8] – have already been discussed in public. These attempts try to clone the system they infect into a virtual machine and install themselves as hypervisor above the virtualised system. A system that is virtualised with such malware will be completely ignorant of the fact that it has been captured. Since all installed scanning software will run in the virtualised environment, the Trojan horse is able to eradicate all references to it and escape detection. Although there may exist possibilities to detect the fact that the system has been virtualised from within the virtual environment, this attack still remains a serious threat to be considered.
Moreover a virtual machine is easier to steal on portable media. DVDs or USB sticks are easily hidden compared to a physical server and if the stolen copy is a clone of the original, while the original is still running and providing its services, not even its absence will be noticed.
Even if a trusted user takes a copy of a virtual machine with him to continue work at home, that may open up security problems, as the machine leaves the local administrative vicinity and enters a new environment with an unknown security context. While security precautions – like firewalls or virus scanners – may have protected the VM in its original environment, this may no longer be guaranteed in the new environment.
Although one of the envisioned advantages of virtualisation was easier administration, that goal has only partly been achieved. In some scenarios, virtualisation can help to make

administrative issues easier – like when cloning a VM in order to produce a backup copy. Unfortunately however, there still remains some amount of necessary administration. VMs still have to be created, deleted, copied or moved by hand. Some management solutions for virtual machines have been created – VMWare Infrastructure 3 and Citrix XenServer are examples of this [9, 10].

2.7.3 Summary

There are quite a lot of different approaches to virtualisation available today. Within the AutoI project, specific needs have to be catered to. Virtual Routers, as envisaged by the project, will have to operate under high network and CPU load, so a virtualisation solution with low overhead should be chosen. Most virtualisation solutions have basic management primitives in place. The Virtualisation Plane will build upon these management primitives and offer more comprehensive functions called via the vCPI (see Section 6).

3 Virtualisation in AutoI

3.1 Physical and virtual routers

Virtualisation techniques, as described before, are used as basic enabler for the AutoI project. As one of the main component of AutoI the physical routers and servers of a network are virtualised, in order to get the flexible concept of virtual routers and virtual topologies. In figure 3.2.3:1 the concept of these virtual resources is illustrated. It can be seen that on a single hardware, several different virtual machines are hosted, providing virtual routers. Virtual routers are able to provide different protocol stacks, as indicated in Figure 3.2.3:1. The virtual machines access the physical resources through a new layer, called the Future Internet Virtualisation Layer and are managed by other planes through the virtual Component Programming Interface (vCPI). The whole structure is then called a physical component.

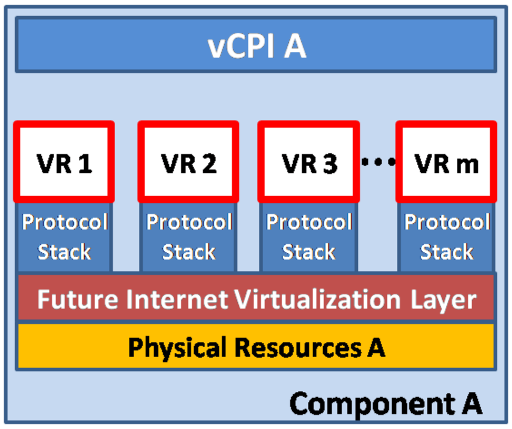


Figure 3.2.3:1 - Physical component with Virtual Routers and Virtual Services

When virtual routers are encapsulated in virtual machines, some advantages arise in their handling. Virtual machines are usually stored in one or more files (or within a file structure) and can easily be moved or copied. In general, virtual machines can be

- started and stopped at any time,
- created to get an additional virtual machine,
- copied to clone a virtual machine
- and be moved to another physical component, e.g. if the hardware fails.

With these basic functions of virtual machines, more complex scenarios can be realised. A backup of a virtual router can be made for example and be restored in short time, when necessary. This enables a “rollback in time”, because it is possible to reactivate a router in a defined state, which has previously been backed up.

3.2 Virtual routers and virtual networks

If virtual routers are interconnected to create networks of virtual routers, it is possible to define virtual topologies, which are different to the real physical network topology. This scenario is illustrated in Figure 3.2.3:2. It can be seen that the virtual network, which consists of four VR2 routers, has only three links, whereas the real topology consist of five links. Moreover, not only can the virtual topology differ from the real topology, but also the bandwidth and other properties of the links can be different in the virtual networks, formed by virtual routers.

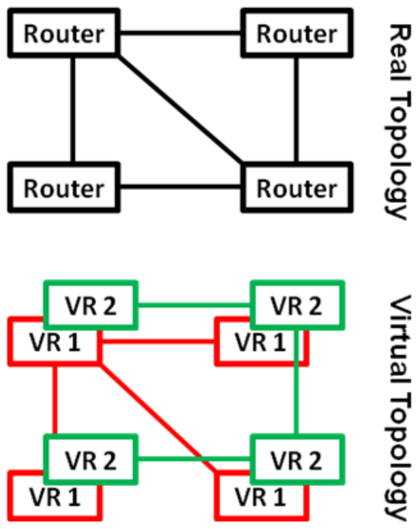


Figure 3.2.3:2 - Physical topology and virtual topologies, formed by virtual routers

Because the virtual routers are manageable, it is possible to change properties (e.g. topology or bandwidth) of the virtual networks dynamically, based on the current demand of services. It is also possible to host “paused” virtual networks, which are preconfigured for a certain scenario (e.g. as an emergency procedure) and can be launched immediately in a predefined state if necessary.

3.3 Selection of virtualisation methods and tools

To realise virtual routers it is necessary to split resources (zone), as it is done in the field of server virtualisation. The virtualisation technology has to provide the feature of splitting the physical resources of a component (CPU, memory, network, disc, etc.) in virtual resources, which are provided for the virtual routers and services.

To achieve the envisaged easy handling of virtual routers, services, and topologies, the compact design of system virtualisation in virtual machines is appropriate. The operating systems within the virtual machines can be adjusted to achieve the required management of virtual routers and topologies and the virtual machines themselves can be created, paused, copied, moved, started and destroyed.

Among the virtual machine monitors, the classical hypervisors, which replicate the instruction set architecture of the physical hardware, seem to be the appropriate choice because of the expected higher performance in comparison to emulators. For the same reason full virtualisation is preferred over hosted virtualisation because the additional layers cause additional overhead, leading to a degradation of service and performance.

A number of virtual machine monitors are available to realize the virtualisation in AutoI. XEN has been chosen as a starting point in network virtualisation. XEN is an open source virtual machine monitor, which is implemented in a modular and well efficient way. Various (x86) operating systems, e.g. different LINUX-based systems can be used as host and as guest with XEN. This makes it a preferable choice for the research and the evaluation of virtualisation features and management features in AutoI.

This kind of virtual networks is described in several publications already [15, 16, 17, 18]. It has been found that the application of system virtualisation in this context necessarily leads to the kind of virtual network presented here.

4 Interfaces to Virtual Networks - vCPI and vSPI

4.1 Interface Architecture

To hide the implementation details of network virtualisation from other planes in the AutoI project, an interface has to be designed which provides access to virtual routers and virtual topologies. In AutoI the separation between the virtualisation plane and other planes relieves the other planes from dealing directly with physical resources. Only virtualised resources are manageable via the virtualisation interface and also monitoring information about the physical and virtual resources can be requested from the virtualisation interface. This separation enables a system-wide management of virtual routers and topologies by the management plane, while the local management (managing the virtual hardware provided for the virtual routers on a single physical component) is done by the virtualisation plane.

The interface will provide functions for managing *available virtual resources* - i.e. resources that are available for future virtual services. In order to provide an appropriate layer of abstraction from the physical resources, the Virtualisation Plane inhibits direct access to physical resources and instead provides suitable monitoring values that abstract from the inherent virtualisation overhead. From that point of view the advertised available virtual resources can be considered as a service guarantee (i.e. a component reporting to be able to provide 256 MB RAM for a virtual service will actually be able to spare 256 MB less a certain amount of virtualisation overhead that has to be taken into account).

Having information about the availability of virtual resources on a certain component is important for other planes to come to informed decisions (e.g. decide whether a component should host another virtual service or not). Without information about the physical capabilities of a component, actions of other planes involving the Virtualisation Plane are prone to become an ineffective trial-and-error process.

Furthermore, the interface will provide functions to manage virtual routers (start, stop, move, copy, create and destroy) and virtual topologies (allocation of available virtual resources like bandwidth or links to virtual routers and virtual services). The interface will be based on the documents [11, 12, 13], defined by the DMTF [14] and the AutoI information model [22].

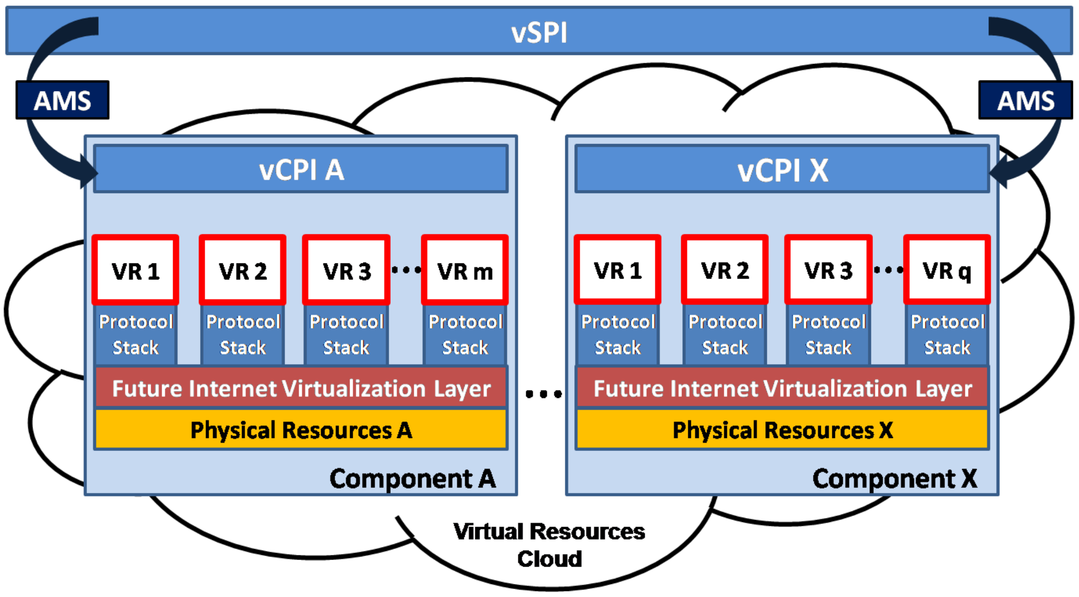


Figure 3.2.3:3 - Virtual System Programming Interface and Virtual Component Programming Interface

As can be seen in Figure 3.2.3:3, the interface is split into two parts: The virtual System Programming Interface (vSPI), which provides abstract methods for global management of components, and the virtual Component Programming Interface mentioned before, which allows to manage the local available virtual resources. The vSPI controls the virtual resources available within the Virtual Resources Cloud through interaction with Autonomic Management Systems (AMSs).

Having access to both, management and monitoring functions, other planes can plan ahead and manage the available virtual resources, initiating the creation of new virtual services, the migration of existing services or the shutdown of superfluous services in a timely manner. Both, management and monitoring functions can be accessed by other planes via the vCPI depicted above.

4.2 DMTF Standards

Several DMTF Standards can be considered as a starting point for management considerations. It has to be noted however, that there are subtle incompatibilities with some DMTF documents that are detailed in D3.1. Therefore, while taking these documents as a valuable first step, the final focus will be on the information model provided by Work Package 3.

4.2.1 Virtual System Profile

The Virtual System Profile is a minimal object model designed to represent virtual systems, its components and to offer basic operations on it. The aim is to present virtual systems to a client in the same way as non-virtual systems. As a virtual system is a specialization of a non-virtual system, the Virtual System Profile divides it up into two parts:

The virtualisation-aware part of a system consists of configuration items representing basic information about the virtual system's state. This might include the system's name, a description, the system type and the current state of the virtual system. Also included is information about resources allocated from the host's resource pool to the virtual system.

The virtualisation-unaware part of a system is an instance of DMTF's Computer System Profile representing a non-virtualised system. Logical devices, which are allocated to a virtual system based on allocation requests, are modelled here as well.

4.2.1.1 Resources

Both physical and virtual resources are modelled as instances of a hardware specific DMTF profile. DMTF provides profiles for a number of resources, e.g. a CPU and System Memory Profile. All resources are unaware of their virtualisation. Therefore, virtual resources do not hold any additional information compared to their physical counterparts. In case it is required to model a resource for which no DMTF profile exists, a Generic Resource Virtualisation Profile exists which can be used for this purpose.

4.2.1.2 System states

A virtual system can be in different states:

Defined: The system has been created by the host and is not running, only persistent resources are allocated.

Active: The system is running and able to perform tasks. Resources assigned to the system are allocated and active.

Paused: The system is not able to perform any tasks. Resources assigned to the system are allocated but not able to be used.

Suspended: The system's state (including resources) is saved persistently. Allocated resources may be de-allocated as needed.

To make transactions between those states, a number of state transactions are defined. It is optional to implement any of the transactions possible from states listed above. Methods for querying the current state and requesting a state change of a virtual system are available.

4.2.2 System Virtualization Profile

The System Virtualization Profile is a minimal object model designed to represent the host of virtual systems and to offer basic operations on them. Additionally, it provides methods to allow their configuration and the control of resource allocations.

4.2.2.1 Creation of virtual systems

This profile offers methods for the virtual system's definition and destruction. If during definition of a particular virtual system no configuration settings are provided, a reference configuration with default values is used. Additionally, resources to be allocated to the virtual system during its lifetime can be provided. At the time the virtual system is not running, disk space needed to store the system's disk image is the only resource persistently being occupied. It is freed again on the system's destruction. Configuration items and resource allocations can still be altered after a virtual system's definition using provided methods.

4.2.2.2 Managing resources

Resources are divided up into persistent and transient ones:

Persistent resources are allocated on a virtual system's definition and stay allocated until its destruction, independent of the virtual system's state (e.g. disk space consumed by the disk image).

Transient resources are allocated only on or after a virtual system's activation during its lifetime. Once the virtual system is deactivated, resources are de-allocated again (e.g. CPU, memory, network bandwidth).

The host system provides a resource pool offering a subset of the host's physical resources allowed for allocation. The resource pool is defined in DMTF's Resource Allocation Profile and not further discussed in this profile. Resources are allocated to a virtual system based on allocation requests, where the desired resource is modelled from a hardware specific profile and passed to a method handling said requests. Furthermore, this profile offers methods for modifying or removing an already allocated resource during the virtual system's lifetime.

4.2.2.3 Snapshots

The implementation of snapshots to reproduce a past point in time is optional. If implemented, it is only required to capture the configuration state of a virtual system at the current time. The inclusion of resources' states, like contents of memory or the state of the disk image, is optional. Methods for creating, applying and destroying a snapshot might be implemented.

4.2.3 Open Virtualization Format

The Open Virtualization Format (OVF) was introduced as a common storage format for virtual machines. It provides a vendor and platform independent way of specifying virtual machine requirements such as the hardware that should be associated with a virtual machine, or the virtual disk that contains the necessary software. Due to its flexibility and compatibility it appears as a good foundation for the specification of Virtual Routers in the AutoI architecture.

4.3 DMTF CIM and the specific AutoI Information Model

In true modelling principles, CIM is actually a mix of an Information & Data model. Information modelling is based on ordering our domain knowledge in technology and platform neutral abstraction levels for a better understanding of the domain, it attempts to simplify the system’s complexity. The blend of conceptual entities and their instances in CIM results in a difficulty to understand the concepts independent of the instantiation details which are specific to platforms and implementations. In addition, CIM does not provide specifications for important network concepts, such as separation of physical and logical device interfaces, templates for network element classes (i.e. routers, bridges) or models for Quality of Service. It is for these reasons that DEN-ng is the basis for modelling resource virtualisation concepts in AutoI. DEN-ng is an information model which captures concepts that enable multiple coherent viewpoints of a communications network, provides for a division of physical and logical resources and allows state machines extension for the effective monitoring of entities. All these concepts are represented while it is maintaining a key relationship with the business goals and quality of service goals. DEN-ng is extended for the purposes of AutoI by building a specific information model AutoI Information Model (AIM) related to the project domain. Virtualisation concepts are designed and added to the AIM model such as: *VirtualResource, VirtualPhysicalResource, VirtualLogicalResource, VirtualResourceState, VirtualLogicalDevice* and *VirtualRouter.* Each of these concepts has appropriate relationships with their underlying physical and logical resources within the model. The specifications of these concepts are defined in D3.1.

5 Programmability of the Virtualisation Plane

5.1 The Virtual Machine Monitor (VMM)

The Virtualisation Plane contains dynamic plug-ins that may be dynamically updated or replaced. The design of this infrastructure should be flexible enough to adapt to changes that may occur. For example, XEN may not be the best hypervisor option in the near future. Other tools with better functionality may be proposed. So, the VMM component should be dynamically replaced by another one that communicates with the new hypervisor. However, this process should not trigger unexpected events. Clearly there is no need for hypervisor replacement in the duration of the AutoI project. However, our infrastructure should be as loosely coupled with XEN as possible.

The new hypervisor should support at least:

- The basic functionalities of XEN that vSPI and vCPI interfaces require.
- Conversion of the XEN image to a compatible format for the new hypervisor.

The main issue here is that this process should not interrupt the services provision. During the VMM migration phase, the two VMM versions may coexist because all the virtual machines should be moved to the new virtualisation environment. This process may take place gradually, so the old VMM version should remain active until all virtual machines are moved. During that transitional phase, the vSPI/vCPI interfaces should communicate with both VMMs with minimum or no impact on the other planes.

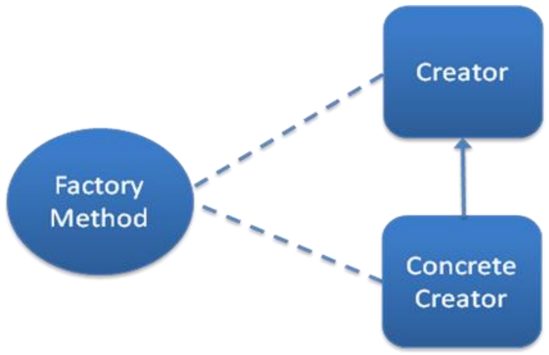
5.2 The vCPI & vSPI interfaces

The interfaces to the Virtualisation plane should have a more static nature than the Virtual Machine Monitor because they interact with a significant number of AutoI components. For example, every component that communicates with the two interfaces and the information model should be updated in order to reflect these changes. In AutoI, we propose a flexible abstract design for the two interfaces, trying to predict the future requirements. Consequently, some abstract methods may not be needed now but are proposed for this reason.

5.3 Communication with the dynamic components

The dynamic plug-ins of the Virtualisation Plane should be updated real-time without interrupting the provided services. So, the dynamic code deployment should be handled by an infrastructure that provides that functionality. We plan to use the factory method design pattern in the execution and service-lifetime functionalities of AutoI. The Factory Method is a creational pattern that models an interface for creating an object. This pattern allows its subclasses to decide which class to instantiate at creation time. This pattern is called a Factory Pattern, since it is responsible for "Manufacturing" an Object. It helps instantiate the appropriate Subclass by selecting the right Object from a group of related classes. The Factory Pattern promotes loose coupling by eliminating the need to bind application-specific classes into the code [1].

As shown in the following figure, the Factory Pattern is associated with a “Creator” and a “Concrete Creator”. This pattern is used when the “Creator class” does not know beforehand all the subclasses that it will create. For example, does not need to know which VMM component is going to be deployed. Instead, the “Concrete Creator” is left with the responsibility of creating the actual object instances.



6 vCPI (Virtualisation Component Programming Interface)

The virtual Component Programming Interface (vCPI) provides a way for the AutoI Management Plane to configure the components and start new Virtual Routers forming Virtual Networks. Each component provides its own instance of the vCPI. The vCPI exposes several different methods to its clients.

6.1 Virtual Machine lifecycle

A simple lifecycle for virtual machines is depicted in the following picture.

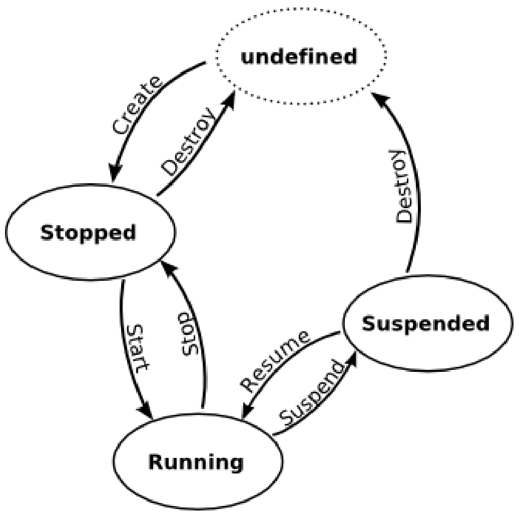


Fig. 0-1: Virtual Router Lifecycle

A Virtual Router can be in the following states:

- Undefined – this is a pseudo-state that has no bookkeeping information associated with it.
- Stopped – A VR that is defined, but not started yet, is in that state.
- Running – This corresponds to a VR that is currently executing.
- Suspended – This corresponds to a VR that was started and subsequently paused with its internal state saved to a file on disk.

6.2 Virtual Machine management methods

These methods manage the Virtual Machines (VMs) on the local Component. Each VM is considered to contain a routing Operating System, forming a Virtual Router (VR).

Method: instantiateVM
Parameters: URL to OVF file
Returns: Error Code
Description: This method creates a new instance of a Virtual Machine, based on an OVF file.

Method: changeVMState
Parameters: VM Identifier object, new VM state
Returns: Error Code
Description: This method changes the state of a Virtual Machine.

Method: initiateVMMigration
Parameters: VM Identifier object, Component Locator object
Returns: Error Code
Description: This method initiates a migration of a Virtual Machine from this Component to another Component.

Method: completeVMMigration
Parameters: Component Locator object
Returns: Error Code
Description: This method completes a migration of a Virtual Machine from another Component to this component.

6.3 Virtual Link management methods

These methods manage a Virtual Link connecting two VRs. The methods have to be called at both end points (at both Components).

Method: instantiateLink
Parameters: VM Identifier object for first VM, VM Identifier Object for second VM, Link Parameters
Returns: VL Identifier object
Description: This method sets up a new Virtual Link between two Virtual Routers.

Method: removeLink
Parameters: VL Identifier object
Returns: Nothing
Description: This method deletes a Virtual Link between two Virtual Routers.

Method: modifyLink
Parameters: VL Identifier, Link Parameters
Returns: Nothing
Description: This method modifies the Link Parameters of a Virtual Link.

6.4 Component Monitoring methods

These methods allow other planes to obtain monitoring values from the Component.

Method: getMonitoringValues
Parameters: None
Returns: Available clock cycles, available RAM, available HD, number of NICs + available bandwidth per NIC
Description: This method returns the monitoring values available at this Component.

Method: getVMList
Parameters: None

Returns: A list of VM Identifier objects
Description: This method returns a list of VM Identifiers corresponding to the Virtual Routers on this Component.

Method: getVLList
Parameters: None
Returns: A list of VL Identifier objects
Description: This method returns a list of VL Identifiers corresponding to the Virtual Links on this Component.

7 vSPI (Virtualisation System Programmability Interface)

The vSPI interface performs a higher-level management of virtual resources than vCPI. This application interface allows Management and Orchestration Planes to control groups of virtual resources, including components and whole topologies. We note that a component is a grouping of resources within a physical host (see Figure X).

7.1 Component Manipulation

The following methods allow Management and Orchestration Planes to manipulate components.

Method: createComponent
Parameters: PhysicalHost, NumberofVMs, VMsConfigurations
Returns: The ComponentID of the new Component.
Description: Creates a new component that includes the physical resources of the specified host. The new component consists of the specified number of virtual machines. The new virtual machines have automatically the specified configuration.

Method: updateComponent
Parameters: ComponentID, ComponentConfiguration / VMsConfigurations
Returns: 0 if it is successful or the error number.
Description: Updates the configuration of a Component or a set of VMs deployed to a Component.

Method: migrateComponentSoftware
Parameters: ComponentID, DestinationPhysicalHost
Returns: 0 if it is successful or the error number.
Description: Migrates the software of a component (i.e., the one that associates with the provided ComponentID) to a different physical host. This includes the virtual resources deployed to this component. The configuration sets of both components and virtual machines remain the same.

Method: getComponentList
Parameters: None
Returns: A list of components.
Description: This method requests the list of active components.

Method: checkComponent
Parameters: ComponentID
Returns: Returns the configuration set of the specific component.
Description: This method requests the configuration set of the specified component.

7.2 **Component Monitoring**

The components can be monitored through the vSPI interface. The main difference between getMonitoringValues and checkComponent is that the former requests a specific parameter while the latter the full configuration set.

Method: getMonitoringValues
Parameters: ComponentID, Parameter
Returns: Parameter value.
Description: Returns the value of the specified parameter that associates with the specified component.

7.3 **Manipulation of Resource Groupings**

Virtual resources from different components can be grouped in order to support a specific service. We note that these methods associate only with the virtual resources of a given service and not the service itself.

Method: establishResourceGroup
Parameters: ServiceID, SetofVMservers / TopologyID
Returns: 0 if it is successful or the error number.
Description: Establishes a new group of resources that associates with a specific service. A service can be associated either with a set of virtual resources or a virtual topology.

Method: removeResourceGroup
Parameters: ServiceID
Returns: 0 if it is successful or the error number.
Description: Removes the resources associated to a specific service.

Method: configureResourceGroup
Parameters: ServiceID, Configuration
Returns: 0 if it is successful or the error number.
Description: Configures / updates the resources of a specific service.

Method: checkServiceResources
Parameters: ServiceID, Parameter
Returns: Parameter value.
Description: Returns the value of the specified parameter that associates with the specified service.

7.4 **Manipulation of Virtual Networks**

The vSPI supports methods that manipulate virtual networks. For example, a new virtual network that follows a specific topology pattern may be created through the vSPI.

Method: createNetwork
Parameters: TopologyPattern, NumberofNodes/setofNodes
Returns: The networkID of the new topology.

Description: Creates a new network that follows the topology pattern, either with the specified number of nodes or from the specified set of nodes.

Method: updateNetwork
Parameters: NetworkID, TopologyPattern, NumberofNodes / setOfNodes
Returns: 0 if it is successful or the error number.
Description: Updates an existing virtual network to: (i) follow the specified topology pattern or (ii) to update the number of nodes, or (ii) to update the participating set of nodes.

Method: removeNetwork
Parameters: NetworkID
Returns: 0 if it is successful or the error number.
Description: Removes the specified network. We note that this network should not be used.

Method: checkNetwork
Parameters: NetworkID
Returns: Configuration set.
Description: Returns the configuration set of the specified topology (e.g., number of nodes, topology pattern etc).

Method: monitorNetwork (Parameter)
Parameters: NetworkID
Returns: A notification with the new parameter value.
Description: Monitors a specific parameter of the configuration set.

Method: getTopologyPatterns
Parameters: Node
Returns: Topology patterns.
Description: Returns the available topology patterns.

Method: migrateVirtualRouter
Parameters: VirtualRouterID, locationA, locationB
Returns: 0 if it is successful or the error number.
Description: Migrates a virtual router from locationA to locationB. This method triggers changes to the two topologies.

8 Conclusion

In this document the state of the art of virtualisation methods has been described. Based upon a comparison of different virtualisation solutions, system virtualisation has been identified as an appropriate method of virtualisation to virtualise the resources of a network. Routers were embedded in virtual machines and were linked via virtual links on top of the physical network. Using this virtualisation method, several independent virtual networks can be created upon a single physical network. Furthermore, this document has described and defined two interfaces that enable the configuration of such virtual networks, the vCPI and the vSPI. The vCPI is the interface of a single component, consisting of hardware, virtual machine monitor and several virtual routers. It enables creation, modification, and destruction of virtual networks by applying basic primitives to the component (e.g. creates, destroys, starts, stops, moves, or copies a virtual router). In addition to this “localised” management of virtual resources, the vSPI is an interface that allows a configuration of virtual resources that is based on a more global view of the system. Groups of virtual resources can be controlled via the vSPI, including components and whole topologies. Both interfaces do also consider the monitoring of virtual resources to enable their reasonable configuration. Also the programmability of virtual resources has been described in this document, considering a dynamic update or replacement of code within the local and the global management functions. Programmability is necessary to achieve an infrastructure design that is flexible enough to adapt to unforeseen changes that may occur in the future.

8.1 Future Work

In the future work, the interface definitions (vCPI and vSPI) have to be further refined and prototyped using JAVA language and (initially) XEN as virtual machine monitor. The programmability of virtual resources has to be further developed and to be prototyped, providing further integration with Management Plane concepts.

8.2 Security in Virtual Networks

In the context of security, virtualisation is a two-edged sword. On the one hand it provides an encapsulation of different services, ensuring that the failure of one service will not hamper other services. With that functionality, even malicious services can be tolerated to a certain extent.
On the other hand, virtualisation provides another layer of abstraction, creating both, more overall complexity as well as additional interfaces that can be targeted by attackers. Any architecture employing virtualisation techniques will therefore have to consider security issues and ensure that no additional threats are introduced.
Within this project it has to be noted that security issues arise at interfaces, requiring access to management functions to be restricted via access control mechanisms. The exact details of how this should be done are outside the scope of this project, however.

References

1. Gopalan Suresh Raj. ***The Factory Method (Creational) Design Pattern***. http://gsraj.tripod.com/design/creational/factory/factory.html
2. P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield. ***Xen and the art of virtualization***. Proceedings of the nineteenth ACM symposium on Operating systems principles. ACM, New York, NY, USA, 2003.
3. ***VMWare ESX – Bare-Metal Hypervisor for Virtual Machines***. http://www.vmware.com/products/vi/esx/.
4. ***VMWare Workstation***. http://www.vmware.com/products/ws/
5. G. J. Popek, R. P. Goldberg. ***Formal Requirements for Virtualizable third Generation Architectures***. Communications of the ACM 17(7): 412 – 421. ACM, New York, NY, USA, 1974.
6. ***Home of the Xen hypervisor***. http://xen.org/.
7. J. Rutkowska. ***Subverting Vista kernel for fun and profit***. Black Hat 2006, Las Vegas, Nevada, USA, 2006.
8. S. T. King, P. M. Chen, Y. min Wang, C. Verbowski, H. J. Wang, J. R. Lorch. ***Subvirt: Implementing malware with virtual machines***. IEEE Symposium on Security and Privacy, 2006.
9. VMWare Infrastructure 3. http://www.vmware.com/products/vi/.
10. ***Citrix XenServer: Efficient Virtual Server Software.*** http://citrix.com/English/ps2/products/product.asp?contentID=683148.
11. ***DMTF Virtual System Profile***. DSP1057, Version 1.0.0a, 2007. http://www.dmtf.org/standards/published_documents/DSP1057.pdf.
12. ***DMTF System Virtualization Profile***. DSP1042, Version 1.0.0a, 2007. http://www.dmtf.org/standards/published_documents/DSP1042.pdf.
13. ***DMTF Open Virtualization Format Specification***. DSP0243, Version 1.0.0d, 2008. http://www.dmtf.org/standards/published_documents/DSP0243_1.0.0.pdf.
14. ***Distributed Management Task Force – DMTF Home***. http://www.dmtf.org/.
15. A. Berl, A. Fischer, H. de Meer, A. Galis, and J. Rubio-Loyola. ***Management of Virtual Networks***. Proceedings of 4th IEEE/IFIP International Workshop on End-to-End Virtualization and Grid Management (EVGM2008), Samos Island, Greece, September 22-26, 2008. Multicon, Schöneiche (Germany), September 2008.
16. C. Fahy, S. Davy, Z. Boudjemil, S. van der Meer, J. Rubio-Loyola, J. Serrat, J. Strassner, A. Berl, H. de Meer, and D. Macedo. ***An Information Model That Supports Service-Aware, Self-Managing Virtual Resources***. LNCS: Modelling Autonomic Communications Environment, Proceedings of 3rd IEEE International Workshop on Modelling Autonomic Communications Environments (MACE2008), Samos Island, Greece, September 22-26, 2008. Springer, Berlin (Germany), September 2008.
17. A. Fischer, A. Berl, and H. de Meer. ***Virtualized Networks based on System Virtualization***. 2nd GI/ITG KuVS Workshop on the Future Internet, Karlsruhe, Germany, November 11th 2008.
18. A. Fischer, A. Berl, and H. de Meer. ***Virtual Network Management with XEN***. GI/VTG Fachgruppe Betriebssysteme und KuVS, Garching, Germany, 23-24 October 2008.
19. M. Abid, A. Berl, Z. Boudjemil, A. Cheniour, S. Davy, S. Denazis, A. Galis, J.-P. Gelas, C. Fahy, A. Fischer, J. Rubio-Loyola, L. Lefèvre, D. F. Macedo, L. Mamatas,

H. De Meer, Z. Movahedi, J. Strassner, H. Zimmermann. ***Autonomic Internet Initial Framework***. AutoI Project Deliverable D6.1, August 2008.

20. A. Berl, A. Fischer, L. Mamatas, A. Galis, Z. Boudjemil, S. Davy. ***Initial Monitoring of Virtual Networks***. AutoI Project Deliverable D1.1, January 2009.
21. G. Pujolle, Z. Mohavedi, M. Abid, D. F. Macedo, S. Davy, A. Cheniour, J. Rubio-Loyola, G. Koumoutsos, A. Galis. ***Initial Orchestration Plane Design***. AutoI Project Deliverable D2.1, January 2009.
22. Z. Boudjemil, S. Davy, D. Muldowney, C. Fahy. ***Information Model***. AutoI Project Deliverable D3.1, January 2009.
23. L. Mamatas, A. Galis, D. F. Macedo, J. Rubio-Loyola, S. Davy, Z. Boudjemil, A. Cheniour, A. Berl, A. Fischer. ***Initial Management and Knowledge Planes Functions and Initial Design***. AutoI Project Deliverable D4.1, January 2009.
24. L. Lefèvre, A. Cheniour, J.-P. Gelas, C. Fahy, D. F. Macedo, A. Fischer, L. Mamatas. ***Initial Service Infrastructure***. AutoI Project Deliverable D5.1, January 2009.
25. J. Rubio-Loyola, J. Serrat, A. Astorga, S. Davy, G. Koumoutsos, S. Denazis, L. Mamatas, Z. Boudjemil, A. Fischer, A. Galis, G. Pujolle, M. Abid, A. Cheniour, L. Lefèvre. ***Initial Test-bed and Use Cases***. AutoI Project Deliverable D6.2, January 2009.