INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Refinement Proposal of the Goldberg's Theory

Jérôme Gallard — Adrien Lèbre — Geoffroy Vallée — Christine Morin — Pascal Gallard — Stephen L. Scott

N° 6613

Juillet 2008

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--6613--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Refinement Proposal of the Goldberg's Theory *

Jérôme Gallard† , Adrien Lèbre† , Geoffroy Vallée‡ , Christine Morin† , Pascal Gallard§ , Stephen L. Scott‡

Thème NUM — Systèmes numériques
Équipe-Projet PARIS

Rapport de recherche n° 6613 — Juillet 2008 — 17 pages

Abstract: Virtual Machines (VM) allow the execution of various operating systems and provide several functionalities which are nowadays strongly appreciated by developers and administrators (isolation between applications, flexibility of resource management, and so on). As a direct consequence, "virtualization" has become a buzz word and a lot of "virtualization" solutions have been proposed, each providing particular functionalities. Goldberg proposed to classify virtualization techniques in two models (Type-I and Type-II), which does not enable the classification of latest "virtualizations" technologies such *abstraction*, *emulation*, *partitioning* and so on.

In this document, we propose an extension of the Goldberg model in order to take into account and formaly define latest "virtualization" mechanisms. After giving general definitions, we show how our proposal enables to rigorously formalize the following terms: *virtualization*, *emulation*, *abstraction*, *partitioning*, and *identity*. We also demonstrate that a single virtualization solution is generally composed by several layers of virtualization capabilities, depending on the granularity of the analysis.

Key-words: Goldberg theory, Virtualization, Emulation, Abstraction, Partionning, Identity

* The INRIA team carries out this research work in the framework of the XtreemOS project partially funded by the European Commission under contract #FP6-033576. ORNL's research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725.

† INRIA Rennes - Bretagne Atlantique, Rennes France – firstname.lastname@irisa.fr
‡ Oak Ridge National Laboratory, Oak Ridge, USA – valleegr, scottsl@ornl.gov
§ KERLABS, Rennes, France – pascal.gallard@kerlabs.com

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Proposition de Rafinement de la Théorie de Goldberg

Résumé : Les machines virtuelles (VM) permettent l'exécution de différents systèmes d'exploitation et fournissent plusieurs fonctionnalités qui sont aujourd'hui fortement apprécié par les développeurs et les administrateurs (isolement entre les applications, flexibilité de la gestion des ressources, etc). Ainsi, le mot "virtualisation" est devenu très prisé et beaucoup de solutions de virtualisation ont été proposées, chacune offrant des fonctionnalités particulières. Dans les années 1970, Goldberg a proposé de classer les techniques de virtualisation en deux modèles (Type-I et Type-II). Le problème est que cette classification ne permet pas de répertorier différentes techniques de virtualisation comme *l'abstraction*, *le partitionnement*, etc.

Dans ce document, nous proposons une extension du modèle de Goldberg, afin de prendre en compte et définir formellement les derniers mécanismes de virtualisation. Après donner des définitions générales, nous montrons comment notre proposition permet de formaliser rigoureusement les termes suivants: *virtualisation*, *émulation*, *abstraction*, *partitionnement*, et *identité*. Nous montrons également qu'une seule solution de virtualisation est généralement composée de plusieurs couches de capacités de virtualisation, en fonction de la granularité de l'analyse.

Mots-clés : Théorie de Goldberg, Virtualisation, Émulation, Abstraction, Partionnement, Identité

1 Introduction

Nowadays, the term "virtualization" is used to designate many solutions (such as abstraction, emulation, and partitioning) that do not necessarly have a clear formal definition. In 1970's, because virtualization was already a buzz word (used in several contexts with several definitions), Goldberg introduced an original definition of virtualization: *A system in which the virtual machine is a hardware-software duplicate of a real existing machine, in which a non-trivial subset of the virtual machine's instructions execute directly on the host machine in native mode* [5, 9, 1].

Initially, the main goal of virtualization was to enable time-sharing on big main-frames having a monotask operating system (OS). However, nowadays, with more and more performant hardware, virtualization is used for many different purposes such as isolation, server consolidation, and application portability. As a consequence, lots of virtualization technologies are developed and the latest techniques don't feet always in the Goldberg classification. For instance, containers, which allow processes to run concurrently on top of the same OS, based on their own view of available resources, can be considered in some extent as a virtualization mechanism which is not adressed by the Goldberg classification.

In this document, we propose an extension of the Goldberg model in order to take into account and formaly define latest "virtualization" mechanisms. After giving general definitions, we show how our proposal enables to rigorously formalize the following terms which are commonly used nowadays: *virtualization*, *emulation*, *abstraction*, *partitioning*, and *identity*. Doing so, we emphasis the fact that a single virtualization solution is generally composed by several layers of virtualization capabilities, depending on the granularity of the analysis.

This paper is not a negative criticism of the Goldberg theory and our rafinement is actually based on Goldberg's definitions. To our best-knowledge, no works have been done to give formal definitions of virtualization solutions and their functionalities. Effectively, works on this topic are generaly focusing on performance evaluation or on new capabilities.

The remainder of this paper is organized as follows: Section 2 introduces the context and defines common terms associated to virtualization. In addition, this section exposes Goldberg work. Section 3 proposes a refinement of the Goldberg's model. This refinement allows us to specify concepts such as *virtualization*, *emulation*, *abstraction*, *partitioning*, and *identity*. Section 4 shows in which way the presented refinement can be used for the classification of some "virtualization solutions". Section 5 exposes the analysis of typical existing systems with our refinement. Finally, Section 6 concludes.

2 Background

Virtualization solutions provide several capabilities which are nowadays strongly appreciated by developers and system administrators. Before to see the limitations of the Goldberg theory with regard to these latest solutions, we present his fundamental classification.

2.1 Goldberg Classification

In 1973, Goldberg proposed a formalization of the virtualization concept and described a classification based on two types: Type-I and Type-II [6]. His model relies on two functions, ϕ and f [6, 8]. The function ϕ makes the association between processes running on the VM and resources exposed within the VM; whereas the function f makes the association between resources allocated to a VM and the bare hardware. Functions ϕ and f are totaly independant, as ϕ is linked to processes in the VM, f is linked to resources.

Definition of the f function of Goldberg Let:

- $V = \{v_0, v_1, ..., v_m\}$ be the set of virtual resources.
- $R = \{r_0, r_1, ..., r_n\}$ be the set of resources present in the real hardware.

Goldberg defines $f : V \rightarrow R$ such that if $y \in V$ and $z \in R$ then

$$f(y) = z, \text{if } z \text{ is the physical resource for the virtual resource } y.$$

Definition of the recursion in the meaning of Goldberg Recursion could be reach interpreting V and R as two adjacent levels of virtual resources. Then, the real physical machine is level 0 and virtual resources is level n. As a consequence, f does the mapping between level n and level $n + 1$.

Recursion example If $f_1 : V_1 \rightarrow R$, $f_2 : V_2 \rightarrow V_1$, then, a level 2 virtual resource name y is mapped into $f_1(f_2(y))$ or $f_1 o f_2(y)$. Then, Goldberg generalized this case with n-recursion: $f_1 o f_2 o ... o f_n(y)$.

Definition of the ϕ function of Goldberg Let:

- $P = \{p_0, p_1, ..., p_j\}$ be the set of processes.

Goldberg defines $\phi : P \rightarrow R$ such that if $x \in P$, $y \in R$ then

$$\phi(x) = y, \text{if } y \text{ is the resource for the process } x.$$

Running a virtual machine: $f o \phi$ Running a process on a virtual machine means running a process on virtual resources. Thus, if processes $P = \{p_0, p_1, ..., p_j\}$ run on the virtual machine composed of virtual resources $V = \{v_0, v_1, ..., v_m\}$, then $\phi : P \rightarrow V$. The virtual resources, in turn, are mapped into their equivalents: $f o \phi : P \rightarrow R$.

General Virtual Machine From the previous statement, Goldberg defined the execution of a virtual machine: $f_1 o f_2 o ... o f_n o \phi$. Figure 1 depicts a simplified view of the Goldberg map composition [6].

From ϕ and f, Goldberg identified two different system virtualization types:

- *Type-I*: the general case of the definition of Goldberg,
- *Type-II*: the case where f does the mapping between resources of level $n + 1$ to processes of level n.

2.2 Goldberg Classification Limitations

Nowadays, in addition to the Goldberg Type-I and Type-II models, two new approaches have to be considered: *system-* and *process*-level virtualization.

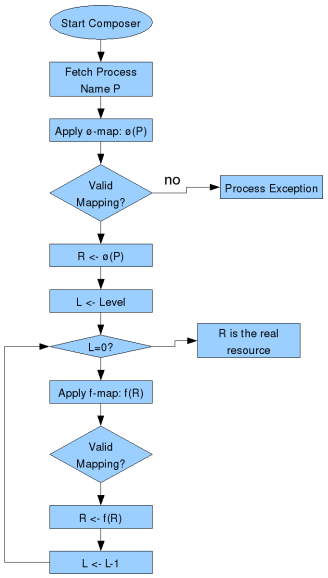

Figure 1: Simplified view of the Goldberg map composition

System-level virtualization (issued from the Goldberg definition) This approach aims at virtualizing a full OS: a virtual hardware is exposed to a full OS within a VM. The system running in a VM is named a *guest OS*. The VM cannot execute privileged instructions at the processor level. To access the physical devices, drivers are hosted in a privileged OS, called *host OS*. Moreover, virtual machines run concurrently and their execution is scheduled by a *hypervisor*. The hypervisor is also in charge of forwarding all privileged instructions from VMs to the host OS. Type-I virtualization is when the hypervisor is run directly upon the bare hardware, *e.g.,* Xen [2] whereas Type-II virtualization is when the hypervisor is ran on the host OS, *e.g.,* QEMU [3] and VMware Server [10].

Process-level virtualization It consists of running several processes concurrently on top of the same OS, each having its own view of available resources. OpenVZ [7], *chroot* [4], and *containers* capabilities provided by recent kernels are examples of process-level virtualization. Based on Goldberg terminology, it means the function ϕ realizes (this is done by the kernel of the host OS) the mapping of the virtualized process to the resources (virtual or not), and the function f is the standard mathematical function identity (the virtualized resources are already the same as the physical resources, and therefore the composition is already done).

The present work aims at refining the Goldberg model in order to include new virtualization technologies. Our refinement is based on the extension of two concepts introduced by Goldberg: *primitives constructs* and *derived constructs* [9]. Primitive constructs deal with hardware status whereas derived constructs extend the hardware by mapping processes. In other terms, primitive constructs deals with physical value of the hardware (for instance *2 GB* of space disk) whereas derived constructs deals with software intervention (for instance *partition ext3*). Primitive and derived constructs seem to be linked directly to the function f, however, to our best-knowledge, this never has been clarified and studied.

3 Refinement of the f Goldberg Function

In this section, we propose a refinement of the Goldberg theory which allows us to formaly define the terms *virtualization*, *emulation*, *abstraction*, *partitioning*, and *identity*.

Primitive constructs and derived constructs deal with two different aspects of resources, respectively their physical and logical characteristics. The goal of our proposal is to refine these two notions in order to improve the virtualization definitions. Therefore, in the rest of the document, we focus only on the resource set R, the virtual resource set V and the f function proposed by Goldberg.

3.1 Definitions

Definition 1 (Capacity, functionality, and status attributes) A resource (M), virtual or not, is characterized by a set of attributes.
Capacity attributes: the atomic attributes that define a resource (M).
Functionality attributes: the atomic operations provided by a resource (M).
Status attributes: the resource status that is exposed to the users.

Take the example of a hard disk (HDD). We could describe this HDD by: a capacity attribute: *10 GB* of disk space; functionality attributes: *read* and *write* operations; and a status attribute: *ext3* partition.

Definition 2 (Attribute Sets) We can also defines attribute sets:
Set of capacity attributes: $C = \{attributeC_1, attributeC_2, ..., attributeC_k\}$ is a set of capacity attributes. We note C_n the set of capacity attributes at level n, hence $C_n \subseteq C$.
Set of functionality attributes: $Q = \{attributeQ_1, attributeQ_2, ..., attributeQ_k\}$ is a set of functionality attributes. We note Q_n the set of functional attributes at level n, hence $Q_n \subseteq Q$.
Set of status attributes: $E = \{attributeE_1, attributeE_2, ..., attributeE_k\}$ is a set of status attributes. We note E_n the set of status attributes at level n, hence $E_n \subseteq E$.

With our example, we have $attributeC_1 = 10\ GB$, $attributeQ_1 = read$, $attributeQ_2 = write$, and $attributeE_1 = ext3$.

Definition 3 (Resource refinement) From the previous definitions, we refine the f function of Goldberg by giving three new functions to characterize a resource (M) (virtual or not):

- c, a function from a set of *capacity attributes* to another set of *capacity of attributes*, *i.e.*, $c_{n+1} : C_{n+1} \rightarrow C_n$.
- q, a function from a set of *functionality attributes* to another set of *functionality attributes*, *i.e.*, $q_{n+1} : Q_{n+1} \rightarrow Q_n$.
- e, a function from a set of *status attributes* to another set of *status attributes*, *i.e.*, $e_{n+1} : E_{n+1} \rightarrow E_n$.

Let's go back to our HDD example, let $hd \in R$ corresponding to our physical HDD resource with $f : hd_{n+1}(1\ GB, ext2, read, write) \rightarrow hd_n(10\ GB, ext3, read, write)$. This could be noted:

- $c_{n+1} : hd_{n+1}(1\ GB) \rightarrow hd_n(10\ GB)$,
- $q_{n+1} : hd_{n+1}(read, write) \rightarrow hd_n(read, write)$,
- $e_{n+1} : hd_{n+1}(ext2) \rightarrow hd_n(ext3)$.

It means that the HDD provides: (i) at level n, *10 GB* of disk space, and *ext3* file system, and *read* and *write* operations, and (ii) at level $n+1$, *1 GB* of space disk with a *ext2* file system with the same *read* and *write* operations. In that particular case, functionality attributes have not been modified.

In the rest of this document, we adopt the following notation:

Set / Level	$n+1$	n
Resource (virtual or not): $R \cup V$	$\{hd_{n+1}\}$	$\{hd_n\}$
Capacity: C	$\{1\ GB\}$	$\{10\ GB\}$
Functionality: Q	$\{read, write\}$	$\{read, write\}$
Satus: E	$\{ext2\}$	$\{ext3\}$

Definition 4 (Instructions) We define $instruction_n(fnct)$ a function giving the number of instructions necessary to execute the function $fnct$ at level n.

For instance, take the functionality "+": (i) in upper language level, we could use the operator + (*i.e.,* x + y), or (ii) in lower language level, we had to use operator *add, move* (*i.e., move x, move y, add*). In this example, one instruction is necessary with the upper language level whereas 3 instructions are necessary on the lower language level. Then, we could write: (i) $instruction_{n+1}(+) = 1$, and (ii) $instruction_n(+) = 3$.

3.2 Refinement Proposal

Notation used in this section:

- let set A and set B, we note $A = B$ if $A \cap B = \emptyset$
- let set A and set B, we note $A \neq B$ if $A \cap B \neq \emptyset$
- let AND and NOT the binary operator $\wedge$ and $\neg$.

Based on the previous definitions, we can designate (see Figure 2):

Definition 5 (Virtualization) $(Q_{n+1} = Q_n)\ AND\ (E_{n+1} = E_n) \Rightarrow Virtualization$ (this definition is from the Goldberg definition of virtualization).

Definition 6 (Identity) $(Virtualization)\ AND\ (C_{n+1} = C_n) \Rightarrow Identity$

Definition 7 (Partitioning) $(Virtualization)\ AND\ (C_{n+1} \neq C_n) \Rightarrow Partitioning$

Definition 8 (Emulation) $NOT(Virtualization) \Rightarrow Emulation$

Definition 9 (Simplicity) We define the simplicity by the comparison of the number of instructions needed to execute a functionality at a level n and $n+1$:

$$\begin{cases} simplicity(fnct) = 0,\ if\ instruction_{n+1}(fnct) \geq instruction_n(fnct) \\ simplicity(fnct) = 1,\ if\ instruction_{n+1}(fnct) < instruction_n(fnct) \end{cases}$$

Definition 10 (Abstraction)

$$(Emulation)\ \ AND\ \ (\forall fnct \in Q_{n+1},\ \ simplicity(fnct) = 1) \Rightarrow Abstraction$$

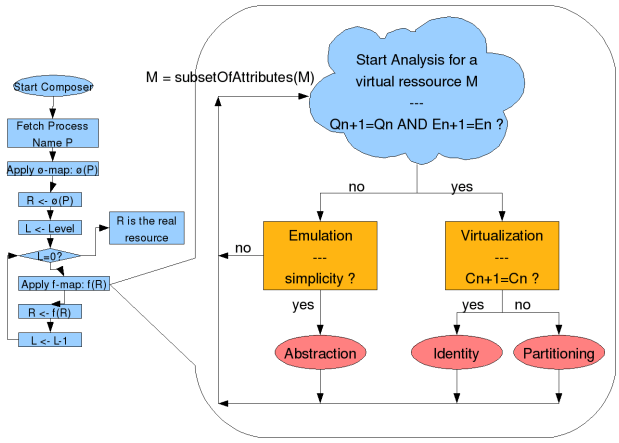

Figure 2: Representation of our refinement
Goldberg gives the definition of *virtualization* and *emulation*, with our refinement, we include the definition of *abstraction*, *partitioning* and *identity*.

We start the analysis with a virtual resource $M \in V$ composed of several attributes. According to the resource's attributes, a given system provides virtualization or emulation (*e.g.*, abstraction, partitioning, identity). Then, by recursivity, it is possible to take a subset of these attributes (*subsetOfAttributes*) and to start the analysis on this subset. Doing so, it is possible to refine the virtualization capabilities of such systems.

4 Refined Model Application

All examples we use are described with our refinement of the Goldberg theory. Doing so, we show clearly the meaning of *virtualization*, *emulation*, *abstraction*, *partitioning*, and *identity*. Our first example in Section 4.1 deals with *emulation* and *abstraction*. Then, in Sections 4.2 and 4.3, we address *identity* and *partitioning*. Section 4.4 deals with the problem of virtualization granularity.

4.1 Emulation and Abstraction

Based on our model, the following example shows emulation in a general way:

Set / Level	$n+1$	n
$R \cup V$	$\{object_{n+1}\}$	$\{object_n\}$
C	$\{attributeC_1\}$	$\{attributeC_A\}$
Q	$\{attributeQ_1\}$	$\{attributeQ_B, attributeQ_C\}$
E	$\{attributeE_1\}$	$\{attributeE_A\}$

Thus, we have two cases: *emulation* and *emulation-abstraction*.

Emulation (see Definition 8) adds functionalities to level $n+1$ that are not available at level n: $simplicity(attributeQ_1) = 0$.

Abstraction (see Definition 10) reduces the functional complexity exposed from a level n to a level $n+1$. Moreover, if $simplicity(attributeQ_1) = 1$, then this is *emulation-abstraction*. From the end-user point of view, it is easier to use the level $n+1$ rather than the level n. Functionalities at level $n+1$ are made by functionalities provided by level n. It is not possible, from level $n+1$, to

use directly any functionalities provided by level n. Abstraction is a particular case of emulation associated to the notion of simplification. The case of a calculator providing at level $n+1$ the functionality "+" and at level n the functionalities "move" and "add" illustrates the concept of abstraction:

Set / Level	$n+1$	n
$R \cup V$	$\{calculator_{n+1}\}$	$\{calculator_n\}$
C	$\{\}$	$\{\}$
Q	$\{+\}$	$\{add, move\}$
E	$\{\}$	$\{\}$

$simplicity(+) = 1$.

4.2 Virtualization - Identity

Based on our model, the following example shows virtualization-identity in a general way:

Set / Level	$n+1$	n
$R \cup V$	$\{object_{n+1}\}$	$\{object_n\}$
C	$\{attributeC_1\}$	$\{attributeC_1\}$
Q	$\{attributeQ_B\}$	$\{attributeQ_B\}$
E	$\{attributeE_A\}$	$\{attributeE_A\}$

Identity (see Definition 6) is when the action executed in a virtualized environment is the same than the one directly executed on the resources. With identity, the whole resource at level n is exposed to the upper level.

Now, we could instantiate this general example to a VM that is directly accessing the hard disk. We define hd_{n+1} the VM hard disk and hd_n the real hard disk:

Set / Level	$n+1$	n
$R \cup V$	$\{hd_{n+1}\}$	$\{hd_n\}$
C	$\{2\ GB\}$	$\{2\ GB\}$
Q	$\{powersafe\}$	$\{powersafe\}$
E	$\{ext2\}$	$\{ext2\}$

4.3 Virtualization - Partitioning

Based on our model, the following example shows virtualization-partitioning in a general way:

Set / Level	$n+1$	n
$R \cup V$	$\{object_{n+1}\}$	$\{object_n\}$
C	$\{attributeC_1\}$	$\{attributeC_A\}$
Q	$\{attributeQ_B\}$	$\{attributeQ_B\}$
E	$\{attributeE_A\}$	$\{attributeE_A\}$

Partitioning (see Definition 7) is the creation of separates sub-parts of a resource at level n, each part being exposed at level $n+1$. Moreover, each part is isolated from others by hardware or software mechanisms. For a given sub-part of a resource, partitioning allows identity.

For instance, if hd_{n+1} is the VM hard disk, and hd_n the physical hard disk, we have:

Set / Level	$n+1$	n
$R \cup V$	$\{hd_{n+1}\}$	$R_n = \{hd_n\}$
C	$\{2\ GB\}$	$C_n = \{10\ GB\}$
Q	$\{powersafe\}$	$Q_n = \{powersafe\}$
E	$\{ext2\}$	$E_n = \{ext2\}$

4.4 Emulation and Virtualization

The distinction between emulation and virtualization is difficult and actually depends on the granularity used to describe a system. For instance, with a calculator that provides at level n the operations "+, −, ∗, and $", and at level $n+1$ the operations "%, and +", we could write:

Set / Level	$n+1$	n
$R \cup V$	$\{calculator_{n+1}\}$	$\{calculator_n\}$
C	{}	{}
Q	$\{\%, +\}$	$\{+, -, *, /\}$
E	{}	{}

$simplicity(\%) = 1$, $simplicity(+) = 0$.

In this case, $Q_{n+1} \neq Q_n$, *i.e.,* this is *emulation* (see Definition 8). Intuitively, we understand that the function % is emulated by the functions +, ∗, and /. In addition, the simplicity of % is 1, therefore it is *abstraction*.

Moreover, the level $n+1$ provides the function +; this function is not emulated. In fact, from the + operation point of view, this is *virtualization - identity*.

This example shows that a system could be composed of emulation and virtualization according to the analysis granularity (see Figure 2): in the first analysis, we show that the system provides *emulation*, and in the second analysis, for a subset of functionalities, the system provides *identity*. Here, it is important to note that our refinement uses the same mechanism of recursion that Goldberg describes in his theory.

5 Use Cases

In this section, we analyse three common virtualization solutions with regard to our refinement.

5.1 Type-I Hypervisor

According to the Goldberg theory, with a Type-I virtualization, ϕ does the mapping between the processes at level $n+1$ and the resources at level n, whereas f does the mapping between the resources at level $n+1$ and the resources at level n. In fact, level 0 is the bare hardware, and level 1 is the virtual resources.

We define $op\_hardware_0 \in Q$ the set of functionalities provided by the bare hardware. We considere that the resources provided by the bare hardware are available 100% of the time. At level 1, we define $op\_hardware_1 \in Q$ the set of virtual functionalities provided by the Type-I hypervisor. In addition, each VM has y% of the time (the time of the physical CPU is shared between VMs). We could write:

Set / Level	1	0
$R \cup V$	$\{typeI\_hypervisor_1\}$	$\{typeI\_hypervisor_0\}$
C	$\{y\%time\}$	$\{100\%time\}$
Q	$\{op\_hardware_1\}$	$\{op\_hardware_0\}$
E	{}	{}

In addition, with Type-I hypervisor like Xen, $op\_hardware_1 = op\_hardware_0$. Then, we could say that, if $y < 100\%time$, then the Type-I hypervisor enables partitioning. However, if $y = 100\%time$ (only one VM is running), Type-I hypervisor enables identity.

If $op\_hardware_1 \neq op\_hardware_0$, then the Type-I hypervisor enables emulation. To our best-knowledge, no Type-I hypervisor enables emulation.

5.2 Type-II Hypervisor

According to the Goldberg theory, with a Type-II virtualization, ϕ does the mapping between the processes at level $n+1$ and the resources at level n, whereas f does the mapping between the resources at level $n+1$ to the processes at level n. In fact, the level 0 is the bare hardware, level 1 the host OS, and level 2 the virtual resources.

We define $op\_OS_1 \in Q$ the set of functionalities provided by the host OS. We consider that the host OS is available 100% of the time. We define $op\_hardware_2 \in Q$ the set of functionalities provided by the Type-II hypervisor. In addition, each VM has $y\%$ of the time (the time of the physical CPU is shared between VMs).

Set / Level	2	1
$R \cup V$	$\{typeII\_hypervisor_2\}$	$\{typeII\_hypervisor_1\}$
C	$\{y\%time\}$	$\{100\%time\}$
Q	$\{op\_hardware_2\}$	$\{op\_OS_1\}$
E	$\{\}$	$\{\}$

In addition, we could say that $op\_hardware_2 \neq op\_OS_1$ because they are semantically different ($op\_hardware_2$ provides low language level like *assembler* whereas op_OS_1 provides high language level). This is emulation (*e.g.,* QEMU without KQemu).

5.3 VMware Server or QEMU with KQEMU (Type-II)

VMware Server [10] and QEMU [3] are Type-II hypervisor. According to the Section 5.2 they provide emulation. However, if we focused on the CPU, VMware Server or QEMU with KQEMU provide OS (level-1) by-pass to exectute processor instructions.

Therefore, from the CPU point of view, we could represent them by: Let's take the example of a 3 GHz – 64 bits CPU at level 0, which is available 100% of the time, and which supports all i386 instructions ($i386\_func_0 \in Q$). From a CPU point of view, the only thing that changes at level 2 (because the level-1 is by-passed) is the percentage of time available for the VM (that is to say, for all available intructions at level 2, $i386\_func_2 \in Q$, we have $i386\_func_2 = i386\_func_0$). For instance, if a VM takes 30% of the CPU time, we have:

Set / Level	2	0
$R \cup V$	$\{cpu_2\}$	$\{cpu_0\}$
C	$\{3GHz, 30\%CPU\}$	$\{3GHz, 100\%CPU\}$
Q	$\{i386\_func_0\}$	$\{i386\_func_0\}$
E	$\{64bits\}$	$\{64bits\}$

According to Definition 7, this system provides partitioning. This results conforms with the common sense. In addition, this example confirms the fact that, how this kind of systems use directly the CPU, it is not possible to migrate this kind of VM from one CPU architecture to another.

These examples show that for a single resource and a single virtualization system, several kinds of "virtualization" techniques can be implemented, based on the analysis granularity.

5.4 Containers

Containers, such as OpenVZ [7], create isolated, secure "boxes" on a single physical server, enabling better server utilization and ensuring that applications do not conflict with each other. Each container performs and executes exactly like a stand-alone server; containers can be rebooted independently from the host OS. With this kind of system, ϕ makes the association between processes and resources provided by the containers; and, f makes the association be-

tween resources provided by containers and physical resources. Therefore, f is the *identity*; the resource provided in containers is the same as the real resource.

Analysis for the CPU point of view Let's take the example of a 3 GHz – 64 bits CPU at level 0, which is available 100% of the time, and which supports all i386 instructions ($i386\_func_0 \in Q$). From a CPU point of view, the only thing that changes at level 1 is the percentage of time available for the container (available intruction are $i386\_func_1 \in Q$). For instance, if a container takes 30% of the CPU time, we have:

Set / Level	1	0
$R \cup V$	$\{cpu_1\}$	$\{cpu_0\}$
C	$\{3GHz, 30\%CPU\}$	$\{3GHz, 100\%CPU\}$
Q	$\{i386\_func_1\}$	$\{i386\_func_0\}$
E	$\{64bits\}$	$\{64bits\}$

According to Definition 7, this system provides partitioning. This results conforms with the common sense. Now, we analyze our example based on two different granularities: (i) with time sharing, and (ii) without time sharing.

(i) CPU partitioning, analysis with time sharing.

Set / Level	1	0
$R \cup V$	$\{cpu_1\}$	$\{cpu_0\}$
C	$\{3GHz, 30\%CPU\}$	$\{3GHz, 100\%CPU\}$
Q	$\{i386\_func_1\}$	$\{i386\_func_0\}$
E	$\{\}$	$\{\}$

According to the Definition 7, this is partitioning.

(ii) CPU identity, analysis without time sharing.

Set / Level	1	0
$R \cup V$	$\{cpu_1\}$	$\{cpu_0\}$
C	$\{3GHz\}$	$\{3GHz\}$
Q	$\{i386\_func_1\}$	$\{i386\_func_0\}$
E	$\{64bits\}$	$\{64bits\}$

According to the Definition 6, this example is identity.

As already said, these other two examples show that for a single resource and a single virtualization system, several kinds of "virtualization" techniques can be implemented, based on the analysis granularity.

5.5 The Operating System Case

In this section we analyse an operating system with our theory, and we show that an OS is in some way a system of virtualization.

An OS is composed of two importants parts: (i) the kernel who makes the link with the bare hardware and (ii) the libraries who make the link between the applications and the kernel. With this decomposition, it is possible to say that the f function of Goldberg makes the mapping between the kernel and the bare hardware whereas the ϕ function makes the mapping between the applications and the kernel. Then, we define the level 0 as the level (bare hardware) providing *binary_ operations* and the level 1, as the level (kernel) providing *human_ usable_ fnct*. Figure 3 presents the different components of an OS.

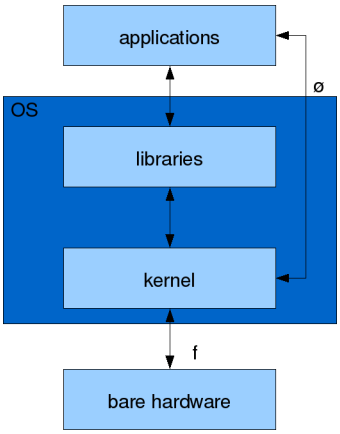

Figure 3: Different components of an OS.

Refinement of the f function for an OS. We define $human\_usable\_fnct$ the set of functionalities usefull for the human being. Then, we define $binary\_operations$ the set of functionalities available on the bare hardware. It is obvious that, for a majority of human being, the use of functionalities usable is more easy that the use of binary operations. This is why we could write:
$\forall x \in human\_usable\_fnct, simplicity(x) = 1$.

Now we propose our refinement:
$OS_1(human\_usable\_fnct) \rightarrow OS_0(binary\_operations)$.

Set / Level	1	0
$R \cup V$	$\{OS_1\}$	$\{OS_0\}$
C	{}	{}
Q	$\{human\_usable\_fnct\}$	$\{binary\_operations\}$
E	{}	{}

We could say that $human\_usable\_fnct \neq binary\_operations$ because they are semantically different ($binary\_operations$ provides low language *assembler* whereas $human\_usable\_fnct$ provides high level language). With this refinement we could say that an OS is an abstraction (definition 10) of the bare hardware.

Refinement of the f function for the virtual memory. The principle is the following: at level 0, the bare hardware provides several kinds of memory (RAM: op_RAM, hardisk: op_HDD, flash: op_flash) and, at level 1, the OS gives to the applications an uniform way to access to those memories. Like the last paragraph, we define $h\_usable\_fnct\_mem$ the set of usefull functionalities to manipulate the memory for the human being. It is obvious that, $\forall x \in h\_usable\_fnct\_mem, simplicity(x) = 1$.

We could propose the following refinement:

$$OSmem_1(h\_usable\_fnct\_mem) \rightarrow OSmem_0(op\_RAM, op\_HDD, op\_flash).$$

Set / Level	1	0
$R \cup V$	$\{OSmem_1\}$	$\{OSmem_0\}$
C	{}	{}
Q	$\{h\_usable\_fnct\_mem\}$	$\{op\_RAM, op\_HDD, op\_flash\}$
E	{}	{}

With this refinement we could say that an OS is an abstraction (definition 10) of the memory of the bare hardware

Refinement of the f function for processes. We take the case of processes. We assume that, in our example, from an OS point of view, a process need only: memory and CPU time. On this example we assume that, at level 0, the bare hardware provides 15% of RAM available and 100% of flash memory available. In addition, the bare hardware provide 100% of CPU time available. To manage these resources, the bare hardware provides three set of tools: *op_RAM*, *op_flash*, and *op_CPU* whiches are operations to control RAM, flash memory and CPU. Then, at level 1 we considere that the process has only 10% of CPU time, and he gets 15% of the available memory of the system. *op_mem* and *op_CPU* are operations available to manage the memory and the CPU.

In this way we could write:

$$OSproc_1(10\%CPU, 15\%MEM, op\_mem, op\_CPU) \rightarrow$$
$$OSproc_0(100\%CPU, 100\%flash, 15\%RAM, op\_RAM, op\_flash, op\_CPU).$$

Set / Level	1	0
$R \cup V$	$\{OSproc_1\}$	$\{OSproc_0\}$
C	$\{10\%CPU, 15\%MEM\}$	$\{100\%CPU, 100\%flash, 15\%RAM\}$
Q	$\{op\_mem, op\_CPU\}$	$\{op\_RAM, op\_flash, op\_CPU\}$
E	{}	{}

With this example, applying our refinement, we obtain that this is emulation (definition 8). Then, with our refinement, it is possible to say that, a OS gives to processes an emulation of the bare hardware.

Focuse on the memory However, if we change the granularity of our study and if we take just management of the memory, then we have:

$$OSproc_1(15\%MEM, op\_mem) \rightarrow$$
$$OSproc_0(100\%flash, 15\%RAM, op\_RAM, op\_flash).$$

Set / Level	1	0
$R \cup V$	$\{OSproc_1\}$	$\{OSproc_0\}$
C	$\{15\%MEM\}$	$\{100\%flash, 15\%RAM\}$
Q	$\{op\_mem\}$	$\{op\_RAM, op\_flash\}$
E	{}	{}

Here, obviously we have, $simplicity(op\_mem) = 1$. With this example, applying our refinement, we obtain that, an OS gives abstraction of the memory to processes (definition 10).

Focuse on the CPU Hence, if we change the granularity of our study and if we only take the management of the CPU, we obtain:
$OSproc_1(10\%CPU, op\_CPU) \rightarrow OSproc_0(100\%CPU, op\_CPU)$.

Set / Level	1	0
$R \cup V$	$\{OSproc_1\}$	$\{OSproc_0\}$
C	$\{10\%CPU\}$	$\{100\%CPU\}$
Q	$\{op\_CPU\}$	$\{op\_CPU\}$
E	$\{\}$	$\{\}$

With this example, applying our refinement, we obtain that, this is an OS gives partitioning of the CPU to processes (definition 7).

To conclude, we show on the example of the OS, that for processes an OS give several layer of virtualization according to the granularity of the study. If we considere that a process use only memory and CPU resources, we could say that a OS gives to processes an emulation of the bare hardware. However, if we affine the study, we could say that from a memory point of view, a OS gives abstraction to processes whereas it gives partitioning from a CPU point of view.

5.6 Java Virtual Machine – JVM

Java is portable oriented object language, that is to say, one Java code could be run on any architectures if this architecture dispose of the good environement. This environement is named Java Virtual Machine (JVM). The principle is the following: the Java code is transformed in bytecode by a Java compilator. Then this bytecode is executed by the JVM on an architecture. Each architecture disposed of its own JVM.

From an OS point of view, a JVM is like a process. Here, we could write: ϕ makes the mapping between the process of the JVM and the resources required by the JVM and f makes the mapping between resources required by the JVM (level 2) and the OS (level 1).

The OS provides operations to manage resources, *op_ OS* and the JVM provides to the *bytecode* the good interface (*op_ bytecode*) to exectute it. Like this we could write: $JVM_2(op\_bytecode) \rightarrow JVM_1(op\_OS)$.

Set / Level	2	1
$R \cup V$	$\{JVM_2\}$	$\{JVM_1\}$
C	$\{\}$	$\{\}$
Q	$\{op\_bytecode\}$	$\{op\_OS\}$
E	$\{\}$	$\{\}$

In addition, if $simplicity(op\_bytecode) = 1$ then, the JVM is an abstraction of the hardware for the byte code, otherwise, it is just an emulation of the hardware.

6 Conclusion

Goldberg defines virtualization with two functions ϕ and f: ϕ does the mapping between virtualized processes and resources (virtualized ot not), whereas, f does the mapping between virtualized resources and real resources. Based on these two functions, Goldberg defines two kinds of system-level virtualization: *Type-I* and *Type-II*. However, we show that some process-level virtualizations, such as containers, do not perfectly fit with the Goldberg classification.

In this document, we propose a refinement of the Goldberg functions, based on the primitive and derived constructs proposed by Goldberg. This allows us to specify systems that do not belong to Type-I or Type-II systems. In addition, we have extended the formal Goldberg definition for *virtualization* and *emulation* in order to introduce the *abstraction*, *partitioning*, and *identity* concepts. Doing so, we emphasis the fact that, even with a single virtualization solution (for instance containers), the virtualization capabilites may differ, depending on the virtualization granularity. In other words, one complex virtualization system could integrate, according to

the granularity of the analysis, several virtualization capabilities. We presented how different analysis granularities can be applied to containers.

Our refinement of the Goldberg model allows us to strictly classify available virtualization solutions.

We think that our model can be extended to systems such as Java Virtual Machines and operating systems. In other terms, it could be interesting to see if a JVM, an OS, or any computing system, can be analyzed with the Goldberg theory and our refinement.

References

[1] G. M. Amdahl, G. A. Blaauw, and Jr F.P. Brooks. Architecture of the ibm system/360. IBM J. RES. DEVELOP. VOL.44 NO 1/2, 1964.

[2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. Bolton Landing, New York, USA, October 2003. SOSP'03.

[3] Fabrice Bellard. Qemu, a fast and portable dynamic translator. Technical report, USENIX Association, 2005.

[4] GNU. Chroot, 2007. Available at http://www.gnu.org/software/coreutils/manual/coreutils.html#chroot-invocation.

[5] R. P. Goldberg. Virtual machines: semantics and examples. Proceedings IEEE International Computer Society, Conference Boston Massachusetts, 1971.

[6] R. P. Goldberg. Architecture of virtual machines. AFIPS National Computer Conference, July 1973.

[7] OpenVZ. Openvz welcome page, 2007. Available at http://wiki.openvz.org/Main_Page.

[8] Gerald J. Popek and R. P. Goldberg. Formal requirements for virtulizable third generation architectures. July 1974.

[9] R. P. Goldberg U. O. Gagliardi. Virtualizeable architectures. Proceedings ACM AICA International Computing Symposium Venice Italy, 1972.

[10] VMware. Vmware server, 2007. Available at http://www.vmware.com/products/server/.

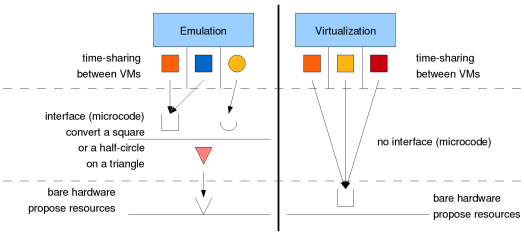

Figure 4: Virtualization vs Emulation

Appendix

A Virtualization vs. Emulation

First of all, we have to make a clear distinction between *virtualization* and *emulation*. Nowadays, we could say that *virtualization* allows to divide/share resources of a computer for several users by applying techniques such as time sharing, virtual memory and so on.

The Goldberg definition agrees with this general statement [5, 9]. Moreover, Goldberg precises the fundamental difference between virtualization and emulation: virtualization is used when a non-protected part of the virtualized code is executed directly on the bare hardware, whereas emulation is used when a protected or non-protected part of the code uses a special microcode (interface) that is not a physical part of the host machine (see Figure 4).

B General definitions

Now, we gives general definitions of *abstraction*, *aggregation*, *partitioning* and *identity*.

- *Abstraction* reduces the functional complexity exposed by a given system. Abstraction is a concept not associated with any specific instance. For instance, at higher level language, the "+" operator is an abstraction of "move" and "add" of lower level language. Abstraction is linked with logical tools.
- *Aggregation* reduces the physical complexity exposed by a given system. This allow to see several "little" object as one "big". For instance, memory from several RAM modules are aggregated (associated) into one big memory. Aggregation is linked with physical tools.
- *Partitioning* divides a resource: (i) physical, or (ii) logical. For instance, it is possible to partition an 20 GB hardisk in two 10 GB partitions (i). In addition, it is possible to partition a processor according to the time for the execution of several processes (ii).
- *Identity* enables the direct use of the native resource.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399