

---

## 2 ANALYSIS

---

### 2.1 VIRTUALIZATION<sup>5</sup>

Virtualization is the technique to create an abstraction level between the hardware resources and the software, such as an operating system (OS) and applications. Software is installed and runs in a virtual machine (VM), which replaces – from the software point of view – the classical machine composed by physical hardware. Obviously, the software which runs on a virtual machine uses the physical resources. The link between the virtual machine and the real hardware is made by a virtualization platform. For example, when the operating system want to write to the hard disk, it actually access to a virtualized disk. The virtualization platform takes the write command and passes it to the physical hardware.

A virtualization platform is software composed of two main parts: the hypervisor and a management interface. The hypervisor is the layer between the physical hardware and the software installed on the virtual machine. It provides four main functions (1):

- **Hardware emulation.** This means the hypervisor presents a virtual hardware environment to the software which runs on the VM. It manages and shares the hardware resources with the other applications which run on the physical machine.
- **Isolation between multiple VMs.** If a hypervisor can provide multiple virtual machines running concurrently, it must isolate the operations of the different VMs, i.e. an error that occurs in a VM should not affect the others.
- **Physical resources allocation.** When an application wants to allocate resources, the hypervisor must do it in the physical hardware. It must manage the allocation between the different VMs, trying to optimize the performance.
- **Encapsulation of the VM.** One goal of the virtualization is the portability between different systems, i.e. to migrate a virtual machine from a physical machine to another one where the hypervisor is installed. The encapsulation allows transferring the entire VM to the other machine, preserving its integrity.

The management interface provides the control of the virtual machine to the user. The most important controls are: to launch and shut down the virtual machine, but also to freeze and resume it, to copy and destroy it. The management interface allows also choosing the allocation of the physical resources, like the amount of RAM, the storage size, the network interfaces and the other I/O peripherals which the virtual machine is allowed to use.

Figure 1 illustrates the architecture of a system composed by a hypervisor which is installed on a host OS and which manages two different virtual machines. In this case, the hypervisor uses the host operating system to communicate with the hardware, because it is treated as the others applications which run on the host OS.

---

<sup>5</sup> Some subsections are taken or inspired from (24 pp. 10-18).

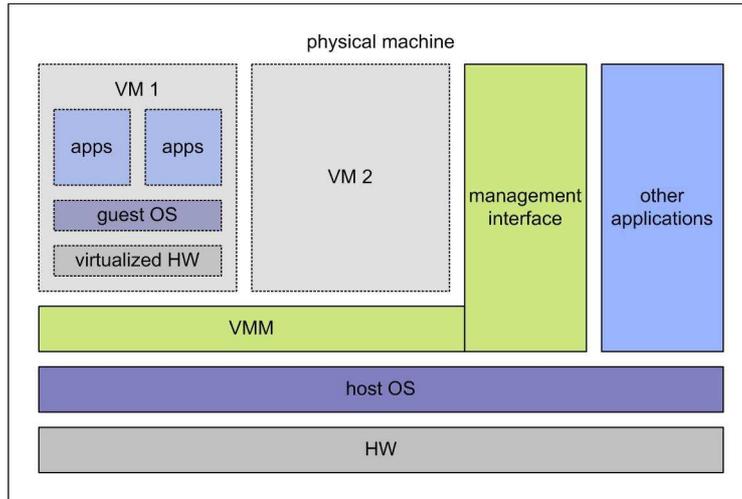


Figure 1 – Virtualization Over OS

### 2.1.1 Virtual Machines

The original goal of a virtual machine is to offer multiple working environments within the same physical computer. A virtual machine is a software implementation of a hardware machine that acts as a container which can run its own operating system and applications.

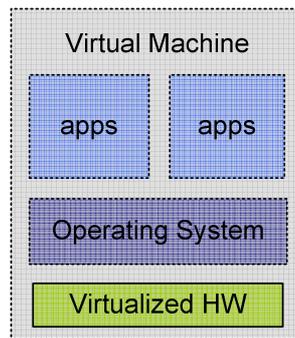


Figure 2 – Virtual Machine Structure

A virtual machine doesn't refer to a specific kind of operating system or hardware. An empty virtual machine comes just with a virtual BIOS boot as would be on a physical machine. The first step before deploying specific application is to install an operating system inside the virtual machine. The virtual machine offers a set of generic virtual hardware. Usually all the virtualization software allow the user to set some properties of the virtual machine like the amount of virtual memory that you want to dedicate to it or the number of processor available.

A physical computer can be powered on, powered off. These actions are also available with a virtual machine. A running virtual machine is often called a live VM. A live VM is a virtual machines which has been powered on. In addition, a virtual machine can be suspended. Suspending a VM means that all the content of the virtual machine's memory is dumped into a file on the host system. By doing this you keep the state of the virtual machine exactly as it was when you suspended it. Then a suspended machine can be resumed. When a VM is resumed, the inverse process is done: read the memory dump file and load the content into the virtual machine memory.

From computers in a network, a virtual machine is seen as real physical computer. Even the virtual machine itself thinks that it is a real physical computer. This means that a VM has its own IP address and its own MAC address.

Basically a virtual machine is materialized as a group of files. Each file has a specific purpose. For instance, most of the virtualization software use one or more files to hold the virtual machine file system, another to describe the VM characteristic like the amount of memory or the number of processor.

Finally here is a list of most of the benefits that are indeed when you use virtual machines:

- A virtual machine can run independently of the underlying physical hardware.
- A virtual machine is isolated from another VM running in the same host machine, as they were two separated physical computers.
- A virtual machine contains a full computational environment. This mean that a virtual machine holds a complete operating system with all the environment variables, the applications with their variables and data.
- A virtual machine is materialized as a bunch of files on the host file system. This indeed that transferring or moving a virtual machine consist on a simple file transfer or copy.

## 2.1.2 Virtualization Techniques

### 2.1.2.1 Full Virtualization

Full virtualization is a technique that uses a hypervisor to provide a fully emulated machine. Then an operating system can run into this virtual machine. The advantage of this technique is the flexibility, because the operating system and the physical hardware are not coupled. The hypervisor mediates between the guest operating systems and the physical hardware. For this reason, the cost of the flexibility is the performance. In fact when a guest operating system performs a request to access the hardware, the hypervisor has to trap his request and handle it because the physical hardware is not owned by an operating system but it is shared with all other operating system. Unfortunately this process is time consuming. Figure 3 shows the position of the hypervisor between the physical hardware and the guest operating systems.

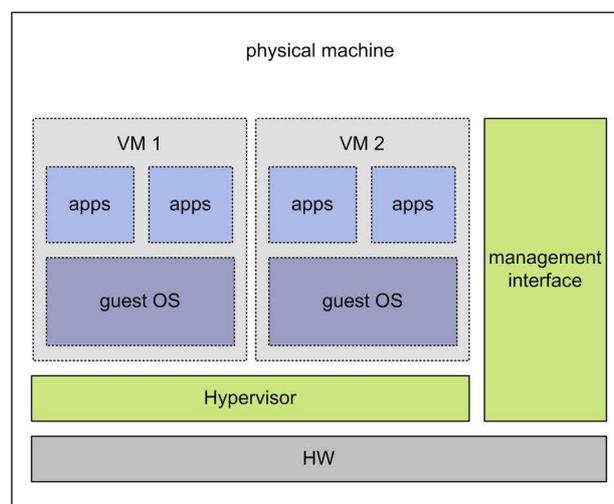


Figure 3 – Full Virtualization

### 2.1.2.2 Paravirtualization

The main difference between the full virtualization and the paravirtualization is that in the second one, the guest OS knows that it is virtualized. Thus paravirtualization involves modifying the OS kernel. With this technique, the guest OS cooperate in the virtualization process. The paravirtualization technique also uses a hypervisor. As we have said before a hypervisor is the software responsible for hosting and managing virtual machines. It runs directly on the hardware. Since the guest OSes are modified, the hypervisor doesn't trap all the guest OS instructions and this clearly involves better performance. However because of the need to modify the OS kernel, the paravirtualization doesn't support non-open source operating system like Windows.

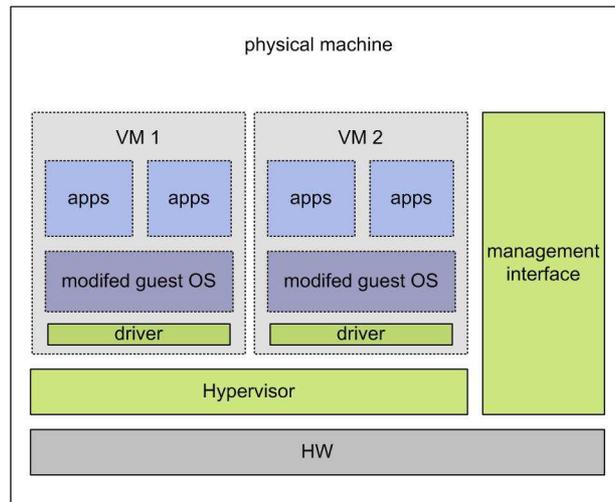


Figure 4 - Paravirtualization

### 2.1.3 Software Appliance

A software appliance is the combination of a software application and a just enough operating system (JeOS) in order to make it able to run on standard hardware or in a virtual machine. A JeOS is an operating system (typically Unix-based) which contains only the set of packages needed by the application. For instance, if we make a software appliance with an FTP server, the operating system doesn't need a graphical interface.



Figure 5 – Software Appliance

The software appliance is particularly suitable to be deployed on generic hardware as well as in a virtual machine. This concept simplifies the installation, configuration and maintenance cycle.

### 2.1.4 Virtual Appliance

A virtual appliance is a software appliance packaged into a virtual machine. A virtual appliance can run within a virtual container offered by the virtualizations tools like VMware or VirtualBox.

Also deploying a virtual appliance is even more simple than deploying a software appliance. The only requirement is to have the suitable virtualizations tools installed.

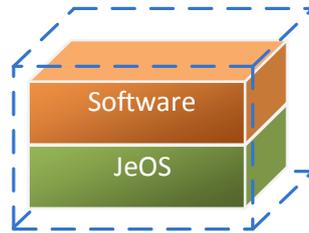


Figure 6 – Virtual Appliance

The difference between a virtual appliance and a virtual machine is that a virtual appliance is a fully pre-installed pre-configured software application and operating system. In the other hand, a virtual machine is just a container to host an operating system and software.

## 2.1.5 VMware

### 2.1.5.1 VMware Server

VMware is a software company which develops a family of virtualization products. For this project, we used versions 1 and 2 of VMware Server. VMware mostly uses the full-virtualization technology for its products (see Section 2.1.2.1). By using the full-virtualization technique, VMware's virtual machines are able to host both Unix-like operating systems and Windows operating systems. The VMware Server main characteristics are:

- available for Windows and Linux;
- Linux and Windows operating systems can be installed in the VMs;
- uses a full virtualization hypervisor;
- is free.

The main user interface of the application is the VMware Server Console. It allows controlling and running the virtual machines which are installed on the local computer or on a remote host. When a virtual machine is started, we can use the guest OS directly from the VMware Server Console window. Figure 7 shows the program running a Unix-like operating system in a virtual machine.

VMware Server Console allows creating virtual machines and set its hardware resources, like the amount of RAM and of disk space, virtual CD-ROM devices and virtual network cards. Virtual machines can be run, suspended, resumed and shut down. When a virtual machine is running, we can use it like a normal machine, just clicking on the console of the VM.

A virtual machine is fully contained into a set of files. These files store the machine environment, like the contents of the virtual hard disks, the configuration of the VM's virtual resources and, if the machine is not shut down, the full content of the virtual RAM. This means we can copy these files to store a back-up of the machine, or we can move them on another computer which runs VMware Server. Then, we can execute the virtual machine in this other PC. VMware offers some management products (other than VMware Server) which facilitate the deployment of virtual machines over a grid of computers.



Figure 7 – VMware Server Console Running a Unix-like Operating System

#### 2.1.5.2 VMware Server General Architecture

VMware Server can be used on a single computer and over a network. VMware Server is able to run a virtual machine on a remote computer. In this case, the PC which stores the virtual machine files and run it is called the host. The computers which access to the host to run a virtual machine are the clients.

The client communicates with the virtual machine of the host from an application. This application can be VMware Server or a program which uses the VMware API. The communication between the client and the host is secured using the SSL protocol over TCP/IP. Figure 8 shows the VMware general architecture over a network. However, during this project we used VMware server only locally. This means that the host and the client are on the same computer.

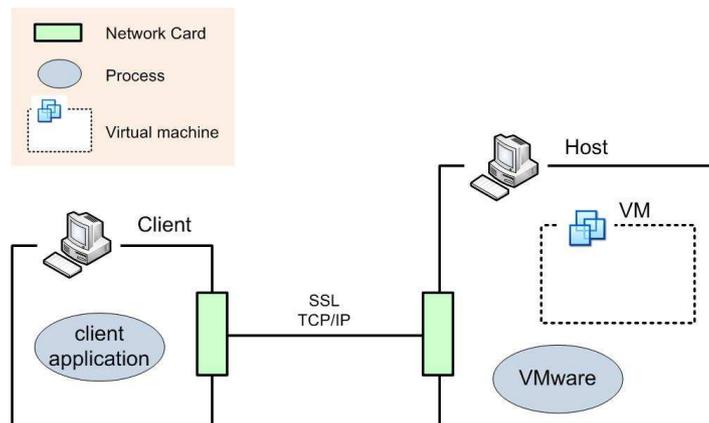


Figure 8 – General VMware Architecture

A virtual machine is composed by some files. They contain all the information about the VM. The most important two are:

- one or more VMDK files, which contain the hard disk drives of the virtual machine and

- a VMX file, which contains the configuration of the virtual machine, like the amount of RAM to be used and the number of processors.

### 2.1.5.3 *Virtual Machine Files*<sup>6</sup>

On the host file system, a virtual machine is composed by some files. VMware uses several file types for its virtual machines. The most important two are (by file extension):

- VMDK: this is a binary file which stores the content of the virtual machine's hard disk;
- VMX: this text file contains the configuration of the virtual machine.

A VMware virtual machine is in fact identified by its VMX file; each VM has one (and no more). A virtual machine can have one or more VMDK files. Other files used by VMware are:

- VMEM: this binary file is a full dump of the virtual machine's memory. When the virtual machine is suspended, one VMEM file is created to store the data in the memory. If the VM has a snapshot taken when the VM was running, a VMEM file contains the data in the memory when the snapshot was taken. (For more details about VMware snapshots, see Section 2.1.5.6)
- VMSD: this text file contains some metadata about the snapshots of the virtual machine;
- VMSN: this is a binary file containing the running state of the virtual machine's snapshot;
- VMSS: this is also a binary file which contains the suspended state of the virtual machine (this file exists only when the VM is suspended).

### 2.1.5.4 *VMware Tools*

VMware Tools is a utility which can be installed on guest operating systems running on virtual machines. It enhances the performance of the VM and adds some management control utilities. In particular, the tools contain the VMware Tools Service, which allows transferring information between the host and the guest operating system, like text (using copy-paste), command invocation and files. This service is useful when we want to automate program execution on the guest OS.

### 2.1.5.5 *VMware API*

VMware offers an Application Programming Interface (API) which is able to manage VMware Server 1.0 from a C program. It is called "VMware VIX"<sup>(2)</sup> and allows to:

- start, suspend, resume, stop virtual machines,
- run programs in the VM and
- copy files from the VM to the computer which runs the C program, and the opposite.

---

<sup>6</sup> For more information, see (25).

The API contains a set of functions which allow controlling a VM on local and remote hosts. This means the C program can be placed on a different computer than that which has VMware Server installed. To access to the remote host, we have to log in using a valid username and password of the host OS.

#### 2.1.5.6 VMware Snapshots

A snapshot is used to save a specific state of the virtual machine in order to come back to this state later in the time. To allow that VMware has to copy and modify some virtual machine files. This section explains what VMware really does with the virtual machine when it takes a snapshot.

Let's take the case of a virtual machine which is composed by two files:

- VMDK: the virtual machine's hard disk;
- VMX: the VM configuration file.

You can take a snapshot when the virtual machine is running or powered off. Let's first talk about what happens when the virtual machine is powered off. The only thing that you need for coming back to the snapshot is to know all the changes that occurred on the virtual machine's disk after that snapshot. VMware realizes this by creating a new VMDK file, called *delta* or *difference* file. The delta file will contain all the modifications done by the user after the snapshot was taken. The old VMDK file, called *base* file, is not affected by the changes done on the VM's disk. It is quite simple to restore a snapshot: VMware has just to delete the delta VMDK and modify the configuration file to use the old VMDK.

When VMware takes a snapshot of a running virtual machine, the main mechanism is the same except that VMware has also to save the state of the RAM memory.

One important note about the VMware Server snapshot mechanism is that it allows only one snapshot per virtual machine. This means that if you already have a snapshot of your virtual machine, the next snapshot that you will take will override the previous one.

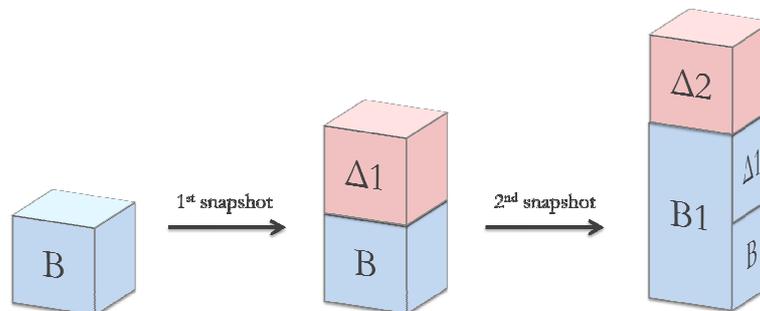


Figure 9 – VMware Server Snapshot Mechanism

Figure 9 shows the snapshot mechanism of VMware Server. The first blue cube represents the base VMDK file (B) and the red cubes represent the difference files (delta). When the user takes the first snapshot, the base VMDK file is locked in its state and a new VMDK delta file is created ( $\Delta 1$ ). Now all the changes you make to the virtual machine will be written into this delta file.

At the next step, when the user makes the second snapshot, VMware merges the previous delta file  $\Delta 1$  and the base VMDK file B to create a new base VMDK file B1. This last one contains the first base VMDK file and all the modifications done until when the user took the snapshot. A new “empty” delta file  $\Delta 2$  is created. As before, the modifications to the virtual machine's disk will be written inside this delta file, and the base VMDK is not modified.

After the second snapshot, the user can only come back to the base VMDK file B1. It is impossible to come back to the base VMDK file B, because the difference file  $\Delta 1$  created during the first snapshot is lost and the base VMDK file B has been modified.

The snapshot mechanism is totally transparent from inside the virtual machine. When a virtual machine has a snapshot, the data on the VM's disk are written only on the delta VMDK file, but are read from both base and delta files. VMware manages the access to the data on the two files transparently for the VM user. Thanks to that, from inside the VM the file system is accessed in the same way than if the VM has no snapshot.

#### 2.1.5.7 VMware Server 1 and VMware Server 2 (TO DO)

### 2.1.6 rPath

rPath(3) is company whose use the concept of virtual appliance for applications distribution and management. rPath offers some products which help the creation of software and virtual appliances. We will shortly explain the main idea of these different tools. The two main tools are rBuilder and rMake.

#### 2.1.6.1 rMake and rBuilder

As explained in Section 2.1.3, a software appliance contains the application we want to distribute and all additional components needed by it, like a configured operating system, shared libraries and other applications. rMake helps appliance developers to prepare packages which are then converted into software or virtual appliance using rBuilder. Figure 10 shows the rPath tools work flow.

With rMake we can create packages for different target users. A kind of target user is called *group* and has some specific environment needs. For example, Java developers need the Java virtual machine and maybe Eclipse in their appliance, and python developers need the python framework and maybe a text editor to develop their applications. The list of software requirements for a certain group is stored in a file called *recipe*.

rMake uses the recipe to pull the software together and build the appliance group, which is the package containing the appliance with the group-specific requirements. In Figure 10, the generic packages are the yellow and the orange cylinders, and the group-specific software is represented by the blue one.

rBuilder is a tool that automates the creation of software appliance and virtual appliance and ease their distribution. rBuilder converts appliance groups generated by rMake into appliance images. For the appliances distribution, rBuilder provides repository which can be accessed through a web-based front-end.

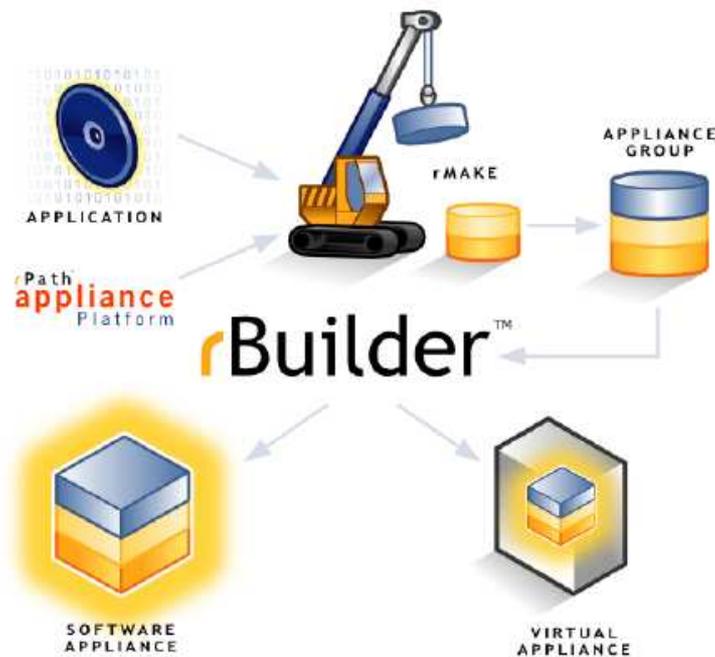


Figure 10 – Appliance Development with rMake and rBuilder<sup>7</sup>

#### 2.1.6.2 Agent

The appliance created by rPath comes with a built-in web interface. This interface allows the user to perform some basic management tasks on the appliance, for example adding a new user inside the appliance or reboot the machine.

The rPath Appliance Platform Agent (rAPA) is an extensible application framework that provides a web-based administration interface for the appliance. To manage an appliance which uses the Agent, access the web interface by using one or either of the following in a web browser:

- [https://<appliance\\_hostname>:8003](https://<appliance_hostname>:8003)
- [http://<appliance\\_hostname>:8004](http://<appliance_hostname>:8004)

The rPath Agent comes with some functionality and you can add your own functionality by adding a plugin to the agent.

A plugin in the rAPA consists of Python code and related files used to accomplish a particular administrative task on the appliance. The agent loads the plugins at startup, renders their interfaces on the web interface and executes their underlying functions.

A plugin is separated in two different components:

Web Component: The web component handles user interaction through the agent's web interface.

---

<sup>7</sup> Image source: (28 p. 2)

**Service Component:** The service component handles prolonged tasks or tasks requiring root privileges.

The web part of the plugin is used to render an HTML interface and handles the user interaction. The function of the web component can be exposed with an XML-RPC interface to allow remote plugin calls, for instance from a program. The service part of the component has to be added only if you plugin will handle prolonged task or requiring root privileges. The usual schema is to access the web part of the plugin through the web interface or by calling a function through the XML-RPC interface. Then the web part will perform some job and schedule a task which will be executed by the service part of the plugin.

The rAPA offers different ways to handle the request by your plugin: immediate execution or scheduled execution. Say that you have to choose if your task will be synchronous or asynchronous. If your task takes a long time you better use the scheduled task and then poll to display the advancement of your task in the web browser. The immediate task is conceived to be used with tasks which complete quickly.

Figure 11 shows an overview of the rPath Agent structure. At the right hand side we have the different interfaces such as HTML and XML-RPC. These interfaces give you an access to the web part of the plugin.

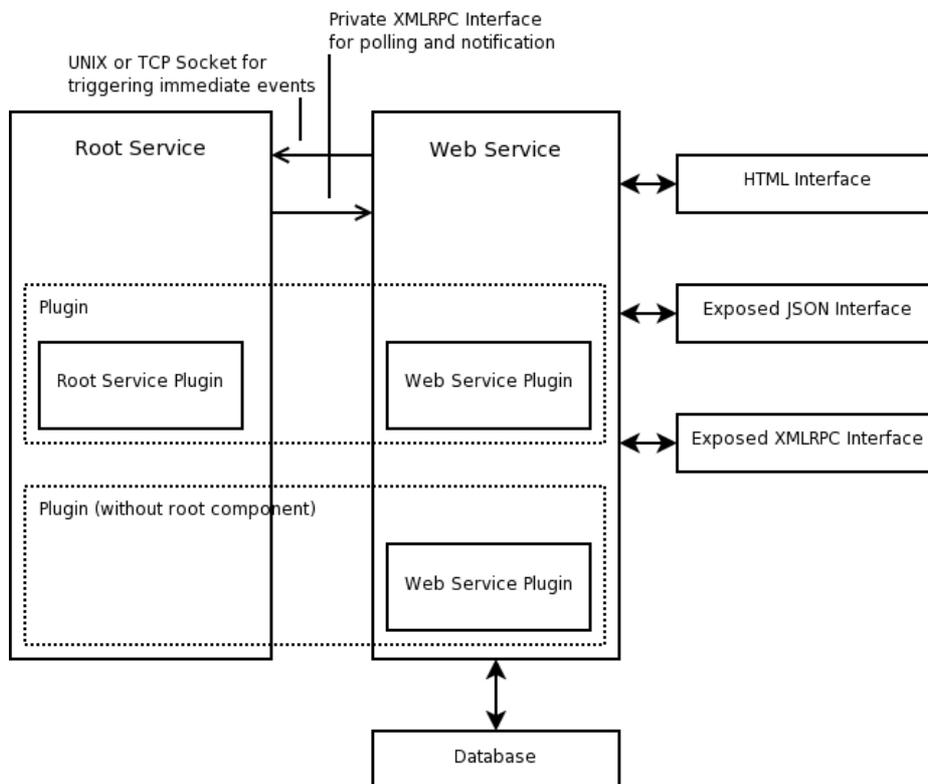


Figure 11 – rPath Agent structure

The rAPA integrates a database and you have the possibility of creating a table inside it. The plugin can use the database to save some information about the task to perform.

Basically the web component posts one or more tasks and write information about these tasks in the database. Then the service plugin fetches and execute these tasks. The service component uses a private XML-RPC interface to call back the web service in order to retrieve some information stored in the database.

### 2.1.6.3 Web Interface

Once you have an rPath virtual appliance ready to use, the first thing to do is to run it and access it by its web interface to configure your appliance. You cannot bypass this step because the appliance comes without any accounts, so you cannot SSH the appliance to configure it. One of the goals the web interface could be to setup a root account for you for instance.

The rPath web site lets you try one of their virtual machine by running it over a cloud. I did it in order to illustrate the following explications. The idea is that you launch a virtual machine on the cloud and then you add software to it. In this example the software is a media wiki. So they let you configure the wiki and allow you to access it at the end of the process.

When the virtual machine is properly started, you have to retrieve its hostname or IP address and enter the address in your browser like `https://hostname:8003`. Then you get the authentication screen. You authenticate by giving the default admin login and password. Then the appliance asks you to change the default admin password. Then comes the step specific to this example: the media wiki configuration (showed in Figure 12).

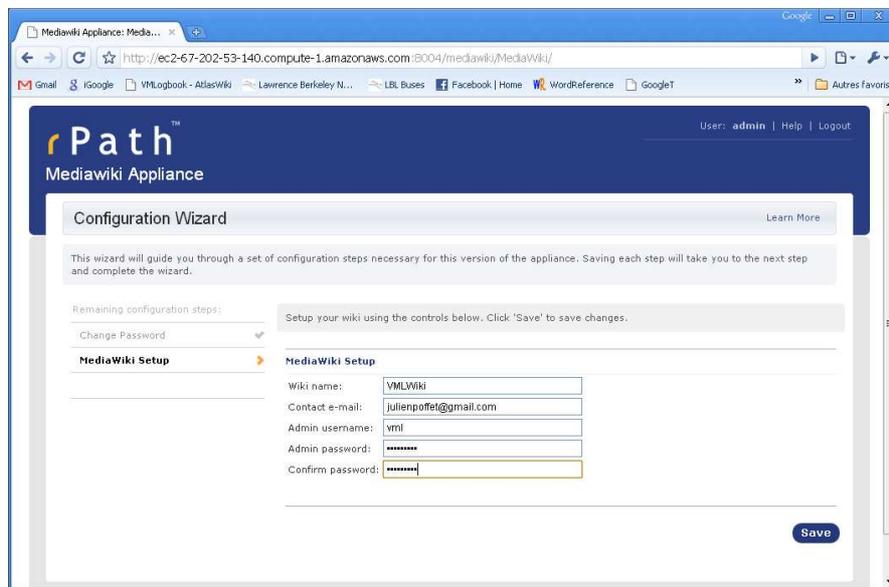


Figure 12 – Virtual Appliance Configuration

After this step, the wiki is ready and fully configured on the virtual machine. To access the wiki, we have just to click on the link provided by the web interface.

To resume here is the steps that we just done:

- Launch a virtual machine on a cloud,
- Access the virtual machine by its web interface,
- Change the default admin password,
- Add the media wiki software to the appliance and configure it,
- Access the media wiki.

### 2.1.7 Open Virtualization Format

Almost all virtualization software uses its own format to package, encode and distribute virtual machines. This means that for now, for instance VMware is not able to run a virtual machine created with Xen. Because of the success of virtualization, now there is a need of a standard way

to package and distribute a virtual machine without regarding to underlying virtualization tools used.

This lead to the idea of a new virtual machines format: Open Virtualization Format (OVF). OVF is both a platform independent, extensible and open packaging specification and a distribution format for virtual machine. For instance we can say that the goal of OVF is to make possible to deploy a single OVF package either Xen or VMware.

For instance with VMware, each virtual machine is described with a VMX file. This configuration file contains information about the virtual hardware used by the virtual machine. The VMX file is proprietary format of VMware. So the idea is to replace this file by an OVF file which is more generic than the VMX one.

Today VMware already offer tools which allow the conversion between a VMware virtual machine and an OVF package and vice versa. With this mechanism we can for instance imagine that a virtual appliance prepared with VMware could be distributed in the OVF format and then converted as a Xen image by passing through an OVF-Xen conversion tool.

There is already an open source library and tools called Open-OVF which support the OVF format. The main goal of Open-OVF is to provide a complete API for creating, using and maintaining OVF packages.

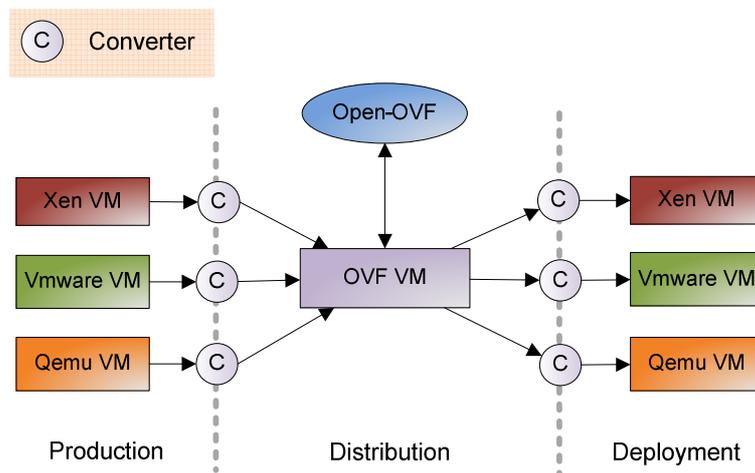


Figure 13 – OVF In the Virtual Machines Lifecycle

Figure 13 shows how OVF is intended to be used in the virtual machines lifecycle. Converters are used to prepare OVF packages for the virtual machines distribution. Before to be used, virtual machines contained into OVF packages must be converted to the target virtualization format.

### 2.1.8 LibVirt

Now there are many virtualization software available on the market. The bests known are of course VMware, VirtualBox and Xen but there is some other software who are less popular such Virtual PC, KVM, QEMU, LXC, OpenVZ, ... With the large number of different software available comes the need to define a common interface between all these platforms. That is the goal of libvirt.

Libvirt comes as a toolkit written in C which works as an intermediate software layer between the user and the underlying virtualization tools. Libvirt offers an API to use all the basic virtual machine management functionalities like start, stop, suspend and snapshot.

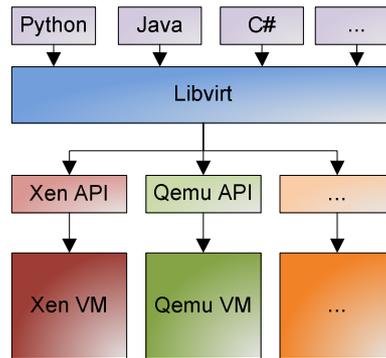


Figure 14 – Libvirt Architecture

The advantage of this additional layer offer by Libvirt is that a program which uses Libvirt will automatically support some different virtualization tools. Otherwise the program has to be adapted for each virtualization tool API. Although Libvirt is written in C, bindings to support other languages than pure C are available, such as for Python, Java, C#, Perl. Figure 14 shows how Libvirt can be used to manage different virtual machines format from application written in multiple programming languages.

At the moment we wrote this report, Libvirt was still in development but very promising.

## 2.2 BACKUPS AND VERSION CONTROL

### 2.2.1 Backups<sup>8</sup>

In computer science, do (or take) a *backup* means make a copy of data and store it. The goal of taking backups is to be able to restore the duplicate when needed. For instance, if the original data has been lost, it is possible to restore the backup. The copy of the data is also called backup.

Taking backups may need a large amount of storage space. Therefore, there are techniques whose objective is to reduce the storage space occupied by backups. If we must do a backup of a file, we are forced to take a full copy of it. But if after this first backup we must take another backup of the same file and the file has not changed, it is unnecessary to store a second identical copy of it.

Let's apply this principle to a folder that contains a large number of files. During the first backup, all the files are copied for backup storage. During the second backup, only the files which have been modified since the first backup have to be stored. Storing the files which have not been modified is unnecessary.

What happens if the original folder is lost and we have to restore the last backup? We have to copy back (or restore) the content of the first backup and then copy back the content of the second one. The second backup cannot be restored without the first one. Because of that, we say that the second one *depends* on the first.

The first backup is a full one, because it contains all the files of the folder. The second one is a partial backup, because in principle it does not contain all the files of the folder<sup>9</sup>, but just a part

---

<sup>8</sup> For more information, see (21)

of them. There are two main ways to take and restore partial backups: incremental and differential backups.

### 2.2.1.1 *Incremental and Differential Backups*

When using incremental technique, each incremental backup contains the files which have been modified since the last backup (any type). This means that each backup depends on the last one. To restore the last backup, it is necessary to start with the first one and then add the content of each following incremental backup. Because the time needed for restoring a long chain of incremental backups may be important, it may be appropriate to take a full backup after a certain number of incremental backups. Then all the chain of backups can be deleted, because we no longer need it.

With the differential technique, we always have only one differential backup. The backup contains the files which have been modified since the full backup. Restoring a differential backup consist always in copying back the files of the full backup and then those of the differential one. Restoring differential backups is then simpler and faster than restoring differential backups. On the other hand, differential backups usually need more storage space. If most of the files have been modified since the full backup, the next differential backups will have almost the same size than the full one. In this case, it is better to take a full backup instead of a differential one, and then delete the differential and the old full backup.

### 2.2.1.2 *Backup Strategy*

In the last sections we presented the three types of backups: full, incremental and differential. They can be combined to increase the performance and save disk space. Let's see how with an example.

Consider we have to keep daily backups of a whole disk. Take a full backup every day is too expensive in term of disk space and performance. Because of that, we create a full backup only once, and then we use partial backups. We have seen in last Section that taking too many incremental backups is not a good idea, because then the recover process becomes too long. Differential backups don't have this problem, but when the amount of data modified on the disk since the full backup is big, differential backups become too big. The problem in this case is the performance: taking the differential backup requires too much time.

A good solution is to combine full, incremental and differential backups. An example of backup strategy is:

- from Monday to Thursday: take incremental backups;
- on Friday: take a differential backup, then delete all incremental backups taken during the week and the last Friday's differential backup;
- on the last day of the month: take a full backup and delete all old full, incremental and differential backups.

With this backup strategy, we always have at least one full backup, zero or one differential backup and zero to four incremental backups (depending on the day of the week). As said before,

---

<sup>9</sup> Unless all the files has changed.

the differential backup always contains the changes since the full backup, and an incremental one contains the changes since the last backup (full, differential or incremental).

If the disk crashes, we can always restore its state as it was the day before. The recover process works as following:

- Restore the full backup: get the state of the disk as it was on the beginning of the current month;
- then restore the differential backup (if it exists): get the state of the disk as it was at the beginning of the current week;
- then restore the chain of incremental backups (if there are incremental backups).

## 2.2.2 Version Control

*Version control systems* (or *versioning systems*) are used to keep a history of the changes done on a set of files. While the goal of the backups is to be able to restore an up-to-date copy of the data, the goal of a versioning system is allow restoring any old version of the data.

A versioning system allows taking files backups. The difference with a “standard” backup is that the one taken with the versioning system has an identifier – a *version* – and is never deleted<sup>10</sup>. A user who wants to restore a backup has to give the version of the backup he needs.

Imagine you do a backup of your files at the end of each day of work and that you need a file you deleted two weeks ago. With the backup system, you can restore the backup you did yesterday, but not an older one<sup>11</sup>. On the contrary, if you were using a versioning system, the day before you deleted the file you would have taken a “versioned backup” of your folder, and now you would be able to restore this version.

Since a version is a backup of data, the full, incremental and differential backups can be used in the versioning systems to gain disk space. The versions are stored into a central repository that is accessible through the versioning system operations.

Applications that implement versioning systems generally offer more functionality than the basic operations (save and restore a version). For example they offer versions history management, the comparison of two different versions, the sharing among users and the concurrent access... Tools like CVS (Concurrent Versions System, (4)) and SVN (Subversion, (5)) are examples of applications that offer a large number of functionality besides the basic version management.

## 2.3 FILE VERIFICATION

### 2.3.1 Comparison By Hash<sup>12</sup>

File verification is used in telecommunication to verify if the transfer of a file is successful. In particular the verification is used to check the integrity of the file (i.e., the content is not

---

<sup>10</sup> Unless the user wants to delete it, of course.

<sup>11</sup> You may have some intermediate backups (differential and/or incremental), but their purpose is not to restore a particular version of your data.

<sup>12</sup> More information can be found in (19), (20).

corrupted by transmission errors) and the authenticity (i.e., the file has not been modified during the transfer). File verification is not only used in telecommunication: it can also be used to check if a file on the hard disk has been modified, for example by a virus or after a hardware crash.

In both examples, we cannot verify the content with a file-by-file comparison: after a transfer, on the local computer we have only one copy of the file and for the verification of a file after a given time, we don't have two copies to compare.

The most popular solution is to compute a *hash value* of the original file and compare this value with the value computed after the transfer or after a given time. A hash is a value computed with a deterministic algorithm using all the bits contained into the file, and can be seen as a sort of fingerprint of the file.

In telecommunication, the hash is computed from the sender before the transfer and is sent with the file to the receiver. The receiver computes the hash on the received file and compares the value with the one got from the sender. If the values are different, the transmission failed. If the values are the same, the transmission probably<sup>13</sup> succeeded. To be able to verify if a file on a disk changes, we must compute the hash value and store it for later comparisons.

The number of possible different inputs of a hash function is bigger than the number of possible outputs. Because of that, *hash collisions* are possible: two different files can have the same hash value. In this case, we talk about *false positive*: the file check result is positive, but it should be negative. Only a comparison bit-per-bit of the source file with the destination file can assure that the destination file is the same than the source. However, hash collisions can often be considered negligible for random file corruptions.

### 2.3.2 Hash Functions and Checksum

There are several algorithms for computing hash values. Each algorithm has different characteristics; some algorithms are designed to be cryptographically secure: this means that it is difficult to *intentionally* modify the content of the file without changing the hash value. Other algorithms don't use cryptographic functions and should be used only to verify files which are not modified with the intention to keep the same hash value.

Another interesting property of the hash functions is the computation cost. Since we need to parse the whole file to compute the hash value, the complexity of the function is typically  $O(n)$ , where  $n$  is the size of the file. However, we can compare the computation cost of different hash functions with  $O(n)$  complexity: different hash functions and even different implementation of the same hash function can have different computational costs.

## 2.4 CERN SPECIFIC SOFTWARE

### 2.4.1 Athena

Athena (6) is the framework used in the ATLAS experiment at CERN for physics data-processing applications. It is composed by a large number of reusable components for the management and control of the experiments on the LHC particle accelerator, for simulations and

---

<sup>13</sup> we cannot assure at 100% that the transmission succeeded by a hash values comparison. The reason is explained below.

data analysis. Other components provide services like the data retrieval and conversion, random numbers generation, configurations management, communication between components, history keeping...

Physicists at CERN develop their analysis applications by combining and configuring the Athena components and by adding their own code, which contains the application logic.

## 2.4.2 CernVM

The complexity and the heterogeneity of the hardware and software environments used by the physicists at CERN often causes compatibility problems. It is often required to reproduce the environment of a machine on another computer, but that could be difficult. The solution proposed by the CernVM project is to use virtual machines to avoid the complexity of reproducing an entire environment.

A virtual machine, as seen in Section 2.1.1, contains a whole environment. The idea of CernVM project is to provide virtual machines in where the physicists will work. Since a virtual machine can be transferred as a common file, the transfer of the entire environment contained into the virtual machine is very simple.

The virtual machines provided by CernVM are simply configurable for the user needs. The VMs are prepared to be used to develop and run analysis on LHC experiments data. Thanks to the virtualization, the virtual machines are independent of the hardware and the software platform (in particular, they are OS-agnostic). This ends up with a portable analysis environment.

### 2.4.2.1 Technology

To reach their goal, the CernVM team used many different tools. One of them is rBuilder from rPath (see 2.1.6.1). With this tool, CernVM can provide virtual machines containing a minimal Linux distribution which includes adequate tools to run the application and experiment software.

The virtual machine provided comes in different format:

- Raw file system image for Xen,
- VMware image,
- Hard disk image for Parallels (Mac) or QEMU (Linux).

The idea of CernVM project is to distribute a very small virtual machine in term of size (about 100 MB). When you download your appliance from the CernVM web site you get a thin OS with minimal software inside it. The software used by the physicist for their analysis may take a lot of disk space (about 10 GB). Of course we don't want to have all this software inside the virtual machine. Basically what they do is to make all these software available on a network location accessible by anyone working in the LHC experiments. This location is called *software repository*. Without entering in the details, the idea is to download and cache only what is needed by the physicist during his work. To do that some networking file system technique are used. Figure 15 shows an overview of this principle.

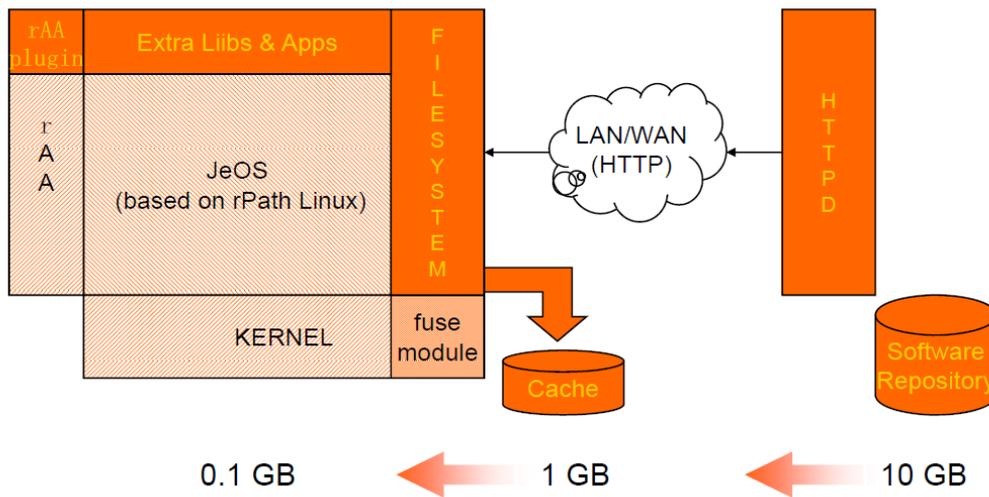


Figure 15 – CernVM Appliance<sup>14</sup>

Thanks to rPath, all virtual machines come with a web user interface (see 2.1.6.3). This interface allows simple configuration management on the VM. The web interface can be extended and customized by adding plugins (see 2.1.6.2). The CernVM team did that in order to add CERN specific behavior to the virtual machine. Here is a non exhaustive list of the functionalities the CernVM plugin provides:

- Configure a first sudo user account,
- Set the root password,
- Select your software requirements for the virtual appliance;
- Set file system options.

Figure 16 shows how the web interface of the rPath appliance with the CernVM plugin looks like.

---

<sup>14</sup> Image source (29 p. 19)



Figure 16 – VM Configuration Through the CernVM Plugin

At CERN there are four different experiments: ATLAS, ALICE, CMS and LHCb. Each experiment has some specific software requirements. Through the CernVM plugin you can set which experiment you belong to, by selecting a virtual organization group.

After choosing your virtual organization group, your virtual machine will be updated in order to meet your virtual group software needs. This process is called *migration* and is provided by rPath. During the migration, software is automatically downloaded from the CernVM website and installed on the virtual machine. Indeed your virtual machine will grow in term of size.

After the migration, your virtual machine is personalized to meet your software requirements. The migration mechanisms is also used to keep the software inside the virtual machine up-to-date.

## 2.5 TAR

Tar is a program which provides the ability to create tar archives. Tar is both the name of the program and the file format of the files it handles. The name tar comes from Tape Archive. This program was initially used for tape backup and other sequential access devices for backup purposes. Today this tool is commonly used to collect some files together into another file called archive. Tar can be used for many purposes, like storage, backup and transportation.

In addition to data archiving, tar can compress the data inside the archive (with an external compression utility) to save disk space. The compression utility can be used in pair with tar by setting some tar options.

### 2.5.1 Incremental Backups

Tar offers a mechanism to perform incremental backups. Remember that the idea of incremental backups is that you save only the difference between now and the previous backup (Section

2.2.1.1 explains the incremental backup mechanism). The first backup will be a full one which contains all the files to save. Next backup will contain only the differences: this is the incremental backup. Each incremental backup is based on the previous one.

Basically what tar do to apply this incremental backup concept is creating an additional file with metadata. This file is commonly called manifest file or snapshot file. The purpose of this file is to help determining which files have been changed, added or deleted since the last backup, so next incremental backup will contain only the modified files. During the first backup, the manifest file is created. Next backup will then be based on this manifest file and so on.

When extracting files from the incremental backup, tar attempts to restore the file system in the exact state it was when the archive was created. Tar will also delete files that did not exist in their directories when the archive was created. This is the normal behavior for the incremental archive. Of course, you can set an option to keep the old files when extracting from an incremental archive in order to reproduce the same behavior than a normal tar archive. To restore a chain of incremental backups, you have to start from the full one and then restore all the incremental backups.

Note that tar uses the files time stamps to determine which files have been modified since the last backup. If the system clock is set backwards for any reasons, the backup mechanism could be unreliable.

A file has three different time stamps: access time, modification time and creation time. The information stored in the manifest/snapshot file is based on these three different time stamps. Usually the access time doesn't really matter when doing backup. This time stamp can be updated when a read access is made on the file. When tar restores the files from an incremental backup, it also restores the time stamps of the files. You can think that it will be a problem for the directories. When tar extracted a whole hierarchy of files and directory, it needs to first create new directory, and then copy the files inside it. When doing this, the time stamp of the directory will be updated. To avoid this, tar maintains a list of the directory created and at the end of the process, when all the files are extracted from the archive, it restores the time stamps of all the directories.

The time stamp that tar will not update during the extraction is the creation time. This doesn't means that tar is not able to restore the exact state of the system. In contrary this behavior is normal. Say you did a backup of a directory, and then you modify some files and create new ones. Next incremental backup must contain all files which have a creation or modification date newer than the last backup date. However, before to take the backup, you extract a tar archive in the directory, and all files in the archive have a modification and creation time older than your last backup. When extracting the files, tar sets the original files modification time, but does not modify their creation time (which is set by the file system to the present, i.e. after the last backup). If during the file extraction tar had set the creation time to the original time, the next incremental backup would not contain the extracted files, which would be wrong.

## 2.6 SSH

SSH (Secure Shell) is a network protocol that creates a secure connection between two network devices. It provides strong authentication and secure communications over unsecure communication channels. This program is usually used to log into another computer over a network in order to launch some remote commands or transfer files to or from this machine.

SSH works on a client/server model. The remote machine must run a SSH server that listen to the SSH port (TCP port #22). Then the client send requests over the SSH client to the remote machine. Both machines can be the server and the client at the same time; that configuration permits to open a two-ways secure communication channel.

Usually the authentication is made by typing a password. This is annoying if you want to automate the authentication mechanism in a script. A solution is to use the key-based password-less authentication.

To setup a key-based authentication, the first thing to do is to generate a pair of cryptographic keys. One is the private key; the other is the public key. The public key will reside on the servers being connected to, while the private key must remain on a secure local area of the client system.

An SSH key-based authentication can be done in one way or in both ways, to establish single or both-ways SSH channels. If you want to setup it in both ways, the public keys must be exchanged between both machines. The password of the server must be provided by the client only at the public key transfer. Once the keys are exchanged, the client is free to log in the remote server without typing a password.

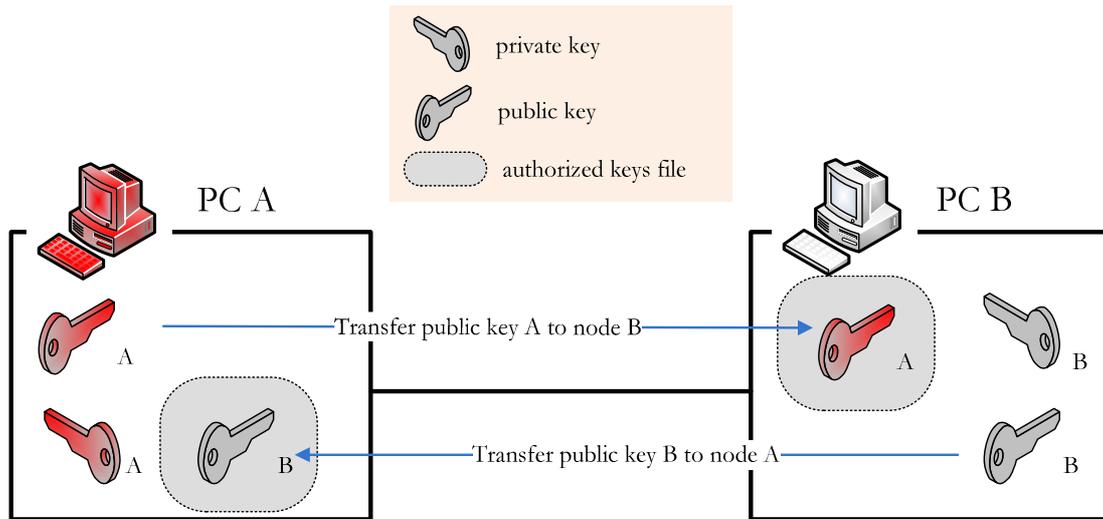


Figure 17 – Public Keys exchange

Even if SSH was primarily used on Linux based operating system to access shell accounts, it works also on the others major operating system such as Windows (with PuTTY) and Mac OS X.

## 2.7 NETWORK FILE SYSTEMS

The file system's goal is to store and organize computer files and their data to ease their research and access. When we talk of network file system, we add the network notion. This means that the file system will work over a computers network. It is usually used to share files, printers or other resources.

Ideally a network file system should appear to its user as a normal file system: the network complexity should be hidden to the user. In this section we present some implementation of network file systems.

## 2.7.1 FUSE<sup>15</sup>

Filesystem in Userspace (FUSE) exports the file system functionality to userspace. FUSE is a loadable kernel module for Unix-like operating systems.

FUSE is particularly useful for writing virtual file systems. The idea is to mount in the file system a directory which point to FUSE. All system calls made by applications or the user to this directory will be handled by the FUSE kernel module.

Figure 18 shows the path of a filesystem call on a directory managed by FUSE. The user launches a command on the directory managed by FUSE (`/tmp/fuse`). The command executes a system call, in this case `stat` (used to get information about a file system object like a file or a directory). Since the system call is addressed to a directory managed by FUSE, it is sent to the FUSE kernel module to be treated. In this example, the FUSE module uses the `./hello` program to retrieve the information about the file system object and return it to the caller.

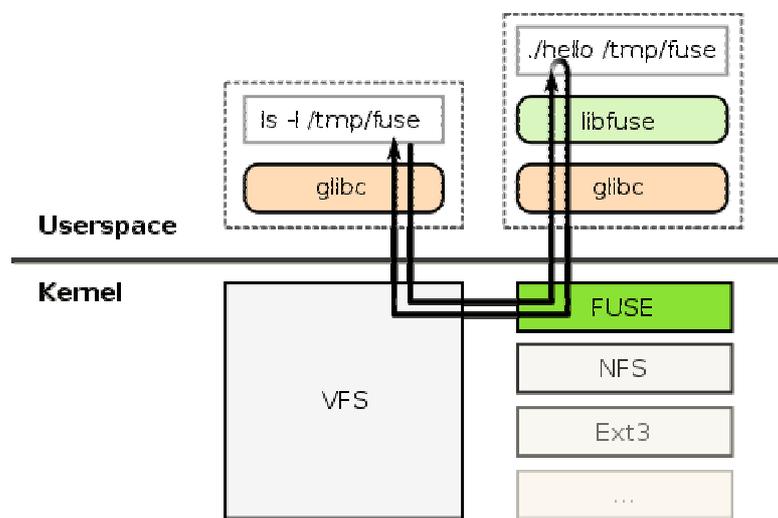


Figure 18 – System Call Path On a FUSE-Managed Directory<sup>16</sup>

FUSE includes these features:

- a library API which allow developers to write their own file systems,
- simple installation (no need to patch or recompile the kernel),
- userspace – kernel interface is very efficient,
- usable by non privileged users.

---

<sup>15</sup> Web site: (30)

<sup>16</sup> Image source: (31)

## 2.7.2 Parrot<sup>17</sup>

Parrot is a tool which acts as an intermediate layer between ordinary programs and remote storage systems accessible via HTTP, FTP or even Chirp (presented in Section 2.7.3). Figure 19 shows an overview of the Parrot mechanism.

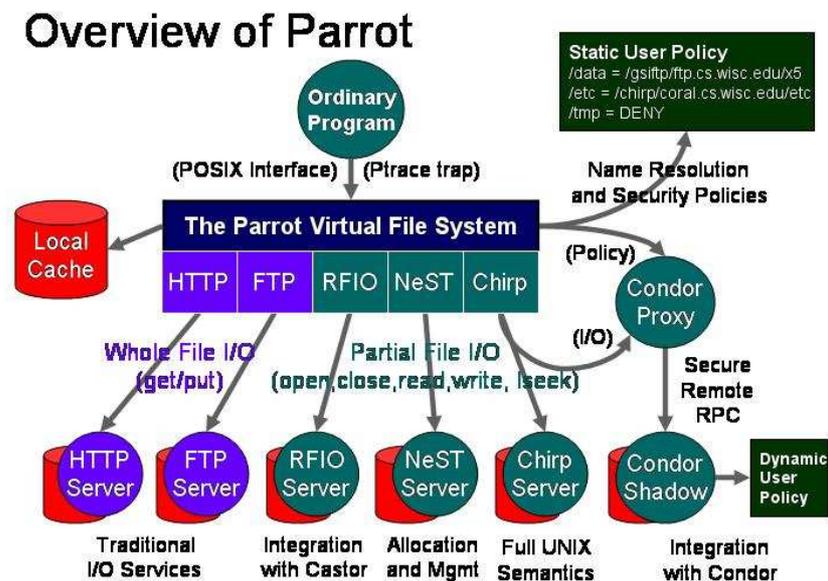


Figure 19 – Parrot overview<sup>18</sup>

Parrot makes a remote storage system appear as a usual local file system to applications. The power of Parrot is that it doesn't require any special privileges, any recompiling or any change to the application that you want to use with.

## 2.7.3 Chirp<sup>19</sup>

Chirp is a distributed file system which is easily deployed without special privileges, and provides strong and flexible security mechanisms. Chirp allows file sharing over the wide area network. This tool is commonly used in grid computing to share data between many computers to solve data intensive problems.

Chirp works in a client/server model. The user who wants to share data on his computer simply launches a Chirp server. The Chirp server makes available on the network a part of the server's file system. Then a Chirp client will be able to access easily this file system remotely.

One of the major advantage of Chirp is that it doesn't require any sort of administrator privileges to access the data. This means that you can connect a Chirp client to a Chirp server without having a valid account neither a password on the remote server. Chirp gives the possibility to the

<sup>17</sup> More information can be found in (32)

<sup>18</sup> Image source: (33)

<sup>19</sup> More information can be found in (34)

user to set a fine-grained access list (ACL) on the server to control that the data will be shared only with allowed people.

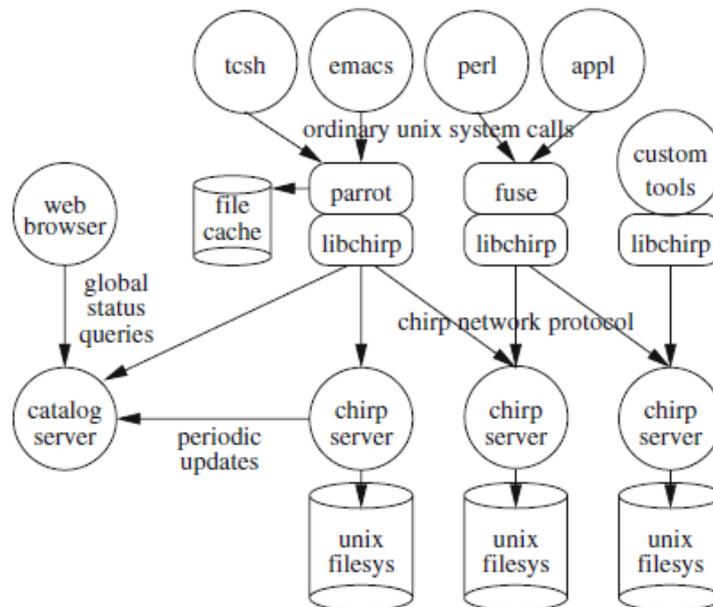


Figure 20 – Overview of the chirp filesystem<sup>20</sup>

Figure 20 shows the main components of Chirp. Chirp can be used in many ways. A library called *libchirp* is provided and allows clients to program directly to the Chirp protocol. This could be useful if we are writing a program that need to control finely the Chirp protocol, but the workload could be too important. Instead of doing this, Chirp encourage user to connect an adapter to their library such as Parrot or FUSE (see Sections 2.7.1 and 2.7.2). These adapters will mount the file system shared by the Chirp server as an ordinary local filesystem. This is the best method if you want to make tier software work with the remote file system without writing yourself the invocations from your software to *libchirp*.

Chirp also provides a command line tool (Chirp client). This allows you to connect to a server, copy files and manage directories, much like an FTP client.

Chirp also send periodic updates on a well-know catalog server. The catalog server can be accessed by a web browser via HTTP to obtain a global list of available servers.

#### 2.7.4 SSH FS

SSH FS is a filesystem client based on the SSH file transfer protocol (see Section 2.6). It allows a secure access to remote files. The implementation of SSH FS uses FUSE (see Section 2.7.1). Since most servers run an SSH daemon, the use of SSH FS is pretty simple: you just need to login to the remote server in order to mount the remote filesystem locally. Then the files in this mounted directory will appear just as if they were local files.

<sup>20</sup> Image source: (35 p. 4)

## 2.7.5 File Systems Synchronization

File systems synchronization in computer science refers to the technique used to keep two or more locations up-to-date with the same files. If a file is added, changed or deleted in one location, the other locations will be updated with this modification.

The synchronization can be one-way or both-ways. If the synchronization is both ways, the most up-to-date files will be available at both locations, regardless of where they were modified.

A synchronization one-way is commonly called mirroring. This means that the files will be synchronize only from a source location to a target location.

To synchronize two directories a solution could be to copy each time the entire directory. Indeed, this is not an efficient way to do it. A good sync program will compare the differences between the two locations and synchronize only the differences. To be able to that remotely, a server has to be installed on the remote computer. To gain time and to limit the traffic flow over the network, the differences can be compressed before the transfer over the network.

### 2.7.5.1 *Rsync*<sup>21</sup>

Rsync is an open source tool to perform fast incremental file transfer. Rsync use a algorithm which allows transferring very efficiently the difference between a file already present at the destination location and its new version in the source location.

Rsync is able to copies files either to or from a remote host, or locally on the current host. Rsync cannot be used to transfer files between two remote hosts. One of them must be local.

### 2.7.5.2 *rdiff-backup*<sup>22</sup>

Rdiff-backup is backup tool written in python. The goal of rdiff-backup is to take the best features of mirroring and incremental backups. Rdiff-backup is built on top of librsync (the implementation of Rsync). This software can be used to take incremental backups of a part of a remote file system. The first backup will contain all the files and then, thanks to the Rsync algorithm, only the differences will be transferred.

The directory which contains the backup files is called repository. Rdiff-backup saves extra reverse diffs in a special directory. These additional files allow rdiff-backup to keep a version history and come back to an old version. This is one of the main advantages of rdiff-backup. This allows you to ask rdiff-backup to restore from your backup directory all the files as they were 5 days ago for instance.

Another advantage is that the repository contains the mirrored arborescence of the source location. In other words, the target directory ends up a copy of the source directory, plus the extra reverse diffs. This means that if you want to restore your last backup you can just copy back the top directory in the desired location. You don't have to use rdiff-backup to restore these files. You must use the tool only if you want to restore a specific old version of the directory. In this case the tool will apply the differences stored in the repository to recreate the old file.

---

<sup>21</sup> Web site: (27)

<sup>22</sup> Web site: (26)

Rdiff-backup needs a POSIX operating system. Rdiff-backup works on Windows, but this configuration is less well tested than the UNIX version which is considered stable.

### 2.7.5.3 Archfs

Archfs is a tool to use in pair with rdiff-backup (see 2.7.5.2). The incremental backup system used by rdiff-backup is very efficient to save disk space, however it has its drawbacks. Users can access files from most recent rdiff-backup easily by just accessing the backup directory. Browsing and displaying the oldest version is more painful.

The aim of Archfs is to build a userspace filesystem by using FUSE (see 2.7.1) and display in a friendly manner the content of the rdiff-backup directory. The file system mounted by Archfs is read-only so it can be used only to browse the backup.

## 2.8 COMPUTER SCIENCE THEORY

### 2.8.1 Design Patterns

In software engineering, a design pattern is a general reusable solution to solve generic problems. We used a couple of patterns in our application. So we will present here their concepts.

Singleton: the singleton pattern is used when you need that there is only one instance of a specific object.

Factory method: The factory method pattern deals with the objects creation. It allows to creates an objects collection without specify exactly the classes.

### 2.8.2 Trees

In computer sciences, trees are a very common data structure. Trees are a useful and efficient way to represent data with a hierarchical structure. A tree is composed of nodes, which holds the information, and arcs (or edges) for the connection between the nodes. Each node can have zero or more nodes descending from it. The most bottom nodes are called *lead nodes*. The top most node is the *root node*. All other nodes which are not leaf nodes are then *body nodes*.

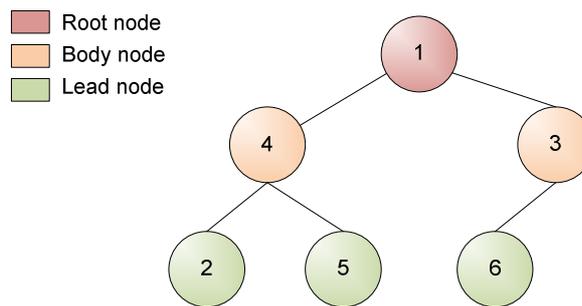


Figure 21 – Tree

Figure 21 is an example of representation of a tree. The nodes are represented with the circles and the arcs by a line connecting two nodes.

### 2.8.3 Graphs

A graph is a kind of data structure in computer science. It consists of a set of nodes (or vertices) and edges. The edges establish the relationship between the nodes, but there is no hierarchic relationship between nodes in a graph.

A graph can be directed or undirected; the direction is a characteristic of the edges. In an undirected graph, if there is an edge between two nodes A and B, A can be reached directly from B and vice versa. With a directed graph, we can specify that the two nodes are connected only in the direction of the edge but not in the opposite direction.

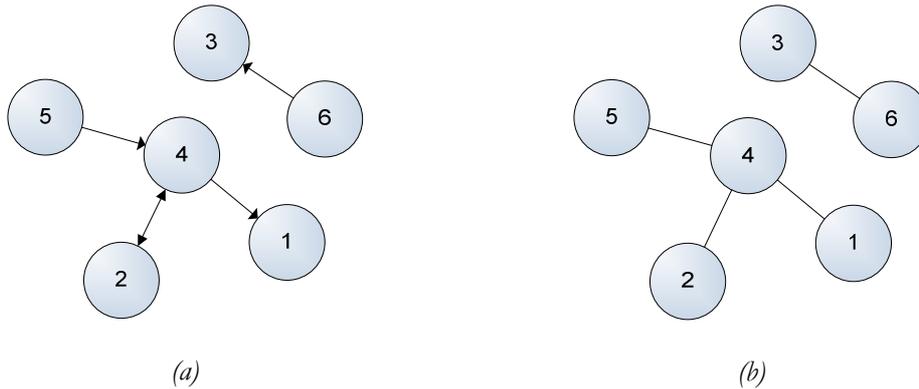


Figure 22 – (a) Directed and (b) Undirected Graphs

Figure 22 shows two examples of directed and undirected graphs. In the first example, the direction of the edges is showed with the arrows. There is a lot of application for the graphs: network flows, route problem, covering problem ...

## 2.9 TECHNOLOGY

### 2.9.1 Python<sup>23</sup>

Python is both a very-high-level language (VHLL), and an object-oriented dynamic language (OODL). Python is almost ideal as a scripting language interface for modern systems and as a stand-alone language. This means that python may be used as well for building applications from scratch as for executing some task needed by another system. Python is also often used as a glue-language to combine components written in other language, like in C++.

Python is highly readable and the syntax is at most simple. The dynamic typing and binding in Python makes everything easier and increases the productivity in applications development. Python allows the organization of the code in package and module; this functionality maximizes the code reuse and the program modularity.

Most people like Python because it provides productivity increasing. Since the programmer doesn't need to explicitly compile Python code, the development cycle is extremely fast. Thanks to its high-level built-in type, a Python program may be three to five times shorter than the same

---

<sup>23</sup> For more information, see (22) and (23).

program written in Java. This factor may increase to ten if we compare a program written in Python with the same program written in C++.

Debugging a Python program is very simple. A bug or a bad input will always raise an exception. The quickest way to debug a Python program is to introduce a few print statements in the code.

Another important advantage of Python is its portability. A Python program can be interpreted in a number of operating systems, including Unix-based system, Mac-OS and Windows.

Python is absolutely free, even for commercial use. This means that you can sell a program written in Python without pay any licensing fees.

### 2.9.2 XML

XML stands for extensible markup language. XML is a general specification to allow the user to specify its own language to describe his data. This technology is widely used and has a lot of powerful applications, particularly for the management, display and organization of data. One of the main advantages of XML is that an XML file can be read both by human and machine.

XML is particularly suitable for configuration files. It is easy to extract the information because there are programs called parsers which are able to read XML syntax and get the information out for us. It is also possible to apply a transformation to a XML file. This can be useful in order to display the information contained into the XML file or to transform the structure of the XML file in a different one.

Beside the easy way to extract information, the control of the structure of an XML file is quite simple and efficient. This control is useful when a human has to introduce or modify information in that file, but also when the XML file is modified by a program.

### 3 DESIGN

(COMMON)

#### 3.1 VML MAIN COMPONENTS

The physicist works with virtual machines and wants to save the state of his VMs during his work. In VML, folders which contain the virtual machines files are called *working areas*. When VML saves the state of a virtual machine, it actually copies data from the virtual machine and stores some metadata into another folder called *repository*. The virtual machine data and the additional metadata compose a VML entry.

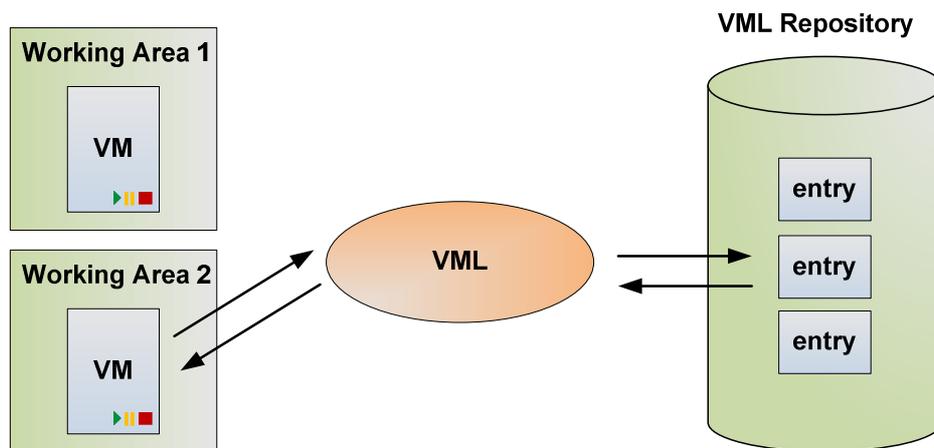


Figure 23 – General Architecture of VML

Figure 23 shows the general architecture of VML with the main components. The virtual machines in the working areas can be directly used by the physicist (i.e., he can start, stop and suspend them). A working area can contain zero or one virtual machine, and the user can have more than one working area.

The entry into the repository corresponds to a virtual machine in a specific state, but the user cannot directly use them from inside the repository. If he wants to restore an entry, he must ask VML to do it. VML then extracts the entry into a working area and the physicist can then use the virtual machine.

To summarize, the two fundamental processes done by VML are:

- save a virtual machine: VML copies some data from the virtual machine and creates a new entry into the repository;
- build an entry: VML extract the entry from the repository and recreates the virtual machine inside a working area.

VML is a versioning system adapted for virtual machines. Each VML entry correspond to a version of the virtual machine and each version can be restored – not only the last one.

### 3.1.1 Entries

The VML repository contains entries, which are descriptions of virtual machines at a given state<sup>24</sup>. One of the VML central component is the VM-entry converter. It provides the conversion in both directions: it can transform a virtual machine to an entry and an entry to a virtual machine.

Basically, the VML repository could contain entire virtual machines instead of “complex” entries. The problem is that a whole virtual machine may need a large amount of disk space to be stored. One of VML’s goals is to reduce this size by not storing redundant information. VML stores only some parts of the virtual machine into the repository. To recreate a virtual machine from these parts, VML needs some metadata which explain how to rebuild the original VM. These “parts of VM” and the metadata compose a VML entry.

VML cannot always store only some parts of a virtual machine. Sometimes it is necessary to store the whole virtual machine into the repository. An entry which contains a whole virtual machine is called a *full entry*.

An entry which does not contain a whole virtual machine is called a *partial entry*. VML can create a partial entry only from a virtual machine which respects a pre condition: the parts of the virtual machine which will not be stored in the repository can be found elsewhere<sup>25</sup>. This pre condition is necessary to make VML able to restore the partial entry.

### 3.1.2 Repository

The repository is the “back-office” of VML. It stores different environments used by physicists. His goal is to manage the entries and they related virtual machines files or environment files.

More precisely the repository contains the files which compose the entries and also the metadata of the entries. The files managed by the repository may represent virtual machines, an environment or something else. This will not make a difference on how the repository will store this information.

Note that the repository is used only as a storage system. This means that a file stored in the repository will not be modifying. Also a virtual machine will never be started inside the repository but outside, in a working area. The only action permit with repository is to add an entry to be stored, retrieve an entry, delete an entry or consult the content of the repository.

---

<sup>24</sup> Here “state” does not refer to the running, paused or shut down state of the VM. With “state of a virtual machine” we refer the VM’s hard disk and RAM contents and the virtual machine specific characteristics, like the configuration file and the other information used by the virtualization platform to manage the VM.

<sup>25</sup> For VML, “elsewhere” is another entry in the same repository or, in special cases, downloadable from a known website.

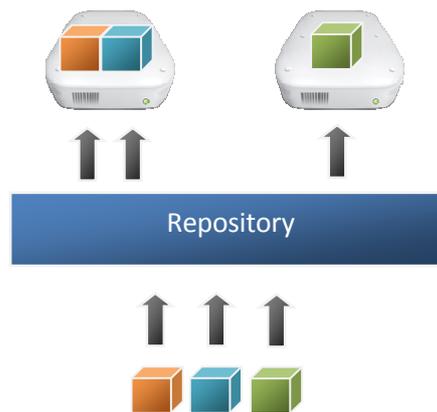


Figure 24 – Adding Entries in the VML Repository

Because of the size of the files, the folder used by the repository will grow very quickly. For this reason, the repository has to be flexible as most as possible. It has to allow that the storage of the entries can be hold by multiple physical disk or partition in order to dispatch the disk usage consumption.

### 3.1.3 Working Area

A working area is a folder on the host machine's file system which contains the virtual machine files. The purpose of working areas is to simplify the VML user commands, by avoiding the use of full paths in the most used VML commands. A working area has a simple identifier (its name) and identifies a folder. The working area name can then replace the folder path in the VML commands.

A working area can be empty (this means there is no virtual machine files in its folder) or contain a VM. Working areas are used by VML in both main operations, "save a VM" and "build and entry". When saving a VM, the virtual machine files are contained in the working area folder. When building an entry, the virtual machine is built in the working area folder.

The user is free to set as many working areas as he needs.

## 3.2 USE CASES

We took the time in the previous section to talk about the different components of VML. Now we will specify more precisely what VML is able to do. For that we made a use case diagram to split VML into separate functionality. Figure 25 show the main group of VML's functionalities.

One of the two VML's key functionalities is the use case "**save new entry**". This use case represents the part of VML which is in charge of taking a backup of the working environment of a particular virtual machine. If the new entry must be placed in a new project, VML executes the use case "**add project**" to add a new project for the entry.

The second most important use case is "**open entry**". This use case offers the possibility of opening the entries present in the repository. This use case is done by restoring the entry and starting the virtual machine. This part of VML has to communicate with the virtual machine.

Use case "open entry" uses another use case, "**restore entry**", which can be executed to build the entry without automatically starting the virtual machine with VML.

The use case “**consult**” displays information about the objects managed by VML, such as the repository, the entries, the content of the working areas and some configuration parameters.

The next one is “**export entry**” which offers to the user a way to extract a specific entry from the repository. This functionality is one of those who will make possible the work sharing between the physicists.

In pair with the export we have “**import entry**” which takes in input the output of the use case “export entry”. This use case permit the importation of VML entry into the repository. An entry imported could also be directly built with the use case “restore entry” and opened with “open entry”.

The use case “**get CernVM**” offers to the user the possibility of downloading a specific version of CernVM by typing a simple VML command. The main goal of this functionality is to offer to the physicist a quick way to start his work with CernVM. Browsing the CernVM website to find a suitable version of their virtual machine will be no longer necessary. With the help of the use cases “import entry”, “restore entry” and “open entry”, the user can directly import, restore and/or open his virtual machine.

With the use case “**consult available CernVM**”, the user can consult the version and formats of CERN virtual machines available on the CernVM website.

The use case “**delete entry**” holds the mechanism for the entries deletion by taking care about the eventual dependency between entries.

The user can easily delete multiple entries by using the use cases “**delete repository**” and “**delete project**”. The first one deletes all entries in the repository, and the second one deletes all the entries in a specific project.

These were the most important functionality offered by VML. There is another group of functionality who deals not directly with the entry management but more with the configuration of VML, regarding to where the virtual machine will be run and where they will be stored.

The use cases “**add repository/working area**” are used to add a new repository respectively a new working area in VML. Then it can be removed from VML by using the use case “**delete repository/working area**”. Finally we have the uses cases “**set default repository/working area**” which as its name says, set the current default repository and working area.

The last use case is “**setup VML**”. This use case will start a little wizard which is supposed to help the user to create the first repository and the first working area.

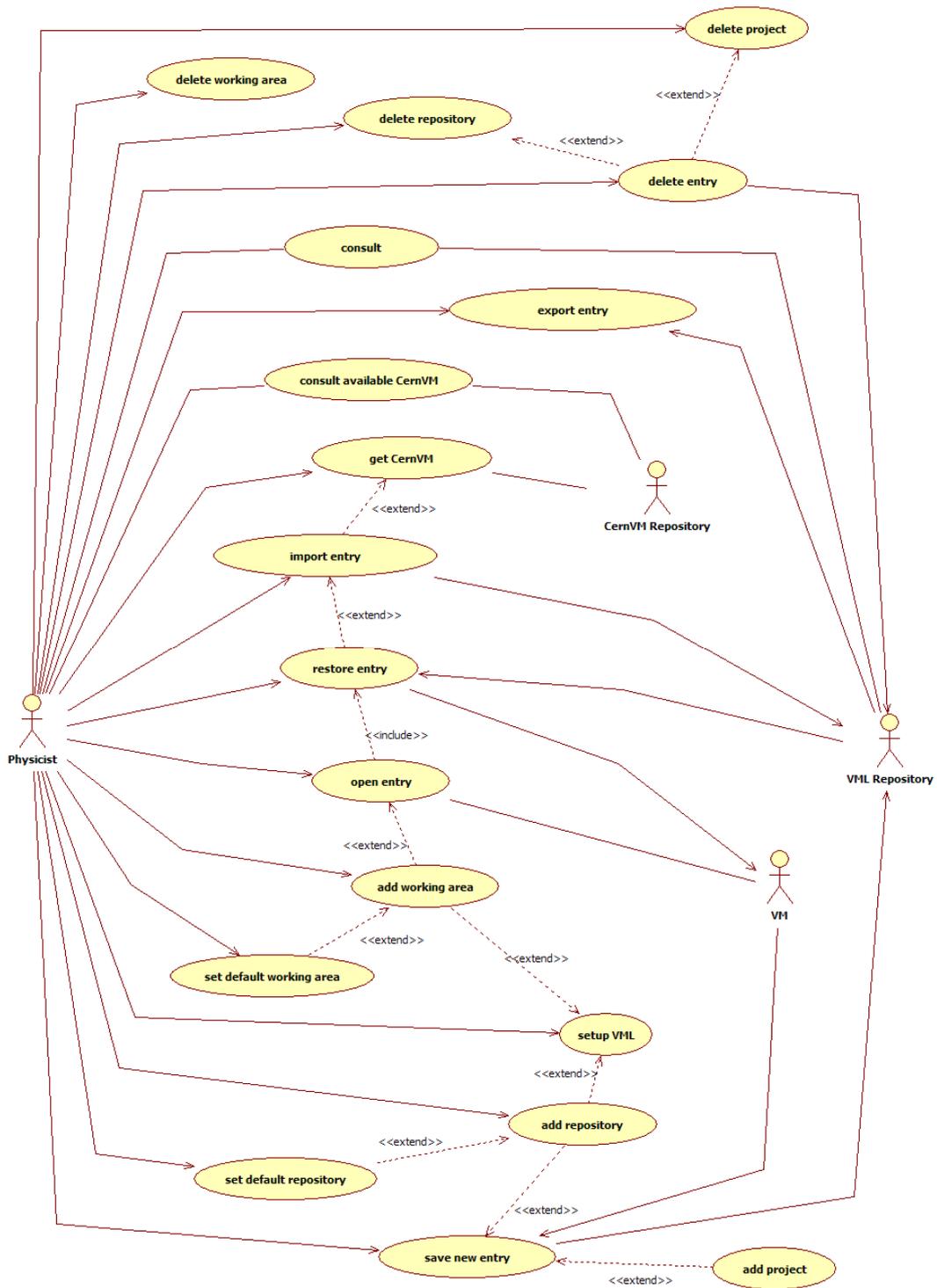


Figure 25 – VML Use Case Diagram

### 3.3 STORAGE AND RESTORATION OF THE ENTRIES

VML has different ways to create partial entries. The next sections explain the different techniques used by VML to store full and partial entries and to rebuild the virtual machines from these entries.

#### 3.3.1 Incremental Backups

Incremental backups is a technique used to reduce the space occupied by data backups. This technique is used by VML for the entry creation. The idea is that at each time the user saves a virtual machine, VML stores only the differences from the last backup. Figure 26 shows an example of an incremental backup sequence. In the example, the physicist works in a virtual machine and during his work, he wants to save the state of the VM.

At the beginning, the virtual machine is in a state S1 and the physicist executes the “save” command of VML. To be able to restore this virtual machine, VML needs to store a whole copy of the VM into the repository, so the entry E1 which correspond to the VM state S1 contains the entire machine.

The physicist continues his work into the machine and then executes another VML “save” command. At this moment the virtual machine is in the state S2. Probably only a few amount of data changed from the state S1, so VML does not copy to the repository the entire VM, but only the differences  $\Delta 1$  between states S1 and S2. These differences compose the new entry E2. The other part of the virtual machine (S1) is already in the repository, so VML doesn’t need to save it. The disk space occupied by E2 is probably much less than the space occupied by E1.

The physicist works in the machine, which is now in the state S3. S3 is composed by:

$$S1 + \Delta 1 + \Delta 2 = S2 + \Delta 2 = S3$$

When the user asks VML to save this state, VML stores into the repository the difference  $\Delta 2$  between states S2 and S3 and calls this entry E3. At this moment, the repository contains three entries: E1, E2 and E3. Each entry corresponds to a different virtual machine state. E1 is a full entry and E2 and E3 are partial entries.

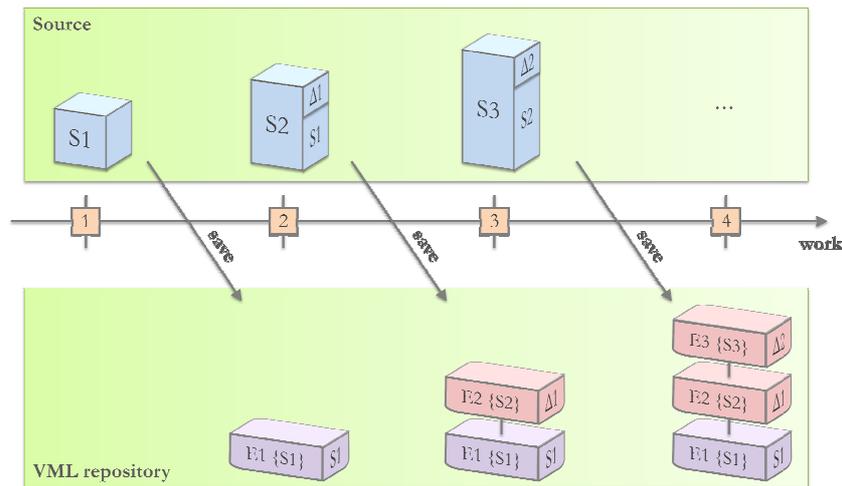


Figure 26 – Incremental Backups in VML

Figure 26 shows how the virtual machine and the repository evolve during the work of the physicist. The cubes upside the arrow represent the virtual machine. On the front of the cube is the name of the state of the VM.

The physicist works in the virtual machine and we represent this by increasing the size of the rectangles. By working in the virtual machine, the physicist modifies its state. The new state  $S_n$  of the virtual machine can be seen as the composition of the last state  $S_{n-1}$  and the difference  $\Delta_{n-1}$ . This composition is represented on the side of the cube.

The part of the figure below the arrow is the VML repository. Cubes in this area are entries saved into the repository. The entry name is written on its front, followed by the name of the VM state represented by the entry (inside the curly brackets). On the side of the cube is written the content of the entry. For example, a cube with  $E2 \{S2\}$  on the front and  $\Delta1$  on the side means that the entry  $E2$  represent the virtual machine at state  $S2$  and that the entry itself is composed only by the difference  $\Delta1$  between state  $S1$  and state  $S2$ .

At each step of Figure 26 is represented the state of the working area and the content of the repository. Let's consider the content of the repository at step 4. If the physicist wants to restore the state  $S1$ , VML will copy the whole content of  $E1$  to the target (i.e., where the entry must be built) and the machine is restored. However, if the physicist wants to restore the state  $S2$ , VML must also copy the whole content of  $E1$  ( $S1$ ) to the target and then add the content of  $E2$  (the difference between states  $S1$  and  $S2$ ). If the physicist asks to restore the state  $S3$ , VML must do the same than for the state  $S2$  and then add the content of  $E3$ .

### 3.3.2 Differential Backups

Incremental backups is not the only technique which can be used to create partial entries. Let's imagine a physicist who works on a virtual machine and who uses incremental backups of the VM. After a year, he may have tenth of entries in his repository. Before to restore the entry  $n$ , VML must restore the entry  $n-1$ , which needs that the entry  $n-2$  has been restoring... and so on to the entry 1. Restore an entry may then take too much time.

A solution of this problem comes with the differential backups. With this technique, all the differential entries refer to only one entry which is stored into the repository. We call this entry the *base*. When VML must build an entry, it has just to restore the base entry and the partial one.

Figure 27 is the equivalent of Figure 26, but in this case VML uses differential backups. The entry  $E1$  is the base image, and  $E2$  and  $E3$  are the partial entries. We what changes from the incremental backups at step 3. Instead of taking a difference between  $S2$  and  $S3$ , VML stores the difference between  $S1$  and  $S3$  into the entry  $E3$ .

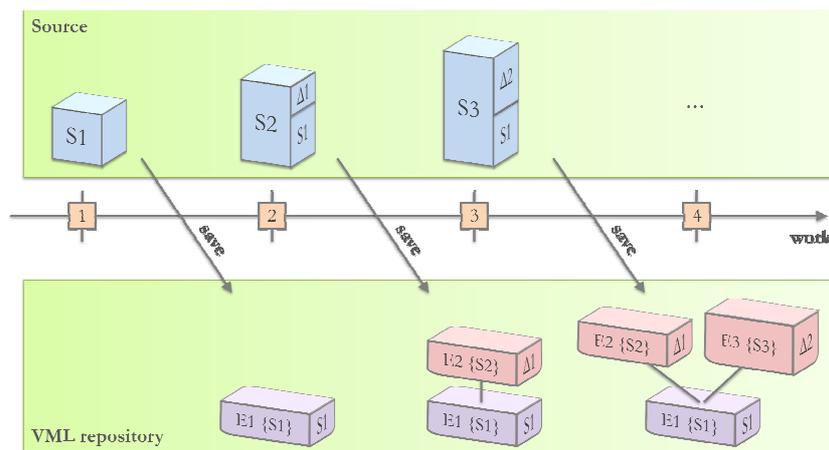


Figure 27 – Differential Backups in VML

### 3.4 DEPENDENCIES BETWEEN ENTRIES

#### 3.4.1 Entry Tree

Let's consider the three entries created during the example in Figure 26. E1 was the full entry, and E2 and E3 were two partial entries taken with the incremental backups. To restore E2, VML needs E1. To restore E3, VML needs E2 and E1. We say that E3 *depends* on E1 and E2. E2 also depends on E1. Note that an entry can depend on more than one entry (like E3), but an entry *directly* depends always on a single entry (or zero)<sup>26</sup>. When entry E2 directly depends on entry E1, we say that E1 is the *parent* of E2 and E2 is E1's *child*. In Figure 26 and in Figure 27 we show this dependency with a connection line between the two entries.

We can represent the dependencies between entries with trees: at the top we have the parent entries and at the bottom we have the partial entries which depend on the parents. The content of the VML repository can be represented with a collection of trees, as show in Figure 28.

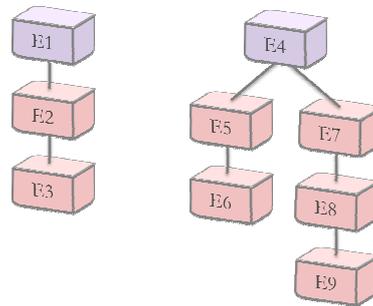


Figure 28 – Tree Structure of the Entries in the VML Repository

The tree on the left side of Figure 28 is generated by a sequence of incremental snapshots. The tree on the right is more complex. The following sequence of actions creates the tree on the right:

- Save the state of the VM. This creates the full entry E4.
- Save twice the state of the VM with incremental backups. The first entry (E5) directly depends on E4. The second entry (E6) depends on E5.
- Save the state of the VM with a differential backup. The differential entry (E7) contains the changes since the last full entry E4, so it becomes E4's child.
- Save twice the state of the VM with incremental backups. E8 and E9 are added to the entry tree.

#### 3.4.2 Adding New Branches in the VML Entry Tree

Taking a differential backup seems the only way to create a “brother” entry like E7 in Figure 28. Incremental backups can only add a child entry to the last one, and a full backup create a whole new tree. It is true that a differential backup creates a new branch on the tree, but the new entry is always a root's child. Then how can we create a tree like the one in Figure 29, with two brother

---

<sup>26</sup> This is true as long as we don't have two entries which are exactly the same into a repository. Even in this special case, we can simplify and consider that a third entry only depends on the first or the second of the two twin entries.

entries (E12 and E14), children of a non-root entry (E11)? It is not possible using only the “save” command, but it is with the “restore” command.

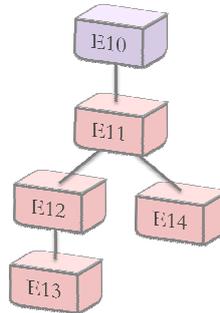


Figure 29 – Example of Entry Tree With a New Branch

Below is an example of actions sequence which could have generated the tree:

- Save the state of the VM. This creates the full entry E10.
- Save three times the state of the VM with incremental backups. Entries E11, E12 and E13 are created.
- Restore the state of entry E11.
- Save the state of the VM with an incremental backup. Since we restored entry E11 before, the new entry is added as a new child of entry E11 and not as E13’s child.

The VML user is then free to come back to an old state of his virtual machine, use it and then save the state, without affecting the entries already stored in the repository. He can continue using and saving both “original version” and “new revision” of the same virtual machine.

### 3.4.3 Build Path

When the user wants to restore a specific entry, VML has to choose which entries are needed to reconstruct the target entry. We called that the *build path*. The build path is a list of entries.

Let’s take the example shown in Figure 30. We want to build entry #4 from scratch. When we say from scratch, it means that the entry is built starting from the first (full) entry of the build path. This is the case when the user wants to build the entry in an empty working area, for example.

The tree on the left of Figure 30 shows the dependencies between entries and, in red, the entry we want to build. On the right is the resulting build path, highlighted in blue and indicated with the arrows. If we take a look at the entries dependencies in Figure 30 we can see that the build path will be: #1, #3 and #4. So to reconstruct entry #4, VML will before reconstruct the entry #1, then entry #3 and finally entry #4.

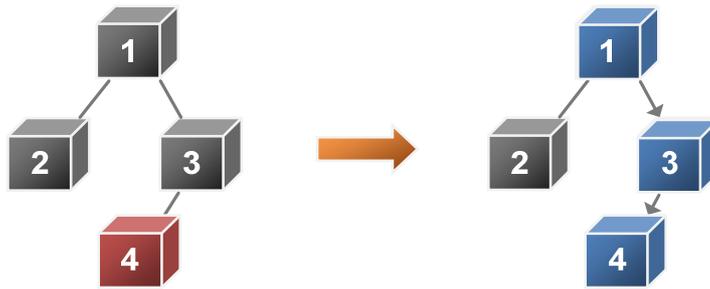


Figure 30 – Build Path from Scratch

Now if we take the case of a not empty working area, the build path computation will be different. In fact, if the working area already contains an entry, VML will try to use it – if possible – to save time during the entry reconstruction. Let's take another example. Figure 31 shows that we want to reconstruct entry #6, but entry #3 is already present in the target working area. So in this situation the build path will be: #3, #4 and #6. This means that VML will directly build the entry number #4 on top of entry #3 and ends with the reconstruction of entry #6.

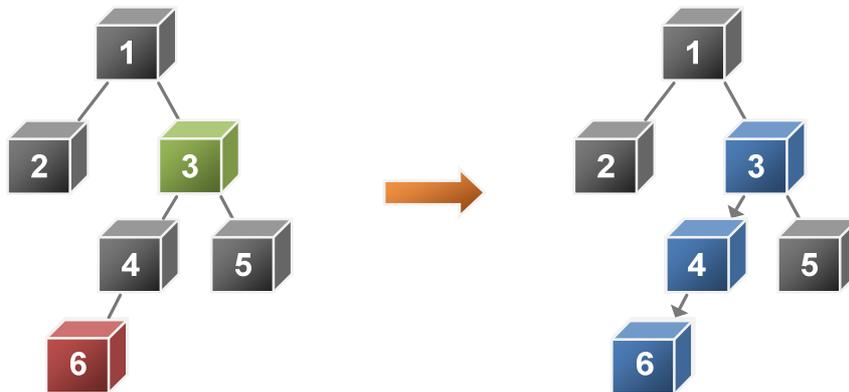


Figure 31 – Build Path from Entry

If we had to reconstruct the entry #6 from scratch, the build path would have been: #1, #3, #4 and #6. If we had to reconstruct entry #2, the build path would have been #1 and #2: the fact that entry #3 is already in the working area does not help for the entry reconstruction

#### 3.4.4 Entry Removal

The disk space needed by the repository may grow very quickly. The entries itself can occupy a big amount of space on the disk. For these reasons, VML provides a functionality to delete the entries. However, because of the dependency between entries, the entries should not be deleted without taking some precautions.

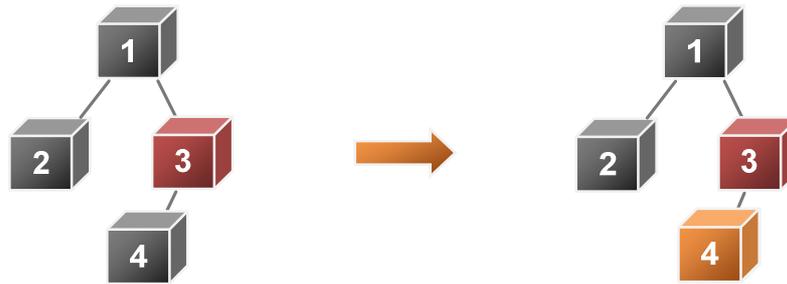


Figure 32 – Entry Removal – Fail

Let's take a look at Figure 32. This example takes the case when the user asks to delete an entry which has dependency. The user wants to delete entry #3. The problem is if VML delete entry #3, VML will no longer be able to reconstruct entry #4. In this case, before to actually delete the entry, VML warns the user that if he chooses to delete entry #3, he will be no longer able to restore entry #4.

VML allows the user to delete more than one entry at once. In this case, VML can compute all dependencies and see if the list of entry to delete will causes construction problems for the other entries. Figure 33 shows that the user wants to delete entries #3 and #4. They can be deleted without any problem because the other entries don't need them to during their reconstruction.

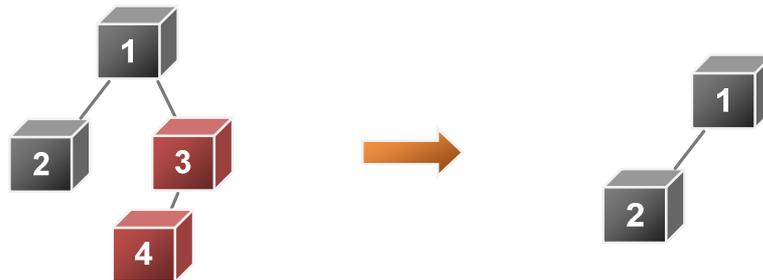


Figure 33 – Entry Removal – Success

### 3.4.5 Entry ID

The purpose of this Section is to explain why VML needs to identify virtual machines and entries, and how VML does that.

VML manages the repository content (the entries), but it doesn't have the control on the virtual machines – the sources of the "save" VML command and the targets of the entry reconstruction. This means that after the user restores the state of a virtual machine using VML, he can do whatever he wants on this VM: modify its content, replace it with another virtual machine, delete it...

What happens if the user restores the state of a virtual machine using VML, then deletes the VM and replaces it with a different one, and finally wants to save this new virtual machine with VML? VML should not assume that the virtual machine the user wants to save is the last restored one. VML must discover which virtual machine the user wants to save: it is a new one, never saved before, or is maybe a VM which has previously been saved?

The first case is the simplest one: VML must just take a full backup of the virtual machine. The second case is different: if the VM has previously been saved, at least one entry in the repository matches this virtual machine. VML must then find this entry, in order to take an incremental

backup based on it. How can VML recognize the right entry? VML needs to relate the virtual machine with the entry in the repository: this is done by assigning an identifier (ID) to each entry in the repository, which is the same than the identifier of the virtual machine.

We don't discuss in this section how the virtual machine ID is read or computed, because is implementation dependant. The basic concept is that VML uses IDs to know if there is a relationship between an entry in the repository and a virtual machine, and at the creation of the entry, VML establish this relationship by assigning the virtual machine's ID to the new entry.

Let's see what happens in state 3 of Figure 26 (represented here in Figure 34). At this moment, VML has to save the virtual machine at state S3. It has to different ways to do it:

- save the whole virtual machine (S2 + Δ2) or
- if the base S2 is already in the repository, save only Δ2.

We know that:

$$S1 + \Delta1 + \Delta2 = S2 + \Delta2 = S3$$

and that S2 can be reconstructed by building E2.

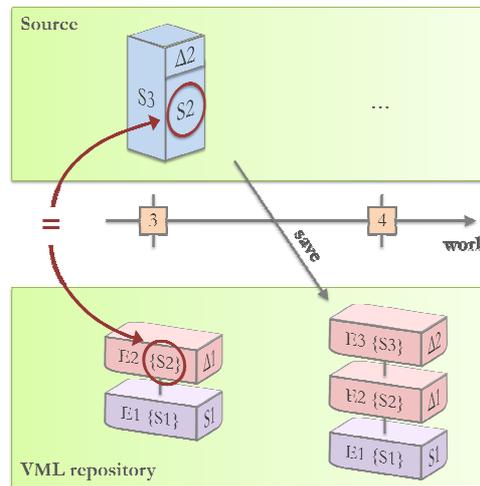


Figure 34 – Comparison of a Virtual Machine With the Entries in the Repository

S2 is actually already available into the repository: entry E2 corresponds to it. Even if entry E2 doesn't directly contain S2 (it actually contains only a part of S2, Δ1), we know that we can recreate it by building E2 (which also needs E1). Now, when saving the virtual machine at state S3, VML should take the right decision: copy only Δ2. To be able to take this decision, it needs to realize that S2 can be reconstructed by building E2. E2's ID is the same than the ID of the virtual machine at state S2. By comparing the IDs, VML can recognize that a portion of the virtual machine (S2) is already available in the repository.

Note that the ID identifies the state of a virtual machine disk, and not its memory state. This was a choice taken during the design of the application: all VML entries contains data for the virtual machine's disk reconstruction, but only some VML entries can also restore the VM's memory state.

---

## 8 BIBLIOGRAPHY

---

1. **Williams, David E. and Garcia, Juan.** *Virtualization with Xen*. s.l. : Elsevier Science Ltd, 2007.
2. VIX API. *VMware*. [Online] <http://www.vmware.com/support/developer/vix-api/>.
3. **rPath.** *rPath*. [Online] <http://www.rpath.com/corp/>.
4. *CVS - Concurrent Versions System*. [Online] <http://www.nongnu.org/cvs/>.
5. Subversion. *Tigris.org*. [Online] <http://subversion.tigris.org/>.
6. The Athena Framework. *CERN*. [Online] <http://atlas-proj-computing-tdr.web.cern.ch/atlas-proj-computing-tdr/Html/Computing-TDR-21.htm>.
7. *Xen*. [Online] <http://xen.org/>.
8. XenApi. *Xen Wiki*. [Online] <http://wiki.xensource.com/xenwiki/XenApi>.
9. *User-mode Linux Kernel*. [Online] <http://user-mode-linux.sourceforge.net/>.
10. VMware Server. *VMware*. [Online] <http://www.vmware.com/products/server/>.
11. VMware Player. *VMware*. [Online] <http://www.vmware.com/products/player/>.
12. Parallels Workstation. *Parallels*. [Online] <http://www.parallels.com/en/products/workstation/>.
13. *VirtualBox*. [Online] <http://www.virtualbox.org/>.
14. **innotek GmbH.** *innotek VirtualBox User Manual*. Version 1.5.6. 2008. <http://www.virtualbox.org/download/UserManual.pdf>.
15. Microsoft Virtual PC. *Microsoft*. [Online] <http://www.microsoft.com/windows/products/winfamily/virtualpc/default.mspx>.
16. Microsoft Virtual Server. *Microsoft*. [Online] <http://www.microsoft.com/windowsserver/virtualserver/>.
17. *IEEE 803.2 Ethernet Working Group*. [Online] <http://www.ieee802.org/3/>.
18. cksum. *The Open Group*. [Online] <http://www.opengroup.org/onlinepubs/009695399/utilities/cksum.html>.
19. Hash function. *Wikipedia*. [Online] [http://en.wikipedia.org/wiki/Hash\\_\(computer\\_science\)](http://en.wikipedia.org/wiki/Hash_(computer_science)).
20. **Henson, Val.** An Analysis of Compare-by-hash. [Online] <http://www.valhenson.org/review/hash.pdf>.
21. **Harwood, Mike.** Storage Basics: Backup Strategies. *Enterprise Storage Forum*. [Online] September 24, 2003. [http://www.enterprisestorageforum.com/management/features/article.php/11186\\_3082691\\_1](http://www.enterprisestorageforum.com/management/features/article.php/11186_3082691_1).
22. **Lutz, Mark.** *Programming Python*. s.l. : O'Reilly, 1996.
23. *Python Programming Language - Official Website*. [Online] <http://python.org/>.
24. **Cavalli, Andrea and Poffet, Julien.** *SecurVMone - Project's Report*.
25. What Files Make Up a Virtual Machine? *VMware*. [Online] [http://www.vmware.com/support/ws5/doc/ws\\_learning\\_files\\_in\\_a\\_vm.html](http://www.vmware.com/support/ws5/doc/ws_learning_files_in_a_vm.html).
26. *rdiff-backup*. [Online] <http://www.gnu.org/savannah-checkouts/non-gnu/rdiff-backup/>.
27. *rsync*. [Online] <http://samba.anu.edu.au/rsync/>.



28. **rPath.** *Application to Appliance.*
29. **Buncic, Predrag.** *Portable Analysis Environment Using Virtualization Technology.*
30. *Filesystem in Userspace.* [Online] <http://fuse.sourceforge.net/>.
31. *Wikipedia.* [Online] [http://en.wikipedia.org/wiki/File:FUSE\\_structure.svg](http://en.wikipedia.org/wiki/File:FUSE_structure.svg).
32. *Parrot.* [Online] <http://www.cse.nd.edu/~ccl/software/parrot/>.
33. *Overview of Parrot.* [Online] <http://www.cse.nd.edu/~ccl/software/parrot/parrot-poster.jpg>.
34. *Chirp.* [Online] <http://www.cse.nd.edu/~ccl/software/chirp/>.
35. **Douglas Thain, Christopher Moretti and Jeffrey Hemmes.** *Chirp: A Practical Global Filesystem for Cluster and Grid Computing.* 2008.