A Reconfiguration Language for Virtualized Grid Infrastructures

Rémy Pottier, Marc Léger, Jean-Marc Menaud

A Reconfiguration Language for Virtualized Grid Infrastructures*

Rémy Pottier, Marc Léger, and Jean-Marc Menaud

Ascola (EMN/INRIA, LINA)
Ecole des Mines de Nantes
4, rue Alfred Kastler
44307 Nantes, France
first-name.last-name@emn.fr

Abstract. The growing needs in computational power to answer to the increasing number of on-line services and the complexity of applications makes it mandatory to build corresponding hardware infrastructures and to share several distributed hardware and software resources thanks to grid computing. To help with optimizing resource utilization, system virtualization is a more and more adopted technique in data centers. However, this software layer adds to the administration complexity of servers and it requires specific management tools to deal with hypervisor functionalities like live migration. To address this problem, we propose VMScript, a domain specific language for administration of virtualized grid infrastructures. This language relies on set manipulation and is used to introspect physical and virtual grid architectures thanks to query expressions and notably to modify VM placement on machines.

1 Introduction

Data centers are one of more important Internet component (with the access point and network). These infrastructures are used most of the time to host on-line services. Traditional datacenters typically host a large number of relatively small-sized applications, and host hardware and software for multiple organizational units or even different companies when traditional cluster belongs to a single organization, with a relatively homogeneous hardware and system software platform, and share a common systems management layer. So from an architecture perspective, datacenter is closer to a grid architecture (which is a clusters federation) than to a single cluster. That is why, and like grid, the physical administration of this infrastructure is a real challenge, both in monitoring and in its administrative base operations (shutdown, reboot, etc.).

From a software perspective, virtualization [8] has spread in datacenter. It allows to gain efficiency for resource utilization and flexibility for application execution. In this approach, each small-sized application hosted is running in a

* This work is partially funded by the SelfXL ANR/ARPEGE project (http://selfxl.gforge.inria.fr/dokuwiki/doku.php).

virtual machine. Virtual machines (VMs) can be thus used to consolidate the workloads of under-utilized servers to use fewer physical machines so as to save on hardware and power consumption [10]. However virtualization as a new abstraction layer adds complexity for data center administrators. If administrators agree on the benefits of virtualization to reduce costs and improve flexibility, most also recognize that it makes the administration more complex and error prone. That is why many companies think about adapting their management tools and instrumentation to current needs of virtualized environments. One of the new issues raised by administrators is the fine management of VMs.

Administrators want more particularly to express complex queries on resources (introspection) and manipulate elements (intercession). Low levels APIs (e.g. Xen API [4]) provide some primitive operations on VMs with for example instantiation, shutdown, static or live migration, etc. but no complex operations on sets of elements. Manipulation of collections of resources is then done by invoking these APIs in some general purpose or scripting languages which are not necessarily adapted in terms of concision and precision of their syntax.

Our proposition relies on the use of several domain specific languages (DSLs) for grid administration. First of all, we need to define a model for describing grid resources. In our model, a grid user defines a task in the form of a set of VMs called a virtual job which can execute on the grid. A grid is then modeled as a graph of physical elements like machines, racks or clusters, and logical elements including VMs, virtual jobs, and users. We define description languages for administrators to represent grid physical architectures, virtual organizations as sets of users. Another language allows user to describe virtual jobs they want to submit to the grid. In addition to these description languages, another DSL is used to manage the resources previously describe (i.e. navigate in the grid physical and logical architecture to select elements with given properties). This last language is also used to execute reconfiguration operations on grid elements like VM migrations [5].

This paper is organized as follows. Section 2 presents some related work about languages for grid management. Section 3 presents our model to represent resources in a grid architecture and some description languages to build these architectures and virtual jobs. Section 4 describes a domain specific language for the grid management based on selections, navigations and dynamic reconfigurations in grid architectures. Our DSL approach for grid administration is evaluated in Section 5 before concluding in Section 6.

2 Related Work

We are considering two categories of work related to grid administration and DSLs, those involving basic operations and those on language aspects.

Basic operations: Shells and APIs. Basic management operations are performed by using hypervisor APIs which allow to manipulate VMs (instantiation, migration, destruction, etc.). As each hypervisor (Xen [4], KVM [11], etc.) has its own

API, the libvirt API [1] may be used to manage different virtualization solutions through a common interface. Above these APIs, shells unify the most common management operations and administrators can similarly manage a server whatever the hypervisor is. Shells (e.g. Usher [13]) and API approaches are designed for local management on a given server. For good working order, administrators need to have information about the whole grid to manage grid resources. To fill this gap, some virtual machine managers [15] (e.g. Virtual Machine Manager [2]) offer an overview of the grid with real time monitoring. These tools help administrators to manage all grid elements with a common interface, sometimes a graphical user interface, whatever the hypervisor is. However, they offer only limited operations in terms of resource queries and complex reconfiguration operations.

Domain Specific Languages Language approaches address description and reservation in grid context. A grid description may be carved up into grid resources description and description of how to use these resources (that is to say job descriptions). The Job Submission Description Language (JSDL) [3] is a XML based language to describe a job and its needs (resources and applications). For a grid architecture description, VXDL [9] is a language for virtual resources interconnection networks specification and modeling. It describes virtual infrastructures, especially virtual network, and queries the model about the network topology. Other specific languages have proposed to permit users to get resources and use them. ClassAd [14] and xRSL [2] are declarative languages with attribute-value pairs. A language keyword identifies properties on which users can make a selection. Users describe resources required (network, disk, memory, etc.) and how to use them. SWORD [1] is a framework which collects grid monitoring information into a database and provides a query language for selecting and ranking required resources. These languages address grid resources utilization and grid description but not administrative tasks.

Our aim is to overcome the limitations of these tools and languages by proposing an approach based on several domain specific languages for both describing and managing (observation and reconfiguration) resources in grids.

3 Specification of Grid Architectures

We distinguish two kinds of actors: administrators and users. Administrators configure servers, networks and, with virtualization, define virtual machines placement. Users submit their jobs and manage them without explicitly choosing specific servers. Each user describes the resources necessary for his job. In our case, grid resources are modeled by virtual machine requirements like in [7]. A job is composed of a set of virtual machines which will be executed on the infrastructure, it is then called a virtual job or *vjob* (called *lease* in other works [16]. Each user belongs to a virtual organization (VO).

[1] http://libvirt.org/html/libvirt-libvirt.html
[2] http://virt-manager.et.redhat.com/

3.1 Life Cycle in Grid Management

Figure 1 describes the classical lifecycle in a grid with the user vjob submission and adminstrator's maintenance operation. First, after the user has specified his vjob, he submits it to the *vjob configuration parser*. This parser builds a vjob with the appropriate number of virtual machines. Then, this vjob is submitted to the *management system*. If the submission succeeds, all virtual machines of the vjob are placed in the grid.

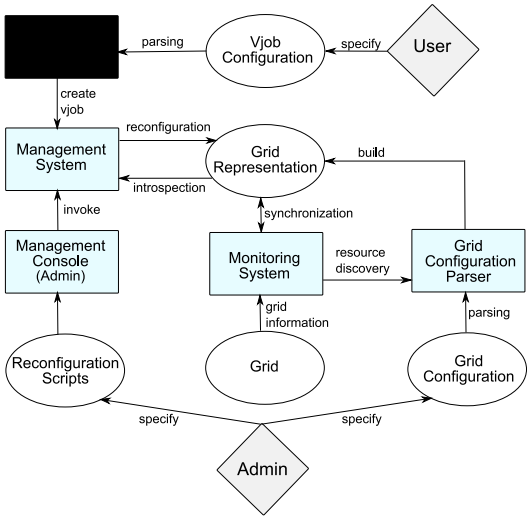

Fig. 1. Global architecture of the grid framework

The management system is used to build a grid representation, to modify it, or to query it. It knows all elements of the *grid representation* and it allows to ensure some good properties such as uniqueness (e.g., a unique IP per machine). The management system also checks that grid elements are correctly and completely configured. For example, the virtual machine memory is essential to place a VM on a server. Moreover the management system is able to place VMs in the grid with respect to their needs. If no placement is found for a VM, it is rejected.

The grid representation is built from a full description of the grid supplied by an administrator. This representation is a structural model of the grid resources (discussed in Section 3.2) and it is tied to the real grid by a monitoring system (in our case Ganglia [12]). This one checks grid representation information to ensure consistency between the real grid and its representation. This causal connection is only maintained for servers and VMs in our framework because the monitoring system does not give information about other elements, like the cluster organization. So if an unexpected event happens, for example an element disappears, the monitoring system detects it and the grid representation is updated accordingly.

Several verifications are performed on this description by the *grid configuration parser* in collaboration with monitoring system and management system.

In the first place, this parser checks the structure of the grid by comparison between the description and the grid model. In the second place, information from grid description are compared to monitoring system information. When a representation is built, the administrator can perform grid reconfigurations, like adding a server, in the management console. He may write a sequence of operations in a script executed in a *management console*. For example, migrate a virtual machine, then shut down a server.

3.2 A Grid Model

A model of grids (Figure 2) has been conceived as a multi-graph with labeled nodes and arcs. In this graph, the nodes correspond to grid elements with properties and operations. The arcs represent relations between these elements. This graph is navigable with bidirectional relations. This model is one particular view of what a grid is, but it can be adjusted for describing other kinds of grid organizations. It is composed of two kinds of elements: physical elements and logical elements.

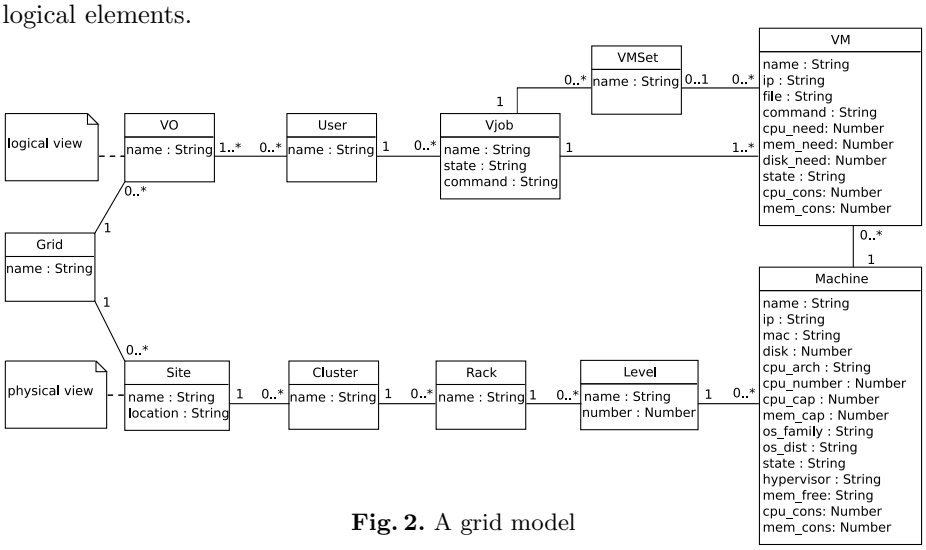

Fig. 2. A grid model

A physical element is basically a server container. The smaller the container is, the more accurate the location information of the server is. Servers are identified by the more generic term *machine*. Some node properties are mandatory and must be initialized to enable grid management. Optional properties, like operating system, allow administrator to simplify the grid management. A machine is placed into a rack at a specific level. A set of racks composes a cluster which itself belongs to a site. A site is a general term to design a set of clusters. A site can represent either simply a room where servers are located, or a city with several data centers. The main element of the logical view is the virtual machine (VM). In a vjob defined by several VMs, we can create special groups of VMs called *VMSet*. For instance, a VMset can be used to group all VMs

containing server for a given tier in a 3-tiers application. Each vjob is linked to its owner represented by a User element. A virtual organization (VO) is a set of users who can connect to the grid. Physical and logical views are linked by the hosting relation between machines and VMs.

An important property is the life-cycle state of machines, VMs and vjobs. These states represent the current element life cycle and allow to restrict the execution of some operations. The machine life cycle is a trivial two-state automaton with *on* and *off* states. The VM life cycle consists of five states: *uninitialized* before some mandatory properties are configured, *initialized*, *started*, *suspended* and *stopped*. As we only consider live migration operations on VMs, the VM state remains *running* during migration. A vjob is a composition of VMs, so a vjob has the same life cycle as a VM.

Several description languages are provided to specify grid architectures. These languages are based on XML and XML Schema, so that they conform to our grid model and its mandatory properties. The first language is used by administrators to describe the physical architecture of grids. This description is used by the grid configuration parser to build the physical representation of the grid.

```
<grid name="pastel">
 <site name="EMN" city="Nantes">
  <cluster name="Xen">
   <rack name="ra1">
    <level number="2">
     <machine hostname="pastel-1.b217.home" ip="192.168.0.107"
      mac="00:21:70:25:55:b0">
      <cpu arch="64" number="2" capacity="2000" />
      <memory capacity="4000" />
      <disk capacity="150" />
      <os family="Unix" distribution="Ubuntu" />
     </machine>
    </level>
    <level number="3">
     <machine hostname="pastel-2.b217.home" ip="dhcp"
     mac="00:21:70:25:55:b1">
      ...
```

Example of a partial description of a physical grid architecture

As different actors handle the logical view differently, there are two languages to describe it. The first one allows an administrator to link users with a grid by specifying VOs. The second one is used by grid users to define their vjobs to be submitted to the grid.

```
<vjob name="myjob" period="24h" command="run.sh">
 <vm hostname="myvm" ip="192.168.0.2"  file="/farm/LennyApp/myvm.cfg">
  <need cpu="2000" memory="3000" disk="500" />
 </vm>
 <vmset name="db">
  <vm hostname="dataBase1" file="/test.net/mysql.cfg">
   <need memory="256"/>
  </vm>
  <vm hostname="dataBase2" file="/test.net/mysql.cfg">
   <need memory="256"/>
  </vm>
 </vmset>
</vjob>
```

Example of a vjob description

4 A Domain Specific Language for Grid Management

Once our grid model has been defined, administrators and users manage resources by navigating and selecting elements in grid physical and logical architectures and by dynamically reconfiguring these architectures. This section describes VMScript, a domain specific language for grid management, i.e. introspection and intercession in grid elements. This language is inspired by previous work on a reconfiguration language in component-based architectures called FScript [6]. Actually, our language is divided into two parts for respectively introspection and reconfiguration. The introspection language, named VMPath, is used to express queries in grid architectures. The reconfiguration language, VMScript, allows the execution of dynamic reconfiguration operations on grids and is a super set of VMPath.

4.1 Selection and Navigation in Grid Architectures

A grid configuration (or architecture) is defined as a labeled directed multigraph. To query these architectures, the VMPath language is used as a side-effect free declarative language. It is restricted to the navigation in grid architectures, the selection of grid elements by their location or their properties. Therefore, the execution of a VMPath expression cannot lead to modifications in grids.

VMPath syntax. The language has a very concise but powerful syntax based on XPath 1.0 [17], the W3C standard query language for XML documents. Several arguments are in favor of this choice:

- XPath does not depend on the specific syntax of XML documents, it can be used on abstract graph models such as our grid model. Actually XPath only defines concepts of nodes, properties and relations between nodes.
- The syntax is open and flexible. Although XPath specifies a fixed set of nodes and relations (XPath axes) to query XML documents, it is possible to define new types of nodes and relations. Our grid model does not use XPath base XML axes (child, attribute, etc.) but defines its own navigation axes.
- The syntax is concise and readable, an XPath allows to express one-line queries. Moreover, XPath defines a node-set data type which allows powerful set queries with set operations.

Despite all these advantages, VMPath does not rely on existing XPath implementations because these implementations are too tied to XML representations.

The generic syntax of a VMPath expression consists of a sequence of steps separated by slashes (cf. Figure 3). A step is composed of an axis specifier which indicates the arc to follow in the graph for navigation, and a set of optional predicates to filter the selected nodes. There is no intrinsic notion of hierarchy in navigation and so a navigation axis does not necessarily represent a hierarchical relation between elements. The beginning of the expression, `$grid`, refers to the initial node set used in the query. This node set is stored in a VMPath

variable and denotes in this case a grid element. The navigation axis used in the expression is the site axis which basically selects all site nodes belonging to the grid. This set of sites is then filtered thanks to a predicate to select only sites which are named '*Paris*', i.e. all sites which are localized in Paris. The '@' symbol is used to query the value of the name property.

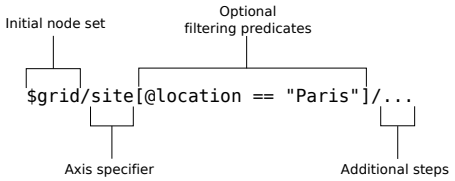

Fig. 3. Syntax of query expressions

VMPath is a dynamically typed language (type checking is performed at runtime). The four primitive data types defined are the same as in XPath 1.0: *node-set*, *string*, *number*, and *boolean*. As there is no notion of attribute nodes, a special type *multi-set* has been added to deal with multi-sets of primitive types. The VMPath language supports the classic arithmetic, boolean and comparison operators and also set operators (union, intersection and difference).

Functions in VMPath are side-effect free procedures. A library of predefined functions is provided with the language. These functions are essentially:

- property accessors to get values of node properties (e.g., '*name*()' to get the name of a node). The '@' notation before a property name is strictly equivalent to the accessor function on the property (e.g., '@*name*'). It should be noted that these functions can be applied to a set of elements. For instance '*name*($*set*)' would return a multiset of strings corresponding to all the names of the elements contained in the set '$*set*'.
- functions for string manipulation (e.g., '*concat*()' for string concatenation, '*match*()' to test the matching of a string and a regular expression)
- aggregation functions on element collections: '*size*()' (returns the cardinality of a set), '*sum*()' (returns the sum of a number set), etc.

VMPath examples. VMPath can be used to express a wide range of queries on grid architectures. Some examples are presented afterwards.

A selection of all racks in a grid is performed thanks to the following expression:

```
$grid/site/cluster/rack
```

A shortcut navigation axis is usable when there is no ambiguous path in the graph to reach the wanted nodes. For instance, the previous expression using a shortcut axis could be expressed as follows:

```
$grid//rack
```

Grid elements can be selected by the value of properties. For example, we may want to find the rack which contains a machine with a specific IP address in a cluster:

```
$cluster//machine[@ip == '192.168.110.36']//rack
```

4.2 Dynamic Reconfiguration of Grids

The VMPath query language is integrated into another DSL focusing on the dynamic reconfiguration of grids, VMScript. VMPath expressions are used to select the grid elements to reconfigure. VMScript is an imperative language providing procedures and control structures so as to program reconfiguration scripts of grids.

Procedures. VMScript makes the distinction between two kinds of procedures: functions and actions. Functions are side-effect free procedures only for grid introspection, whereas actions are intercession procedures to actually modify grid configurations. A primitive action in our model is a primitive graph transformation in a grid representation such as listed below:

- Addition or removal of a node. For instance, to add or remove a cluster node in the graph, we could use respectively the following procedures:

  ```
  new-cluster();
  delete-cluster($cluster);
  ```

- Addition or removal of a relation between nodes. For instance, these two procedures respectively add and remove a *rack* relation between a cluster and a rack, i.e. add and remove a rack in a cluster:

  ```
  add-rack($cluster, $rack);
  remove-rack($cluster, $rack);
  ```

- Modification of the value of a node property. To change the name of a grid, the following setter is applied:

  ```
  set-name($grid, "mygrid");
  ```

All these primitive actions are automatically generated from the description of our grid model so that possible modifications in the model are transparently taken into account. Native procedures like primitive actions are implemented in Java, the implementation language of the VMScript interpreter. However it is possible to define procedures directly in the VMScript language. These user-defined procedures can be loaded at any time in the interpreter. A VMScript procedure is specified by means of the *function* or *action* keywords:

```
function isEmpty(set) {
  return size($set)==0;
}
```

```
action migrate-vjob(vjob, dest) {
  for $vm: $vjob/vm {
    migrate ($vm, $dest);
  }
}
```

The first procedure is a function which returns true if a set is empty, false otherwise. The second procedure is an action which takes a virtual job and a machine in argument. It consolidates the vjob on the same destination machine by migrating all of its VMs. These two procedures are part of a standard library. This library contains utility procedures which are loaded when the interpreter is instantiated.

Control structures. VMScript supports classic control structures in addition to the sequencing of instructions.

New variables can be created by assigning them an initial value. Variables are mutable and their scope is defined by the block where they are declared. In the following example, '*grid*' is a global variable since it is defined outside any block. It is initialized with a grid node built from the configuration file *mygrid.xml* describing a grid architecture. A declared variable is then referenced by means of the '$' symbol.

```
grid = adlnew-grid(``mygrid.xml");
echo($grid);
```

The conditional execution if-then-else uses the standard C syntax. The following example tests if the memory capacity of a machine is above a threshold. If the test evaluates to true, it adds a VM to the machine, otherwise it prints a message to the standard output.

```
if($machine@mem_cap >= 2000) {
  add($vm,$machine);
} else {
  echo(``Not enough memory.");
}
```

Iteration is restricted to finite sets with a *for* loop. This limitation prevents from programming infinite loops and non-terminating scripts. The execution semantics of an iteration is to sequentially iterate on every element in the set. For example, the following code iterates on every machine in the grid which do not host any VM.

```
for $m: $grid//machine[size(./vm)==0] {
  shutdown($m);
}
```

Some native actions are defined to take either a single value or a set in argument. In the latter case, the primitive action is executed in parallel on each element of the set. For instance, the previous example could be executed in parallel with the same action but without explicit iteration:

```
shutdown($grid//machine[size(./vm)==0]);
```

An explicit *return* statement allows the stopping of a program execution and returns to the caller with possibly sending a value.

```
function cpu_cons_average(machines) {
  return average(machines@cpu_cons);
}
```

Execution model. Primitive actions defined in VMScript are directly mapped on operations in the grid model. For instance, a `migrate-vm` action on a VM node corresponds to a `migrate` Java method on a VM object from our model API. As previously mentioned and thanks to the causal connection between the grid and its representation, the execution of an operation in the model comes to execute operations on real machines and VMs through SSH and calls to native APIs (e.g., Xen API for Xen VMs). VMscript code is executed in an interpreter programmed in Java which can be embedded in applications. Furthermore, an interactive console is provided so as to interactively execute queries on a grid and reconfiguration scripts.

5 Evaluation

We show first in this section the expressiveness of the VMScript language by comparing it to another general purpose scripting language linked to a VM API. We then present several use cases we experimented with VMScript for grid management.

5.1 Comparison with a General Purpose Language

In this section, we compare an action written in VMScript with the same action written in the scripting language Bash. The purpose of the example is to shut down machines in order to perform some maintenance tasks on hardware (for example, changing a hard disk). We want to select machines from their CPU capacity and their kernel version. If these machines do not host VMs, we shut them down. For this experimentation, all machines are stored in a same rack. These machines boot a Linux operating system with different kernel versions and different CPU and memory capacities. Each machine runs a Xen (v3.2.1) hypervisor.

From a UNIX shell, we get the cpu capacity from reading the */proc/cpuinfo* file and just keep the value of the metric *cpu MHz*. We obtain the kernel version by the command *uname -r*. To check that there is no VM hosted by the machine, the Xen API is invoked for listing all hosted virtual machines on a node.

```
1 for machine in $*; do
2   kernel=$(ssh root@$machine uname -r)
3   cpu=$(ssh root@$machine cat /proc/cpuinfo|grep 'cpu MHz'|head -n 1|sed
      s/[^0-9.]//g)
4   if [ $cpu = "2000" -a $kernel = "2.6.26-1-xen-amd64" ]; then
5     if [ -z "$(ssh root@$machine xm li | sed '1d' | sed '1d')" ]; then
6       ssh root@$machine halt
7     fi
8   fi
9 done
```

Action written in a bash script

From the VMScript console, we set a variable "rack" with the rack to analyze. So we select all machines of this rack with the wished "cpu_cap". For the kernel version, the usage of an optional property "os_dist" is required. We query the

grid representation with the function "size()" to check that no VM is running on a machine.

```
1 shutdown($rack/machine[@cpu_cap = 2000][@os_dist ='2.6.26-1-xen-amd64'
      ][size(vm) == 0];
```

Action written in VMScript

We can see the benefits from using the VMScript DSL because of:

- its concision: the VMScript action takes a single line versus 9 lines of codes in Bash,
- its homogeneity and genericity: its not necessary to invoke a specific hypervisor API in the code,
- its guarantees: the shutdown action in VMScript has a precondition to check that there is actually no VM hosted on a machine before shutting it down.

5.2 Some Common Use Cases in Grid Management

Some samples of VMScript code are given afterwards to exemplify the use of the language for grid management.

The action *keep-min-nodes* is used to ensure that a given number of machines is started so that they can easily host new VMs.

```
action keep-min-nodes(grid, nb) {
  for s : $grid//site {
    no_vm = $s//machine[size(vm) == 0];
    on = size($no_vm)
    if($on > $nb) {
      shutdown(subset($no_vm, $on - $nb))
    } else if($on < $nb) {
      power-on(subset($s//machine[@state == 'off'],
        $nb - $on));
    }
  }
}
```

The following action gets servers running a Xen hypervisor then puts a new Xen configuration file and restarts the Xen daemon.

```
action hypervisors-config(grid, hypervisor, filePath) {
  xen = $grid//machine[@hypervisor ='xen' && @os_family == 'Linux'];
  put-file($xen,filePath);
  execute-command($xen,'/etc/init.d/xend restart');
}
```

In the next action, we add a new server in the physical architecture and migrate virtual machines of the most overhead servers to free a piece of memory.

```
action new-machine(grid, hypervisor, filePath) {
  add-elements('new-server.xml');
  new-server = $grid//machine[@name == 'pastel-90'];
  server1 = $grid//machine[@mem_free == min(grid//machine@mem_free)];
  migrate($server1/vm[@mem_need == max(./machine/vm@mem_need)],
  $new-server);
}
```

6 Conclusion

Managing a virtualized grid infrastructure is a hard task and some tools are required to help administrators with this. In the same way, although a grid can aggregate a lot of heterogeneous physical and software resources, it must offer a simple interface to its users, i.e. the application (job) providers. Regarding these preoccupations, we proposed a domain specific language approach for grid management.

More precisely, several DSLs are used for grid description, query and reconfiguration. All these languages rely on the definition of a particular model of a grid. A graph based representation of grids is maintained at runtime and conforms to this model. A first description language allows administrators to specify a grid physical architectures as a hierarchical assembly of physical elements like machines and clusters. A second language is used to group grid users by sets in virtual organizations. The last description language is dedicated to the specification of jobs (called virtual jobs) by users. A job, which has to be executed on the grid, is described essentially as a set of VMs. A user can specify the resources (CPU, memory, etc.) required to run the job.

The VMScript language focuses on querying grid architectures and on grid reconfiguration. A subset of the language is declarative and is used to query the grid through its runtime representation. This query is done by navigating in the graph and selecting elements with some optional predicates. The imperative part adds side effects to the language with control structures and procedures called actions. These ones actually modify the grid architecture, for instance by placing and moving VMs on machines.

Our grid model essentially focuses on the representation of machines as physical resources. It does not deal at the moment with network topology and properties such as latency and bandwidth. However, as this model is extensible without actually modifying the language syntax, these new preoccupations could be introduced for future work provided that a suitable monitoring system gives that information.

References

1. Jeannie Albrecht, David Oppenheimer, Amin Vahdat, and David A. Patterson. Design and implementation tradeoffs for wide-area resource discovery. In *In Proceedings of 14th IEEE Symposium on High Performance, Research Triangle Park*, pages 113–124. IEEE Computer Society, 2005.
2. Globus Alliance. Extended resource specification language (xrsl). Technical report, Globus Alliance, 2009.
3. Ali Anjomshoaa. Job submission description language (jsdl) specification, version 1.0. Technical report, Global Grid Forum, 2005.
4. 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. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

5. Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.
6. Pierre-Charles David, Thomas Ledoux, Marc Léger, and Thierry Coupaye. Fpath & fscript: Language support for navigation and reliable reconfiguration fractal architectures. *Annals of Telecommunications - Special issue on Software Components - The Fractal Initiative*, 64:45–63, 2009.
7. Renato J. Figueiredo, Peter A. Dinda, and José A. B. Fortes. A case for grid computing on virtual machines. In *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 550, Washington, DC, USA, 2003. IEEE Computer Society.
8. R. P. Goldberg. Architecture of virtual machines. In *Proceedings of the workshop on virtual computer systems*, pages 74–112, New York, NY, USA, 1973. ACM.
9. Pascale Vicat-Blanc Primet Guilherme Piegas Koslovski and Andrea Schwertner Char ao. Vxdl: Virtual resources and interconnection networks description language. In *Networks for Grid Applications*, 2009.
10. Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 41–50, New York, NY, USA, 2009. ACM.
11. Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, June 2007.
12. M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30(7):817–840, July 2004.
13. Marvin McNett, Diwaker Gupta, Amin Vahdat, and Geoffrey M. Voelker. Usher: an extensible framework for managing custers of virtual machines. In *LISA'07: Proceedings of the 21st conference on Large Installation System Administration Conference*, pages 1–15, Berkeley, CA, USA, 2007. USENIX Association.
14. Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *In Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, pages 28–31, 1998.
15. Mendel Rosenblum and Tal Garfinkel. Virtual machine monitors: Current technology and future trends. *Computer*, 38(5):39–47, 2005.
16. Borja Sotomayor, Kate Keahey, and Ian Foster. Combining batch execution and leasing using virtual machines. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 87–96, New York, NY, USA, 2008. ACM.
17. World Wide Web Consortium. XML path language (XPath) version 1.0. W3C Recommendation, November 1999. `http://www.w3.org/TR/xpath/`.