Hindawi Publishing Corporation
Advances in Software Engineering
Volume 2012, Article ID 212343, 19 pages
doi:10.1155/2012/212343

Research Article

An SOA-Based Model for the Integrated Provisioning of Cloud and Grid Resources

Andrea Bosin[1, 2]

[1] *Dipartimento di Fisica, Università degli Studi di Cagliari, Complesso Universitario di Monserrato, 09042 Monserrato, Italy*
[2] *Istituto Nazionale di Fisica Nucleare (INFN), Complesso Universitario di Monserrato, Sezione di Cagliari, 09042 Monserrato, Italy*

Correspondence should be addressed to Andrea Bosin, andrea.bosin@dsf.unica.it

Received 15 June 2012; Revised 10 September 2012; Accepted 25 September 2012

Academic Editor: Guoquan Wu

In the last years, the availability and models of use of networked computing resources within reach of e-Science are rapidly changing and see the coexistence of many disparate paradigms: high-performance computing, grid, and recently cloud. Unfortunately, none of these paradigms is recognized as the ultimate solution, and a convergence of them all should be pursued. At the same time, recent works have proposed a number of models and tools to address the growing needs and expectations in the field of e-Science. In particular, they have shown the advantages and the feasibility of modeling e-Science environments and infrastructures according to the service-oriented architecture. In this paper, we suggest a model to promote the convergence and the integration of the different computing paradigms and infrastructures for the dynamic on-demand provisioning of resources from multiple providers as a cohesive aggregate, leveraging the service-oriented architecture. In addition, we propose a design aimed at endorsing a flexible, modular, workflow-based computing model for e-Science. The model is supplemented by a working prototype implementation together with a case study in the applicative domain of bioinformatics, which is used to validate the presented approach and to carry out some performance and scalability measurements.

1. Introduction

In the last years, the availability and models of use of networked computing resources within reach of e-Science are rapidly changing and see the coexistence of many disparate paradigms, featuring their own characteristics, advantages, and limitations. Among the main paradigms, we find high-performance computing (HPC), grid, and cloud. In all cases, the objective is to best provide hardware and software resources to user applications with the help of schedulers, reservation systems, control interfaces, authentication mechanisms, and so on.

At the same time, a number of works [1–4] have proposed a number of models and tools to address the growing needs and expectations in the field of e-Science. In particular, the works in [4, 5] have shown the advantages and the feasibility, but also the problems, of modeling e-Science environments and infrastructures according to the service-oriented architecture (SOA) and its enabling technologies such as web services (WS). Among the main advantages of such approach, we find interoperability, open standards, modularity, dynamic publish-find-bind, and programmatic access.

A detailed comparison of the characteristics of HPC, grid, and cloud paradigms is presented in [6], where it is observed that none of these paradigms is the ultimate solution, and a convergence of them all should be pursued. Even if many computing paradigms show some kind of SOA awareness, heterogeneous resource provisioning still remains an open problem because of the many integration issues, such as usage model, life-cycle management, authentication, application programming interfaces (APIs), adopted standards (or no standard at all), services (e.g., storage, authentication), network segmentation or isolation between different resources, monitoring, workflow languages, and engines.

In this paper, we suggest a model to promote the convergence and the integration of the different computing paradigms and infrastructures for the dynamic on-demand provisioning of the resources needed by e-Science environments. Promoting the integration, cooperation, and interoperation of heterogeneous resources has the advantage of allowing users to exploit the best of each paradigm. In addition, the availability of cloud resources, capable of instantiating standard or custom virtual machines (VMs) on a per-user basis, gives a number of value-added enhancements:

(i) allowing the execution of platform-dependent applications which may be bound to specific operating system flavors or libraries not generally available;

(ii) permiting dynamic provisioning of different workflow engines;

(iii) giving the additional flexibility to run web service applications and enact workflows where and when the user wishes (e.g., users may well decide to take advantage of pay-per-use commercial providers such as Amazon or Rackspace).

Our model is not meant to replace existing HPC, grid, and cloud paradigms, rather it is an attempt aimed at complementing, integrating, and building on them by playing the role of a dynamic resource aggregator exposing a technology agnostic abstraction layer.

At the same time, we aim at endorsing a flexible, modular, workflow-based computing model for e-Science. Our proposal borrows many SOA concepts and standards from the business domain, including the adoption of the *Business Process Execution Language (BPEL)* for workflow design and execution. A motivation of our approach is almost evident: the SOA paradigms, and in particular web services and BPEL, are based on widely accepted standards and supported by many software tools, both open source and commercial.

In addition, we describe a working proof-of-concept implementation together with a case study in the bioinformatics domain, which is used to validate the proposed approach and to address issues such as performance and scalability, shortcomings, and open problems.

Though we are not presenting a revolutionary approach or completely new ideas, we believe that the exploration, exploitation, and integration of existing technologies and tools, especially those based on open standards, in new and different ways can contribute with added value to the field and may be of interest to readers. In particular, with respect to previous work, we have the following:

(i) BPEL has been used unmodified, while other works have added custom extensions;

(ii) the model is open to the integration of both new computing infrastructures and workflow systems;

(iii) the model is meant to work with the widest spectrum of resources (in most works, resources are limited to just one type, e.g., grid);

(iv) open standards have been adopted to the maximum extent (in many other works, nonstandard ad hoc solutions have been adopted);

(v) we are not (yet) proposing a production implementation; instead, we have chosen to explore and verify in practice the possibility of employing tools and technologies based on open standards in new and different ways, strongly believing that this may give a valuable contribution to tackle the problem of heterogeneous resource provisioning;

(vi) we are not focusing on directly improving existing works and systems, rather we investigated a different direction that might be promising, being mainly based on open standards;

(vii) the model is open not only to incorporate previous work (and indeed it does), but also to be incorporated.

The paper is organized as follows. Section 2 reviews some related work, while Section 3 provides a summary of resources and provisioning systems. Section 4 presents a model promoting the convergence and the integration of such systems for the dynamic on-demand provisioning of resources from multiple providers. Section 5 describes a working implementation of the model, Section 6 covers a case study in the bioinformatics domain, and Section 7 presents some of the results and discusses issues and open problems. Conclusions are then drawn in Section 8.

2. Related Work

Software applications have been built to address a wide spectrum of scientific workflows, ranging from basic tools that are designed to handle “desktop” tasks such as simple data analysis and visualization to complex workflow systems that are designed to run large-scale e-Science applications on remote grid resources. These systems need to support multiple concurrent users, deal with security requirements, and run workflows that may require the use of a sophisticated layer of services [7].

In [3], authors identify desiderata for scientific workflow systems—namely, clarity, predictability, reportability, and reusability. Moreover, ease of composition and editing, the ability to automatically log and record workflow enactments, and the flexibility to incorporate new tools are all important features [7]. The interoperability aspects of scientific workflow systems are addressed in [2], which investigates the differences in execution environments for local workflows and those executing on remote grid resources. A complete overview of features and capabilities of scientific workflow systems is presented in [1].

There is a number of widely recognized grid workflow projects like Triana [8, 9], Kepler [10], Pegasus [11], and ASKALON [12]. Many of these began their life in the “desktop” workflow space and have evolved over time to address the large-scale e-Science applications. Specifically designed for the life sciences, Taverna [13, 14] was the first system to recognize the importance of data provenance and semantic grid issues.

While developed for the business domain, technologies like BPEL are recognized suitable to address the requirements of e-Science applications [4], supporting the composition of large computational and data analysis tasks that must be executed on supercomputing resources. BPEL is recognized by [1] as the de facto standard for web-service-based workflows. An architecture for the dynamic scheduling of workflow service calls is presented in [15], where control mechanisms of BPEL are combined with an adaptive runtime environment that integrates dedicated resources and on-demand resources provided by infrastructures like Amazon Elastic Compute Cloud. Reference [16] presents the design and implementation of a workflow management system based on BPEL in a grid environment. Based on BPEL, QoWL [17] and GPEL [18] are significant examples of workflow systems designed for dynamic, adaptive large-scale e-Science applications.

The use of BPEL for grid service orchestration is proposed as foundation in [19] since it fulfills many requirements of the WSRF standard. The appropriateness of BPEL is also examined and confirmed in [20–22]. These works mainly focus on scientific workflows and rely on extending or adapting BPEL, thus creating dialects.

Many authors consider the problem of resource provisioning from the perspective of conventional grid applications. Following this approach, grid and cloud convergence is achieved in such a way to expand a fixed pool of physical grid resources, usually managed by a workload and resource management system (WRMS), by providing new virtual grid resources on-demand and dynamically leasing them from one or more dynamic infrastructure management systems (DIMSs).

Reference [6] presents a hybrid computing model which is aimed at executing scientific applications in such a way to satisfy the given timing requirements. Applications are represented by workflows, that is, a set of jobs with precedence constraints. The basic building block is the elastic cluster, characterized by (1) dynamic infrastructure management services; (2) cluster-level services such as workload management; (3) intelligent modules that bridge the gap between cluster-level services and dynamic infrastructure management services. An infrastructure for the management and execution of workflows across multiple resources is then built by using multiple elastic clusters coordinated by a workflow management system.

Reference [15] explores on-demand provisioning of cloud resources directly at the workflow level, using BPEL as the workflow language. When, during workflow enactment, a service is invoked, the request is routed to the service instance running on the best-matching host (e.g., lowest load); if no best-matching host is available, a new VM is provisioned from a cloud to run a new instance of the required service.

The idea of virtual clusters on a physical grid is explored in [23], where virtual organization clusters provide customized, homogeneous execution environments on a per-virtual organization basis. The authors describe a clustering overlay for individual grid resources, which permits virtual organizations of federated grid users to create custom computational clouds with private scheduling and resource control policies.

The feasibility of using one or more cloud providers for deploying a grid infrastructure or parts of it has been studied in [24]; such an approach permits the elastic growth of the given grid infrastructure in such a way to satisfy peak demands or other requirements.

The approach presented in this work has been previously pursued [5, 25], with particular emphasis on SOA and BPEL adequacy for e-Science environments and less attention to the general problem of resource provisioning.

3. Resources and Provisioning Systems

We start this section with a brief description of the most common resource types available to e-Science environments and then discuss the main features of the existing management systems.

High-performance computing (HPC) resources are tightly-coupled (e.g., by a high-performance communication device such as an InfiniBand switch) sets of computing equipment with (usually) a high degree of hardware and software homogeneity. HPC resources are in most cases managed by legacy schedulers which represent a big issue for interoperability. HPC infrastructures typically provide the following:

(i) high-performance hardware enabling strongly coupled parallelism,

(ii) resource scheduling/reservation,

(iii) homogeneous hardware and software environments.

Grid resources are loosely coupled sets of computing equipment with (usually) a local (e.g., per site) high degree of hardware and software homogeneity. Physical separation is not an issue and resources are managed through the abstraction, of virtual organizations (VOs) with the possibility of federating many different VOs. The open grid services architecture [26], or OGSA, is a framework for developing grid management services according to the service-oriented architecture, thus enabling ease of integration and interoperability between sites, even if grids have limited interoperability between different grid software stacks [6]. Grid infrastructures may offer the following:

(i) large sets of computing resources,

(ii) resource scheduling/reservation,

(iii) advanced storage services,

(iv) specialized workflow systems,

(v) advanced monitoring services,

(vi) homogeneous operating system and software environments.

Cloud resources are based on loosely coupled sets of computing equipment with no need or guarantee of hardware and software homogeneity and in many cases distributed at different physical locations. In this paradigm, the physical machine (PM) hardware and software are almost completely hidden, and a virtual machine (VM) abstraction is exposed to the user. Most cloud management systems are designed according to SOA concepts and expose their functionality through Soap-based or RESTful interfaces, thus improving ease of access, integration, and interoperability. One interesting characteristic of clouds is the ability to provide resources with certain characteristics, that is, specific software libraries or configurations, dynamically on demand and on a per-user basis [24]. Cloud infrastructures present the following:

(i) extremely flexible/customizable operating system and software environments,

(ii) API based on open standards (e.g., Soap, REST),

(iii) on-demand provisioning,

(iv) increasing the number of resources offered by commercial providers.

Specialized resources are "unique" nodes on the network where a very specific activity can take place (e.g., data acquisition from an attached physical device), but sharing such resources on the network may not be straightforward, since in many cases they expose custom APIs.

At last, *personal resources*, such as desktops or laptops, provide users with the maximum ease of use in terms of flexibility and customization, often at the expense of limiting the interoperability with other resources. A personal resource usually is the main door for accessing the network, and it is both the starting and end point for user interaction with e-Science environments (e.g., start an experiment, monitor its execution, collect results, and write reports).

HPC and grid resources are usually controlled by *workload and resource management systems (WRMSs)* which support (1) resource management; (2) job management; (3) job scheduling. The term job refers to the set of instructions needed to run an application or part of it on a batch system, written according to some Job Description Language (JDL). A WRMS can be built on top of another WRMS, as it is usually the case with grid middleware which relies upon an underlying batch system referred to as a local resource management system (LRMS).

Users are granted access to a WRMS in a number of ways: (1) interactively from a front-end machine using a command line interface (CLI); (2) programmatically by means of specialized API; (3) in some cases, programmatically by means of technology agnostic API, for example, Open Grid Forum Distributed Resource Management Application API (DRMAA) [27]. Authentication mechanisms may range from simple credentials such as user/password to sophisticated X.509 certificates based on a public key infrastructure, both for users and software services.

An example of grid WRMSs is gLite computing resource execution and management (CREAM) [28] for which job submission and execution can be summarized as follows: (1) a grid user, using his/her X.509 credentials, obtains an authentication proxy from the VO membership service (VOMS), prepares the job, and submits both the JDL file and proxy to CREAM service; (2) user authentication and authorization (by Local Centre Authorization Service, LCAS) is performed; (3) user request is mapped to a CREAM command and queued; (4) the CREAM command is processed and mapped to a specific LRMS request, and grid user is mapped to an LRMS user (by Local Credential Mapping Service, LCMAPS); (5) the request is submitted to the LRMS.

Cloud resources are typically supervised by *dynamic infrastructure management systems (DIMSs)*, which offer two kinds of functionality: (1) physical resource management; (2) service management, where provisioned services are implemented using virtual resources. We are mainly interested in the *infrastructure as a service (IaaS)* model, which offers the maximum flexibility by providing virtual machines and the management interface, as well as storage. User access is performed: (1) interactively from a client application; (2) through a browser by means of suitable plug-ins; (3) programmatically by means of specialized API; (4) programmatically by means of technology agnostic API such as Amazon Elastic Compute Cloud (EC2) and Simple Storage Service (S3) [29, 30], or Open Cloud Computing Interface (OCCI) [31]. Authentication mechanisms may vary as in the case of WRMS.

OCCI is an open protocol and API originally created for all kinds of management tasks of (cloud) infrastructures such as deployment, autonomic scaling, and monitoring. It has evolved into a flexible API with a strong focus on interoperability while still offering a high degree of flexibility. OCCI makes an ideal interoperable boundary interface between the web and the internal resource management system of infrastructure (cloud) providers.

At the heart of OCCI model, we find the *resource* which is an abstraction of a physical or virtual resource, that is, a VM, a job in a job submission system, a user, and so forth. The API allows for the creation and management of typical IaaS resources such as *compute*, *network,* and *sorage*.

OCCI is being implemented only in the latest releases of many cloud DIMSs we have tested (mainly Eucalyptus, OpenNebula, and OpenStack), and as such we were not able to test it extensively, yet we believe it is a first important and promising step towards cross-cloud integration.

An example of a cloud DIMS is Eucalyptus Community Cloud [32] where VM deployment follows a number of steps: (1) a user requests a new VM to the cloud controller (CLC) specifying disk image and characteristics (or flavor), (2) the CLC authorizes the request and forwards it to the cluster controller (CC), (3) the CC performs resource matching and virtual network setup (hardware, IP addresses, and firewall rules) and schedules the request to a node controller (NC), (4) the NC retrieves disk image files from Walrus repository service, (5) the NC powers up the VM through the configured hypervisor, and (6) the user logs into the VM.

Quite interestingly, in [6], it is recognized that both WRMS and DIMS can be well described by the managed computation factory model introduced in [33] and shown in Figure 1. In this model, clients of computational resources do not directly interact with the bare resources, rather they work with the *managed computation* abstraction. A managed computation can be (1) started, (2) stopped, (3) terminated, (4) monitored, and/or (5) controlled by interfacing with a *managed computation factory*. The factory is in charge

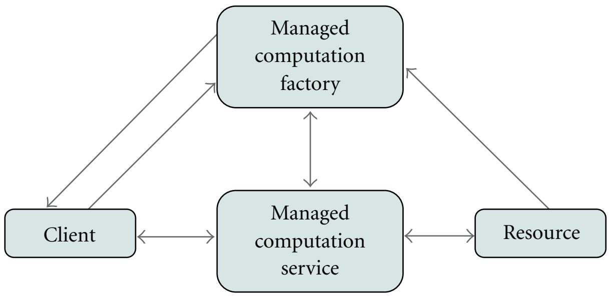

Figure 1: Managed computation factory model.

of creating the managed computation, which is assigned to one or more computational resources. The managed computation often operates as a service, hence the name *managed computation service* in Figure 1.

4. Integrated Provisioning of Resources

To address the needs of a wide community of users wishing to exploit at best the resources they have access to (e.g., those granted to them at no charge), we propose to design an abstraction layer between applications and resources, in such a way that resources can be requested, provisioned, used, monitored, and disposed without worrying about the underlying technological infrastructure. Promoting the integration, cooperation, and interoperation of heterogeneous resources has the advantage of allowing users to exploit the best of each paradigm.

In a scenario based on "smart" resource aggregation, new applications may enjoy many benefits; it is possible to imagine a distributed application orchestrated by a workflow where (1) a VM on a cloud runs the workflow engine, (2) the bulk computation is performed in parallel on an HPC infrastructure, (3) data postprocessing is executed on a grid (or cloud), (4) results are viewed on a personal resource, and optionally (5) steps 2–4 are part of an iterative computation.

At a first sight, there may be no benefits for existing applications, unless they are modified, but cloud flexibility may nevertheless help. Indeed, a user application developed for a grid, or other resources may become unusable if (1) the user has no (or no longer) access to the grid, or (2) an update in grid operating system/middleware requires modifications to the application. A possible solution is then to deploy the desired flavor of grid on top of a cloud, as explored in [24], and run the unmodified application.

In this section, we describe a model to promote the convergence, integration, cooperation, and interoperation of different computing paradigms and infrastructures, for the dynamic on-demand provisioning of the resources from multiple providers as a cohesive aggregate. The model, depicted in Figure 2, builds upon HPC, grid, and cloud systems leveraging existing WRMS and DIMS and connecting the different components through SOA interfaces whenever possible, directly or by means of suitable wrappers/adapters.

At the same time, the proposed design aims at endorsing a flexible, modular, workflow-based collaborative environment for e-Science. The latter sees the integration and interoperation of a number of software components, such as

(i) workflows or processes, to define and coordinate complex scientific application or experiments;

(ii) service interfaces, to expose the logic of scientific applications;

(iii) components, to implement rules and perform tasks related to a specific scientific domain.

At the implementation level, the choice of SOA as the enabling technology for a common integration and interoperation framework sounds realistic due to the:

(i) availability of SOA standards for workflow systems (i.e., BPEL);

(ii) availability of web service libraries to build new applications and to wrap existing ones;

(iii) existence of SOA standards covering areas like data access, security, reliability, and so forth;

(iv) access to a number of computing infrastructures (e.g., grid) is, at least partially, SOA aware.

In the envisioned collaborative environment, resources are needed both for running scientific (web) services and processes, and we distinguish two kinds of resource requests:

(i) scientific service (SS) requests for the execution of scientific application modules; scientific services, possibly exposed as web services, have the responsibility of the computation on the assigned resources;

(ii) scientific process requests for the execution of scientific workflows; scientific workflows or processes, enacted by a workflow engine, orchestrate one or more scientific services and are in charge of (1) obtaining the resources needed for each computation represented by a service and (2) managing their life cycle; scientific processes may as well expose workflow instances through a web service interface.

Scientific service and process requests, conforming to SOA, are described by XML documents (see Figure 4 for an example).

Resources, which can be physical (PM) or virtual (VM), are classified accordingly: *worker nodes* run scientific applications modules, while *workflow engine nodes* are dedicated to the execution of workflow engines (WfEs). The distinction between workflow engine and worker nodes is mainly logical; a WfE node is a VM (or PM) resource allocated for the execution of a workflow engine, while a worker node is a PM or VM allocated to run a module of a scientific application in the form of a web service or other type of software component. However, it should be noted that, at different times, a resource may act as a worker node or as a workflow engine node, depending on the managed computation assigned to it. In addition, all the resources

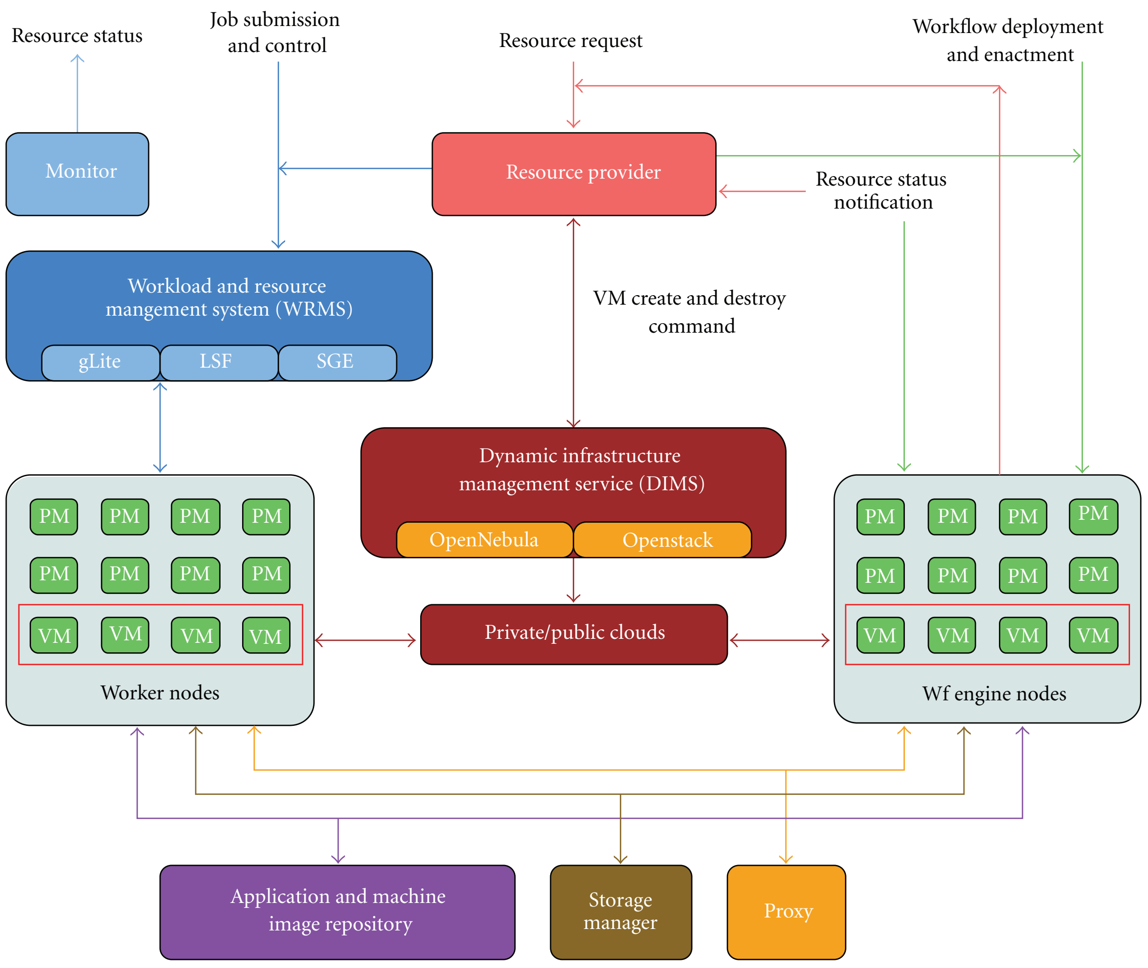
Resource status
Job submission and control
Resource request
Workflow deployment and enactment
Monitor
Resource provider
Resource status notification
Workload and resource mangement system (WRMS)
gLite
LSF
SGE
VM create and destroy command
Dynamic infrastructure management service (DIMS)
OpenNebula
Openstack
PM
VM
Worker nodes
Private/public clouds
Wf engine nodes
Application and machine image repository
Storage manager
Proxy

Figure 2: Model of dynamic on-demand resource provisioning system.

belong to one or more WRMSs and/or DIMSs, with the possible exception of some personal and specialized resource.

Figure 3 shows a diagram synthesizing the dependencies and interactions between users and the main components of the model and its subsequent implementation, which we are going to describe in greater detail in this and the next sections.

4.1. Resource Provider. At the core of the model, we find the *resource provider* service which acts as a high-level managed computation factory exposing a web service interface:

(i) it accepts requests for resources from clients;

(ii) it translates and forwards requests to the underlying WRMS and DIMS;

(iii) it receives resource heartbeats and notifications;

(iv) it notifies asynchronous clients of resource endpoint, availability, and status changes.

All resource requests are submitted to the resource provider, which takes the appropriate actions based on the request type, as we are going to describe in the following subsections together with the other components of the model. A successful resource request will start the desired managed computation and provide the user with the computation service network end point and identifier: due to the nature of the resources and of their management systems, this may not immediately happen, leaving users waiting for a variable amount of time. Consequently, the resource provider offers different strategies to inform users when their managed computation is at last started; in particular, clients may submit their resource requests in a number of ways:

(i) a synchronous request returns successfully within a given timeout only if the computation starts before the timeout expires;

(ii) an asynchronous request without notification returns immediately: clients may periodically poll the resource provider to know if the computation has started;

(iii) an asynchronous request with notification returns immediately but allows clients to be notified as soon as the computation starts (clients must be able to receive notifications).

The resource provider is also responsible for disposing resources (canceling jobs in a WRMS or terminating VMs in

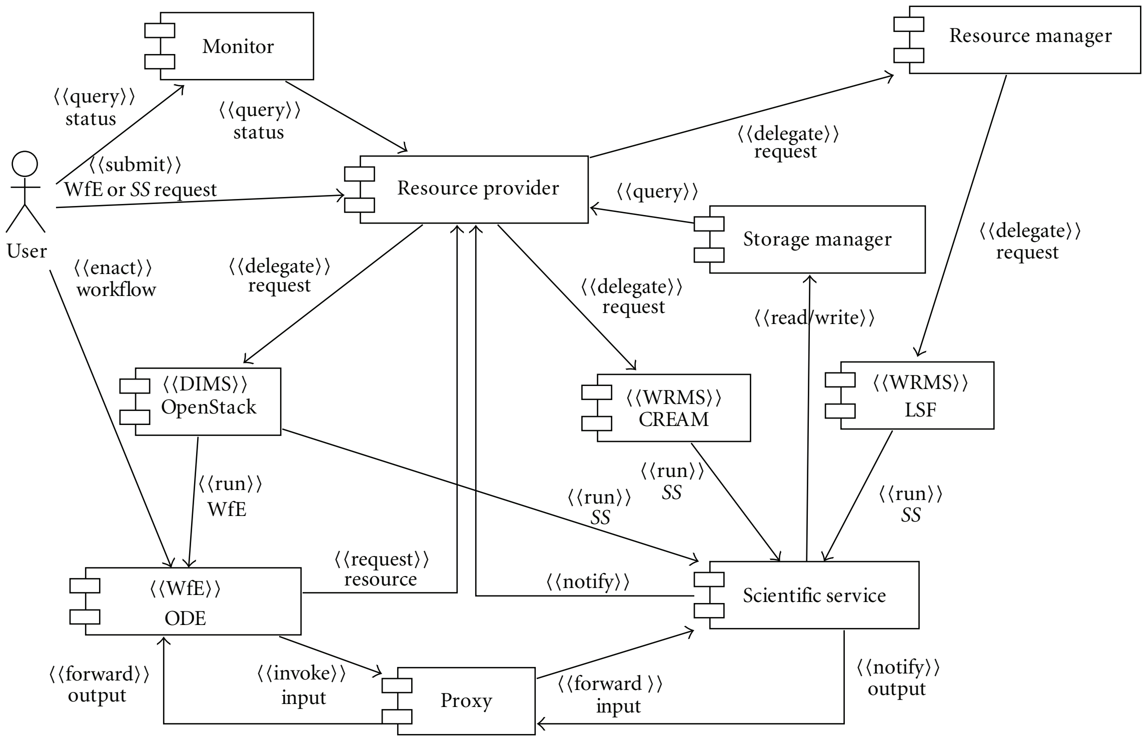

Figure 3: Component diagram with basic dependencies.

a DIMS) upon explicit request, resource lifetime expiration, or in case of problems. The associated scientific service or workflow engine is informed of the imminent disposal, is explicitly in charge of its own clean-up (and should be implemented accordingly), and is required to notify the resource provider immediately before exiting.

4.2. Scientific Process Requests. A scientific workflow or process is a high-level description of the process used to carry out computational and analytical experiments, modeled as a directed graph consisting of task nodes and data-flow or control-flow edges denoting execution dependencies among tasks. A process instance is executed by a workflow engine, an application which needs a computing resource to run on. During workflow execution or enactment, workflow engines generally schedule tasks to be invoked according to the data-flow and control-flow edges. A request for a resource dedicated to the execution of a workflow engine, workflow deployment, and the subsequent enactment of a workflow instance is satisfied by materializing an appropriate VM, that is, an operating system embedding the desired workflow engine. The request specifies the following:

(i) the cloud type (e.g., Eucalyptus Community Cloud) that determines the API to use when connecting to the cloud-specific *DIMS*;

(ii) the DIMS interface endpoint, typically a URL;

(iii) the user authentication credentials;

(iv) the VM characteristics or flavor, for example, number of CPUs, RAM size, and so forth;

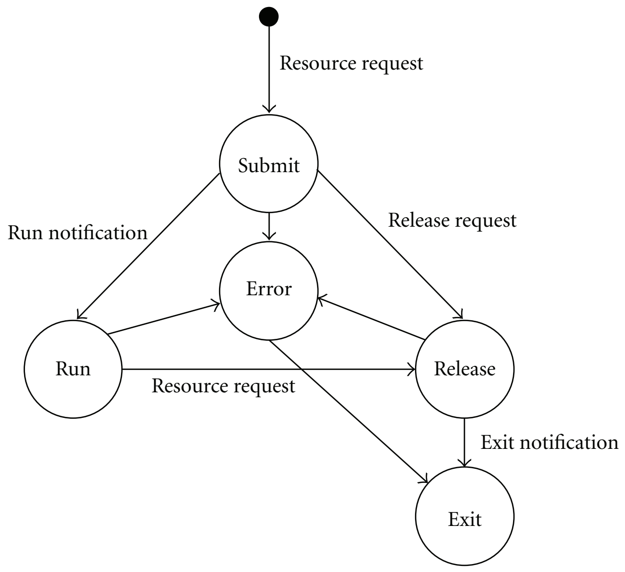

Figure 4: State diagram for an abstract resource.

(v) a custom VM image embedding the desired workflow engine;

(vi) the workflow image, that is, the URL of the archive containing the BPEL documents and related files.

When the resource provider processes the request, a VM creation command is submitted to the cloud-specific DIMS. The latter schedules the instantiation of a new VM with the given characteristics on the corresponding cloud (pool of resources) and returns a VM identifier or an error. As soon as the VM is up and running, the embedded workflow engine is started and the workflow image is downloaded and deployed; the resource provider is then notified of the WfE interface end point.

A workflow is an abstract representation of a scientific application and is made of a set of tasks or jobs with precedence constraints between them [6]. Each job needs a resource to run on, and the workflow places the corresponding scientific service requests on the resource provider.

4.3. Scientific Service Requests. A request for a resource needed to run a job, that is, a module of a scientific application in the form of a scientific service, specifies the following:

(i) the resource type, either a resource managed by a *WRMS* (e.g., CREAM) or a cloud resource made available through a DIMS;

(ii) the WRMS or DIMS interface endpoint (URL);

(iii) the user authentication credentials;

(iv) the resource requirements, for example, RAM size;

(v) the web service (application) image, that is, the URL of the archive containing the web service executable and related files.

The resource provider forwards the request to the specified DIMS or WRMS. A conventional job submitted to an HPC or grid batch system contains the instructions needed to start the computation, written according to some Job Description Language (JDL). In our case, the job responsibility is to start the web service and wait until it stops, while the real computation is orchestrated by a workflow through one or more invocations of the web service. The resource provider is in charge of preparing the job submitted to the WRMS by translating the specifications contained in the resource request (an XML document) into the corresponding JDL. Job submission to a WRMS returns a job identifier or an error. If the job is to be executed by a dedicated VM allocated on a cloud, the VM is created as described in the previous section, except that a standard machine image is used unless a custom image is specified; the VM is instructed to execute the job after booting the operating system.

As soon as the job is started on a worker node, be it a PM or VM, it notifies the workflow (via the resource provider) of the endpoint of the web service, which is ready and listening for incoming messages. When all the resources requested by the workflow are available, the execution can proceed with the invocation of all the required web services in the specified order.

Algorithm 1 shows a fragment of the XML document representing a scientific service request for a cloud resource, where the cloud interface is Amazon EC2, the WS image can be download from the URL http://ws-cyb.dsf.unica.it/ws/jar/WekaDataMining.jar, and the cloud provider is OpenStack [34] with its own set of requirements (endpoint, region, VM image, and flavor) and user credentials (username/password).

4.4. External Resources. In the foreseen environment, specific activities (e.g., data acquisition from an attached physical device) can take place on specialized nodes on the network; these activities may be exposed by dedicated static services which can be invoked from within a workflow.

In addition, an interesting degree of interactivity may be added to plain workflow execution by invoking a visualization service, for example, at the end of each iteration in an iterative simulation. Such visualization service may be well executed on a personal resource, such as the user desktop, to give a real-time visual feedback of the state of the simulation.

If the service is static, its endpoint is "immutable," and it can simply be hard-coded into the workflow description, but the visualization service presents a problem. Hard-coding a user-specific endpoint (the desktop network address) makes the workflow unusable by different users. We explicitly handle resource requests that refer to external resources and for which creation is not needed. This allows workflows to manage all jobs exactly in the same way, delegating to the resource provider the responsibility for provisioning a new resource only if needed. The latter is directly notified by resources of their endpoint on the network, and in turn, it can notify workflows in such a way to allow dynamic endpoint injection.

4.5. Proxy. In general, worker nodes and workflow engine nodes live on different private networks which are not directly connected to one another (the only assumption we do is that they can open network connections towards the Internet through some kind of Network Address Translation or NAT service), so a *proxy* service is essential in routing messages from scientific processes (workflows) to scientific services (web services) and vice versa.

4.6. Other Services. The other components that complete the model are briefly described here, leaving some details to the next section. The *storage manager* provides temporary storage to data-intensive scientific services; multiple storage managers can coexist in such a way to ensure optimal performance and scalability when needed. The *monitor* is a simple service that can be queried for resource status information. The *application and machine image repository (AMIR)* hosts the applications (e.g., scientific service and process images) and virtual machine disk images that will be executed on the provisioned resources.

5. Implementation

In this section, we describe a working prototype implementation of the model outlined before. We have tried to give a high-level schematic description of the implementation in such a way to avoid overwhelming the reader with unnecessary and less interesting details. We have privileged

```
<BSRequest>
  <BSScheduler>EC2</BSScheduler>
  http://ws-cyb.dsf.unica.it/ws/jar/WekaDataMining.jar
  <BSJavaOptions>-Xmx2048m</BSJavaOptions>
  <BSRequirements>
    <EC2Params>
      <ec2Provider>nova-ec2</ec2Provider>
      <ec2Endpoint>http://172.16.3.3:8773/services/Cloud</ec2Endpoint>
      <ec2Region>nova</ec2Region>
      <ec2Image>ami-lennyws</ec2Image>
      <ec2Flavor>m1.small</ec2Flavor>
    </EC2Params>
  <BSRequirements>
  <BSCredentials>
    <CryptedPWAuth>
      <username>... </username>
      <cryptedPW>... </cryptedPW>
    </CryptedPWAuth>
  <BSCredentials>
</BSRequest>
```

Algorithm 1: XML fragment of the document representing a scientific service request.

the description of the idea and of the issues of the single components in the context of the presented model.

According to the general philosophy of an SOA-based framework, we have developed a number of infrastructure (web) services corresponding to various components of the model—using the Java programming language—and reused open-source tools and systems whenever possible, operating the necessary integration. Figure 3 shows the component diagram of the presented implementation with the basic dependencies between components, as better described in the following subsections.

Integrating WRMS and DIMS requires (1) that they expose a network-enabled public interface, possibly based on SOA standards and (2) embedding the necessary logic into a module of the resource provider. Such integration is usually performed for entire classes of WRMS or DIMS: a specific instance may then provide its resources at any time as soon as a resource request refers to it. This allows, for example, drawing resources from two or more Eucalyptus Community Cloud infrastructures at the same time.

In this work, we have developed modules to interact with the following WRMSs:

(i) gLite v3.2 compute element (CE) [35];

(ii) gLite v3.2 computing resource execution and management (CREAM) [28];

(iii) platform load sharing facility v6.2 (LSF) [36];

(iv) oracle grid engine v6.2 (SGE) [37].

CE and CREAM expose OGSA-compliant interfaces and, in addition, it is possible to resort to existing Java API such as jLite [38]. On the contrary, LSF does not offer a similar functionality, so we have developed a simple but effective web-service wrapper, the *resource manager*, and deployed it to one of the submission hosts, on top of the LSF command line interface. We have followed the same strategy for SGE, too, even if the latter supports the Open Grid Forum Distributed Resource Management Application API (DRMAA) [27].

On the cloud side, we have integrated the following:

(i) Open Nebula [39] defined by its developers is as an "industry standard open source cloud computing tool";

(ii) Open Stack [34] is an open-source porting of Rackspace's DIMS;

(iii) Eucalyptus Community Cloud [32] is presented as "a framework that uses computational and storage infrastructure commonly available to academic research groups to provide a platform that is modular and open to experimental instrumentation and study."

According to their documentation, all these systems expose a RESTful interface compatible with Amazon Elastic Compute Cloud (EC2) [29] and Simple Storage Service (S3) API [30], which are becoming a de facto standard. In practice, the compatibility is not full, but for OpenStack and Eucalyptus, we have successfully employed such interface with the help of the jclouds library [40]. OpenNebula needs additional components to expose the EC2 interface, so we have decided to use its native interface, which is anyway satisfactory.

The above DIMS can cope with different virtualization systems (hypervisors): *Xen* and *Kernel-based virtual machine (KVM)* are the most commonly used. Even if we find KVM more straightforward to use on computers that support hardware-assisted virtualization (both Intel VT-x and AMD-V), the specific hypervisor is almost completely hidden to the user by the same abstraction layer, that is, the *libvirt* virtualization API [41].

5.1. Resource Provider. In the prototype implementation, the *resource provider* functionality is mapped onto a hierarchy of Java classes and is exposed through a web service interface. It would be impossible to describe all classes in detail here, so we only list their key responsibilities:

(i) process all resource requests and make up the corresponding jobs;

(ii) interact with WRMS and DIMS for job and VM scheduling, execution, and control;

(iii) receive status notifications from resources and deliver them to the interested entities (e.g., workflows or monitoring applications);

(iv) dispose resources;

(v) manage resource context and status;

(vi) register proxies, storage managers, and monitoring applications;

(vii) trace message flow and inform registered monitoring applications;

(viii) enforce elementary security practices: authentication credentials containing plain text passwords must be encrypted with the resource provider public key before transmission; if necessary, the use of one-time security tokens can be enabled;

(ix) assign registered proxies and storage managers to scientific services.

5.2. Abstract Resources. The Java classes implementing the resource provider functionality work with an abstract representation of scientific process and service resources, embodied by the class SSResource, which does not depend on the resource type. The dynamic behavior of an abstract resource is captured by the state diagram shown in Figure 4.

If a resource request is permissible, a new instance of the specific SSResource subclass is created with status set to Submit and the corresponding physical or virtual resource creation advances as described in Section 4. As soon as the resource becomes ready, the associated scientific process or service sends a (resource status) notification message to the resource provider, and such event triggers a change of status from Sumbit to Run. Only at this point, the resource can be used, and the resource provider notifies all the interested entities. Once the resource is no longer needed, it may be disposed; its status is then set to Release and the disposal of the corresponding physical or virtual resource proceeds according to the previous section. The scientific process or service sends another (resource status) notification message that causes the status to change from Release to Exit.

All the other state transitions are triggered by error conditions.

5.3. Scientific Services. *Scientific services* implement the logic of scientific applications, usually in a modular form where the whole logic is partitioned into reusable, loosely coupled, cohesive software modules. In an SOA-based framework, scientific services expose their functionality through a web service interface formally described by a WSDL document. To be best integrated in our model, scientific services, in addition to the core scientific functionality just described, must implement some housekeeping operations, too:

(i) immediately after start-up, they must notify the resource provider of their status (RUN);

(ii) immediately before exiting, they must notify the resource provider of their status (EXIT);

(iii) they must retrieve from the resource provider the assigned proxy and storage manager service end points;

(iv) they should expose some standard operations such as version(), load(), and exit(),

(v) they should provide asynchronous operations for long-running computations.

To ease the realization of new scientific services, or for wrapping existing applications into a web service container, we have developed a small housekeeping Java library implementing the above common tasks. As a simple example, consider the *MyScientificService* web service class, implementing the *MyScientificServicePortType* interface (defined by the corresponding WSDL port type) and exposing a *myOperation* operation. The simplified class diagram that illustrates how the scientific service can be implemented in Java, taking advantage of the housekeeping library, is shown in Figure 5. The responsibilities of each class in the diagram are the following:

(i) *SS*: common functionality useful to all scientific services;

(ii) *SSPublisher*: entry-point class which publishes the scientific service and initializes the network channel toward the proxy;

(iii) *SSConnectionClient*: it manages network channel and receives input messages from proxy;

(iv) *SSNotify*: it manages notification messages;

(v) *SSThread*: common functionality useful to all implementations of scientific service synchronous/asynchronous operations;

(vi) *MyScientificService*: scientific service implementation;

(vii) *MyScientificServicePortType*: scientific service interface with the list of all exposed operations;

(viii) *MyOperationThread*: implementation of the scientific service operation *myOperation* which may be executed asynchronously in a separate thread (useful for long-running computations).

Scientific service images are uploaded to the AMIR to be retrieved and deployed by scientific resource requests.

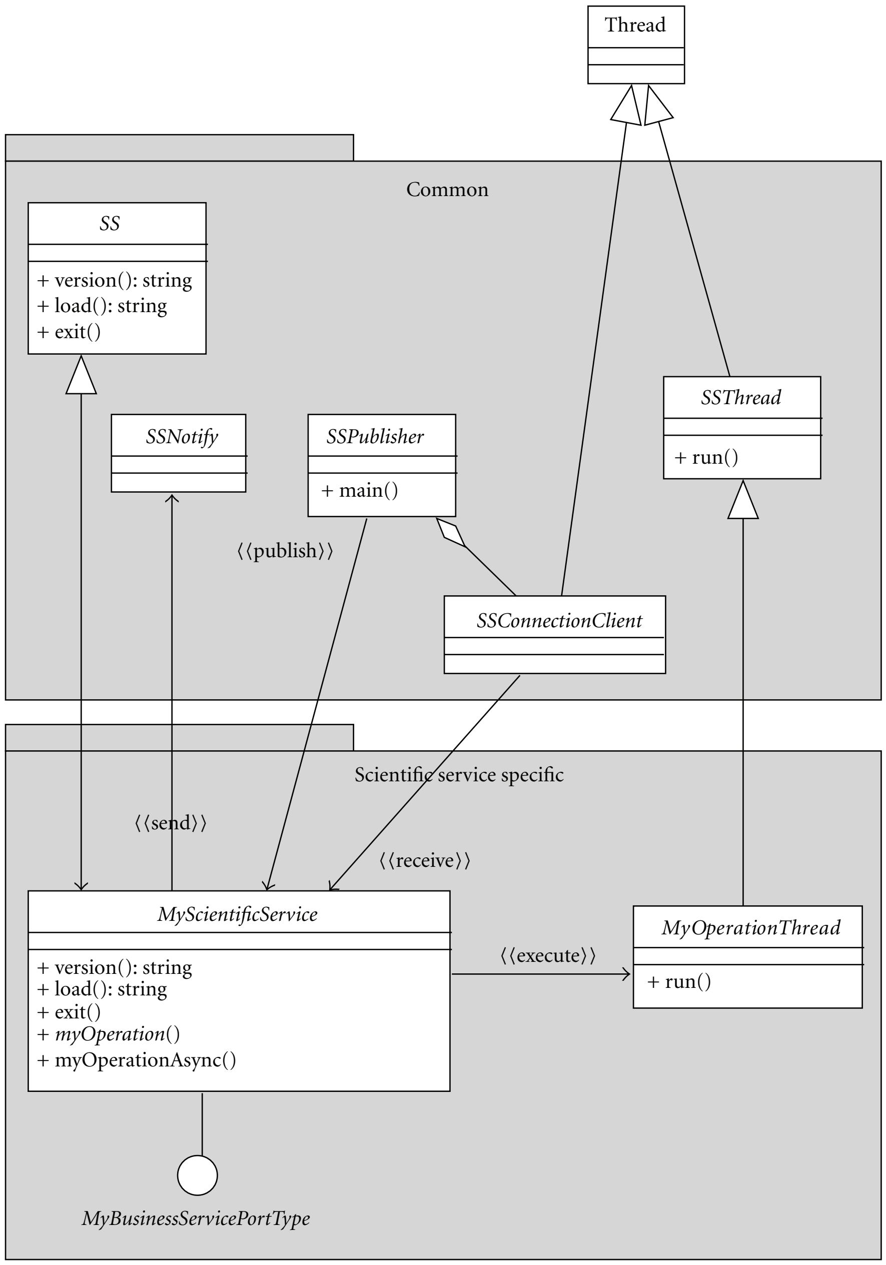

Figure 5: Scientific service class hierarchy.

5.4. Scientific Processes and Workflow Engines. As suggested in [5, 15], the choice of a workflow language naturally leads to the Business Process Execution Language [42] which belongs to the family of SOA standards. BPEL is complementary to the web service definition language (WSDL) [43], and in fact, BPEL instances are themselves web services. Both languages are XML based, and both make use of the XML schema definition language (XSD). A number of BPEL engines were considered:

(i) sun BPEL engine for Glassfish server (no longer maintained);

(ii) Oracle BPEL Process Manager in the Oracle SOA Suite 11g (commercial) [44];

(iii) ActiveVOS BPEL execution engine (commercial) [45];

(iv) Orchestra (open source) [46];

(v) Apache Orchestration Director Engine or ODE in brief (open source) [47].

Due to our preference for open-source software and self-contained tools, in the current implementation, we have chosen to work with ODE v1.3.5. This requires (1) a module in the resource provider to manage WfE requests, (2) a simple SOA wrapper to interface ODE with the resource provider and clients, and (3) a VM image embedding ODE. The WfE node component is then a VM image which provides the guest operating system preinstalled with some ready-to-run workflow engine such as ODE.

It is worth mentioning that the implementation is open to the integration of all standalone self-contained workflow engines, based on BPEL or not, that can be remotely executed on a nongraphical resource.

The VM disk image embedding ODE is deployed to the AMIR together with the workflow images.

5.5. Network Connectivity. One important practical issue not immediately evident from the model is the complexity of the network connections between the different components. We start from some basic assumptions.

(i) The resource provider must live on a public network so that every other component can contact it, including all the users of the system which, in a collaborative distributed environment, can be located anywhere.

(ii) Worker and WfE nodes, as already observed, may live on different private networks which are not directly connected to one another, but we assume that they can open network connections towards the Internet through some kind of NAT service. The same is true for personal resources.

(iii) WRMS and DIMS, if not living on a public network, must be accessible from the resource provider (e.g., via a virtual private network or VPN).

(iv) The application and machine image repository may be on a public network or may be replicated on the private networks attached to each pool of PM/VM.

(v) The storage manager must live on a public network to allow for the sharing of data.

In addition, we have the following constraints:

(1) the resource provider (public network) is required to notify a WfE (private network) when a requested scientific service resource is ready;

(2) a WfE needs to send request-response or one-way messages to scientific services running on WN or personal resources and living on different private networks;

(3) a scientific service running on a WN may need to send (notify) one-way messages to a WfE which is not on the same private network.

The first constraint can be satisfied by setting up a VPN between the WfE node and the resource provider: the VM that executes the WfE can be instructed to initialize the VPN during start-up; the other two by implementing a proxy service as described in the next subsection. The resulting network connectivity is synthesized in Figure 6.

5.6. Proxy. The *proxy* service acts as an intermediary for routing (Soap) messages from WfE to scientific services and vice versa. When the resource provider needs to notify a WfE about the endpoint (URL) of a newly started scientific service, it replaces the scientific service private network address with the proxy public address. In this way, the messages sent by a WfE to a scientific service are in fact sent to the proxy, which performs a sort of message routing to its final destination, based on a special message tag.

For improved scalability, many proxies can register with the resource provider. The latter assigns one of the available proxies to each scientific service. Each proxy is then in charge of routing messages to its own pool of scientific services that may live on different private networks. In principle, we could link the proxy and its pool of services with a VPN, but setting it up requires privileges usually not granted to a job in a WRMS. In practice, every scientific service, upon start-up, simply opens and shares a persistent bidirectional network connection with its own proxy; the latter will use this channel to route all the messages directed to the service.

The s*cientific service proxy* component manages request-response and one-way messages from WfE to scientific services, while the notification proxy component is responsible for managing one-way messages from scientific services to WfE.

5.7. Other Services. As previously discussed, *storage managers* provide temporary storage to data-intensive scientific services. When the output from a scientific service serves as input for a subsequent service invocation during workflow execution, it would be ineffective to transfer the data back to the workflow and forth again to the next service. The storage manager handles plain and efficient HTTP uploads and downloads, assigning a unique public URL to the uploaded data so that they can be easily retrieved later when needed. many storage managers can register with the resource provider, thus ensuring optimal performance and scalability.

Monitor is a simple application with a web service client interface on the resource provider side and an HTTP/HTML interface on the user side, which can be queried for resource status. Monitor shows information, in XHMTL format, about the resource context related to scientific services, such as resource identifier, status, WSDL port type, last operation invoked, and the related time stamp.

The *application and machine image repository* is distributed over a number of components: a simple HTTP server hosts all the scientific service and process images, while every DIMS manages its own dedicated virtual machine image repository.

6. Case Study

The case study focuses on a specific bioinformatics domain: the application of machine learning techniques to molecular biology. In particular, microarray data analysis [48] is a challenging area since it has to face with datasets composed by

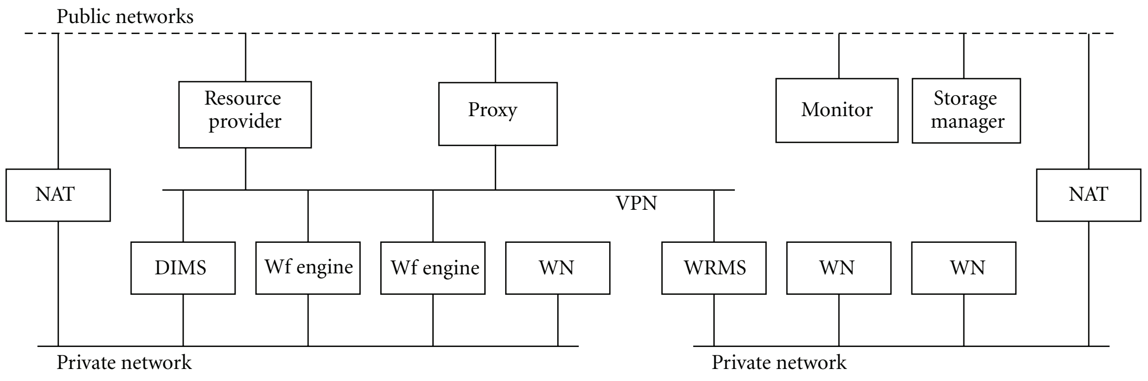

Figure 6: Schema of network connectivity.

a relatively low number of records (samples) characterized by an extremely high dimensionality (gene expression values).

6.1. Scientific Context and Background. Microarray experiments generate datasets in the form of $M \times N$ matrices, where M is the number of samples (typically $M \sim 100$), and N is the number of genes ($N > 10{,}000$). Such gene expression data can be used to classify the biologic samples based on the pattern of expression of all or a subset of genes. Reference datasets, needed by machine learning techniques, are generated from the experimental data by assigning a class label to all the samples (this is usually done manually by a domain expert). The construction of a classifier proceeds by (1) selecting an informative set of genes (features or attributes) that can distinguish between the various classes, (2) choosing an appropriate mathematical model for the classifier, (3) estimating the parameters of the model from the data in the *training set*, that is, a properly chosen subset of the reference dataset, and (4) calculating the accuracy of the classifier using the *test set*, that is, the remaining independent subset not used in training the classifier. With the help of the resulting classifier, class labels can then be automatically assigned to new unclassified microarray samples with an error margin which depends on the accuracy of the classifier.

Due to the huge number of genes probed in a microarray measure, the possibility of training a classifier using the expression values of a smaller but significant subset of all the genes is of particular interest. Many mathematical and statistical methods (e.g., chi-square) exist for selecting a group of genes of a given size and, after filtering out from the dataset all the other genes, a classifier can be trained and its accuracy tested. The subset of significant genes is then iteratively expanded or collapsed, and the previous procedure is repeated; in this way, it is possible to compare the accuracy of the classifier as a function of the size of the subset (i.e., number of attributes) and try to determine the “optimal” group of genes (Figure 7).

The reference dataset used in our experiment covers microarray measures related to the acute lymphoblastic leukemia [49], with 215 samples in the training set and 112 samples in the test set characterized by 12557 gene expression values and 7 class labels.

6.2. User Interaction. User interaction should be kept as much as possible (1) well-defined, (2) simple, and (3) independent of the WRMS/DIMS used, that is, technology agnostic; conforming to SOA, user interaction is based on the exchange of XML documents.

Users wishing to run a workflow-based application should provide the following information:

(i) URL of the image of the workflow to be deployed and enacted on demand (from a public repository or user provided, e.g., AMIR);

(ii) URL of the images of all scientific services to be started on demand and used by the workflow (from a public repository or user provided);

(iii) endpoint and authentication credentials of the WRMS/DIMS required to provide the resources to run the on-demand workflow engine and scientific services.

Based on the above information, users (1) prepare and send to the resource provider the workflow engine request document, (2) when the engine becomes ready, enact one or more workflow instances by preparing and sending the corresponding input documents, and (3) when finished, send to the resource provider the release request for the engine.

According to the information specified in the input document, the workflow is in charge of (1) composing and sending to the resource provider the scientific service request documents for all on-demand services, (2) when all scientific services are ready, invoking them in the specified order with the required input, and (3) in the end, sending to the resource provider the release request for all scientific services.

The SoapUI [50] application provides a GUI that can be of valuable help; after reading the WSDL document associated to a web service, it automatically lays out the structure of the XML input document leaving the user with the simpler task of filling in the data. SoapUI can also send the document to the given web service or workflow.

6.3. Scientific Services. The core of the scientific application is a modular web service, *Weka Data Mining*, which exposes

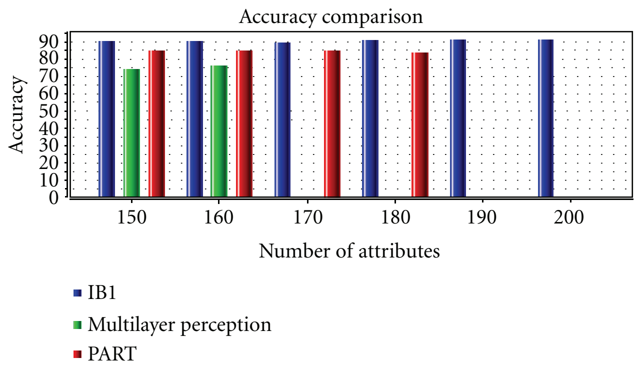

Figure 7: Snapshot of the bar chart generated by the visualization service.

a number of operations implementing basic data mining algorithms such as the following:

(i) feature (attribute) selection and ranking, for example, sort features according to some criteria;

(ii) dataset filtering, for example, remove all but the top-ranked features from a dataset;

(iii) model building, for example, train a classifier using a reference training set;

(iv) model testing, for example, calculate model accuracy using an independent test set.

The web service is built on top of the powerful open-source Weka library [51]. Data mining tasks may be long running, and the service exposes each operation both in synchronous (request-response) and asynchronous (one-way request and notify) form. The service is capable of retrieving input data, which may be large, via HTTP from a given URL, that is, from web server or storage manager; at the same time output data can be saved via HTTP to a given URL, that is, again to a storage manager. The service supports the processing of concurrent requests thus allowing for parallel flows of execution in the invoking workflow.

In addition, we have developed a visualization web service, *Chart*, which is a simple bar chart viewer based on the open-source JFreeChart library [52]; special attention has been paid to properly manage concurrent invocations resulting from parallel workflow execution as described in the next subsection. Figure 7 shows an example of chart generated by the service.

The scientific service images are available from the AMIR.

6.4. Workflows. The workflow developed for this case study is, in fact, a nested workflow. According to its definition, a BPEL process is also a web service, and it is perfectly legal to invoke a process from within another one. The invoked process, or *core process*, orchestrates a single iterative computation which trains a classifier and calculates its accuracy for an increasing number of attributes, as described before. The invoking process, or *wrapper process*, manages the concurrent execution of a variable number of core processes and links to an external visualization service, running on the user desktop, for monitoring the real-time execution of these processes. In particular, the wrapper process waits for the user to launch the visualization service before invoking the core processes. The core process iteratively invokes the different operations exposed by the Weka Data Mining service to build the classification model and to evaluate its accuracy and invokes the visualization service with the accuracy result; the latter accordingly updates the cumulative accuracy bar chart. A snapshot of the bar chart taken during execution, and shown in Figure 7, illustrates some points:

(i) colored bars represent accuracy; bars of the same color are associated to the same type of data mining model and are calculated by the same core workflow;

(ii) groups of contiguous bars of different colors represent the same iteration, that is, the same number of attributes, and are calculated by different core workflows;

(iii) missing bars witness that the execution of core workflows is asynchronous, that different data mining models have different build times, and/or that underlying resources have different speed.

In addition, missing bars may also show that a problem has occurred with one of the core workflows.

Workflow images are available from the AMIR.

6.5. Running Experiments. The general scenario for running an experiment can be synthesized by the following list of user activities:

(1) choose an experiment, that is, the workflow that describes it (an URL pointing to the workflow archive); in case of a new experiment, prepare the workflow image and upload it to the AMIR;

(2) prepare the experiment inputs, that is, make data files available as URLs;

(3) prepare and submit to the resource provider the XML document describing the workflow engine request;

(4) wait for the system to provision the workflow engine and to deploy the workflow;

(5) prepare and submit the workflow input document to the workflow engine;

(6) start the external helper services, if any, as needed, for example, the visualization service;

(7) wait for workflow execution: this typically involves the provisioning of the resources needed by the scientific services and their execution as orchestrated by the workflow.

If the workflow is linked to a visualization service, its progress can be followed graphically (Figure 7). Alternatively, the monitor service can be queried for resource and scientific service status. In addition, a monitoring application can be

```
<classification>
  <trainingDataset>http://www.dsf.unica.it/~andrea/train.arff</trainingDataset>
  <testDataset>http://www.dsf.unica.it/~andrea/test.arff</testDataset>
  <applyDataset>http://www.dsf.unica.it/~andrea/test.arff</applyDataset>
  <classifierName>weka.classifiers.functions.MultilayerPerceptron</classifierName>
  <attributeNumberMin>10</attributeNumberMin>
  <attributeNumberMax>200</attributeNumberMax>
  <attributeNumberInc>10</attributeNumberInc>
  <rankerName>none</rankerName>
  <seriesLabel>MultilayerPerceptron</seriesLabel>
</classification>
```

Algorithm 2: XML fragment of the workflow input document (classification input).

registered with the resource provider to receive and view all resource status notifications and message flows.

Algorithm 2 shows a fragment of the workflow input document which describes the input to the Weka Data Mining service (datasets, classifier, and number of attributes), needed by the core process. In addition, the XML fragment shown in Algorithm 1 represents the scientific service request needed by the wrapper process to obtain a resource to run the Weka Data Mining service. Both XML fragments are part of the same workflow input document.

In the experiment related to Figure 7, the wrapper process enacts three core processes working with different classifiers: nearest neighbors (IB1), neural network (multilayer Perceptron), and rule based (PART); the datasets, the chi-square attribute selection algorithm, and the iterations over an increasing number of attributes are the same. Three different VMs delivered by a private OpenStack cloud run the Weka Data Mining services (one per classifier).

7. Results and Discussion

The extensive tests performed and the results obtained for the case study presented in the previous section help us to address in some detail the following aspects in the next subsections: (1) performance and scalability, (2) shortcomings, and (3) open problems. The performance evaluation presented is by no means exhaustive: extensive comparison with existing and much more mature production systems is left for the future, after all the many tunings and modifications that the proposed prototype system implementation requires.

7.1. Performance and Scalability. To measure the time needed to start a managed computation, we have considered three main phases: (1) the request processing time is small compared to other times; (2) the resource setup time depends on WRMS/DIMS and is similar for the employed HPC and grid systems (on average about 1 minute) and higher for cloud systems, unless special tuning is performed [53] (on average 5–10 minutes which can be reduced to 1 minute); (3) workflow engine and scientific service startup times are comparable to the previous phase but depend on network speed for downloading images. VM performance might be an issue, so we have compared execution times using PM and VM both for workflow engines and scientific services and found no substantial differences; our tests require relevant CPU and network I/O activities and no heavy local disk I/O which may be one of the slowness reasons with VM [53].

The test workflow described above has been enacted with the following parameters: the wrapper process starts 10 concurrent instances of the core process, each invoking its own scientific service during 20 iterations; scientific services have been executed both on homogeneous resources and on a mix of geographically distributed heterogeneous infrastructures such as Cybersar [54] and FutureGrid [55]. The generated data flow is the following: 2000 Soap messages exchanged through the proxy service, 600 data files downloaded from web repositories (approximately 4.5 GB of data), 800 write operations and 1000 read operations to/from storage manage service. The execution time varies with the resource hardware but also with the available network bandwidth between scientific services, data repositories, and storage manager.

Many concurrent enactments of the test workflow can be used to assess system scalability: with the available resources, 10 concurrent workflows were enacted (i.e., 10 workflow engines on cloud and 100 scientific services half on cloud and half on HPC/grid) without any particular problem except for some transient failures in providing the requested resources occurred with some WRMS/DIMS (mostly Eucalyptus). From the tests performed, we expect that the system can manage a number of concurrent scientific services up to 1000 or more if the proxy service is replicated as discussed later on.

A key point in workflow engine and overall system scalability and performance is the management of the data flow for data intensive computations. Many workflow engines (including BPEL) act as brokers for all message exchanges between the scientific services participating in the workflow, but embedding large datasets into such messages is not efficient and can lead to a collapse of both the engine and the proxy service. Rather, web repositories and the storage manager, can be used, and datasets can be indirectly passed to scientific services as URL references; in this way, each scientific service can directly and efficiently read/write data from/to repositories and the storage manager, thus minimizing transfers and improving scalability and performance.

7.2. Shortcomings. A number of problems are related to the BPEL language and the engine implementations. The BPEL specifications [42] do not allow multiple receive activities in the same scope with the same port type and operation. This is an annoying limitation which may be overcome by duplicating the operations with different names. Different BPEL engines miss some of the functionality dictated by the standard or implement nonstandard features. An example is variable initialization, required by some engines and automatically performed by others; another is *partnerLink* assignment whose syntax can be different depending on the engine. In ODE, web service functionality is built on top of Apache Axis2 [56], the older Apache WS framework, and many advanced features are missing, for example, WS-Security.

Another point that deserves some attention is the disposal of resources when they are no longer needed. In principle, the resource life cycle, that is, provisioning, usage, and disposal, can be managed through the workflow if the correct support is provided at the scientific service level. If no error occurs, things go as expected, but if only the workflow fails before disposal, a bunch of resources may be left around reserved, unused, and in some cases charged for. In addition, a failure with a resource or scientific service may leave a workflow indefinitely waiting for it; the list of possible errors is long. BPEL fault and compensation handlers may help in designing better error resilience but can do nothing if the workflow engine or its VM crashes. The infrastructure services (resource provider, DIMSS, and WRMS) must then be charged of the extra responsibility of resource disposal when something goes wrong, for example, by periodically checking resource status for failures or long inactivity periods or other abnormal behavior.

7.3. Open Problems. The model presented in this work represents an attempt aimed at experimenting possible ways to cope with the general and complex problem of the integration of heterogeneous resources from multiple providers as a cohesive aggregate and unavoidably has a number of limitations and aspects not (adequately) covered; among others we have the following.

(i) Scalability with the number of workflow instances and scientific services can be an issue for the proxy service since it mediates most message exchanges; to account for this, more than one proxy can register with the resource provider, which assigns one of the available proxies to each scientific process and service, thus ensuring proper load balancing. Resource provider scalability has not been assessed except for the consideration that resource provider and all related services can be replicated allowing for static load balancing (the same WRMS/DIMS can be shared by different resource providers). The storage manager service can be replicated, too, but assignment to a scientific process or service should be based on some kind of network distance metric.

(ii) Redundancy of the resource provider service could be arguably obtained by means of standard high-availability techniques [57] if persistent data is stored into independent and replicated DBMS; in addition, the use of multiple proxy and storage manager services, as suggested above, should also help to guarantee the necessary redundancy.

(iii) Application recovery in case of network or other failures has not been taken into consideration; the simple solution of automatically reenacting failed workflows may work in case of transient failures but will not work in case of structural failures (e.g., when a computation exceeds the memory limit of a resource).

(iv) The need for a meta-scheduler has not been investigated, since WRMS and DIMS provide their own schedulers; if needed, such a meta-scheduler should be able to manage resource requests with very different requirements and resource pools (WRMS and DIMS) that change over time and over users.

Another important point concerns existing applications, domain-specific and resource-specific tools; many VOs have developed their own domain-specific, workflow scheduling systems which provide extensive monitoring, scheduling, data access and replication, and user priority management capabilities. We expect that users may be concerned by the problems/efforts required to use or even reimplement such applications and tools in a new environment [58]. According to what we said in the introduction, however, our proposal is not meant to "replace" but rather to "complement," so we must be pragmatic: it makes no sense to bother users which are happy with their applications, tools, and resources. Many "conventional" users will simply have no benefits from dynamic on-demand provisioning of resources, but many others may be attracted by this on one hand and by the advantages offered by the SOA approach on the other hand and may be interested in creating new or better applications using existing ones as building blocks; in most cases, it will not be necessary to reimplement applications and tools but simply wrap them with an SOA interface.

8. Conclusion

In this paper, we have presented a unified model for the dynamic on-demand provisioning of computational resources from multiple providers, based on the convergence and integration of different computing paradigms and infrastructures, and leveraging SOA technologies. Heterogeneous distributed resources are presented to their users with the illusion of a cohesive aggregate of resources, accessible through a well-defined standards-based software interface.

In addition, we have explored the adoption of standard unmodified BPEL language and existing engines for workflow description and enactment. Indeed, BPEL is not extremely popular in the workflow community, so we were interested in evaluating (also in practice) its advantages/limitations given its big merit of being an open standard (with special emphasis in relation to SOA) unlike most of the other languages employed in existing workflow systems and also to rouse new interest about it in the workflow community. At the same time, the presented model is intentionally flexible enough to allow us to employ other workflow systems such as Pegasus, Kepler, or Taverna; in particular, we plan to explore Taverna, thanks to the standalone "Taverna Server" component released by the Taverna team and which should be easily integrated with our proposed system.

In our opinion, the model benefits from a number of qualifying and distinctive features; it (1) is founded on firm SOA principles; (2) integrates different computing paradigms, systems, and infrastructures; (3) is open to further integrations and extensions both in terms of new resource management systems (WRMS and DIMS) and workflow management systems; (4) takes advantage from the integration and reuse of open-source tools and systems; (5) includes built-in scalability for some of the components; (6) promotes the porting and the development of new applications exposed as web services.

The effectiveness of the model has been assessed through a working implementation, which has shown its capabilities but also a number of open problems. In addition, the discussion of a case study in the bioinformatics field has hopefully given some hints on the practical possibilities of the proposed approach.

As discussed in Section 7, workflow management poses a number of problems. As well, another consideration is worth mentioning here, that is, the fact that BPEL workflow design is not straightforward even using the available graphical editors. Peeking at the business domain, the potential of the business process model and notation (BPMN) probably deserves some investigation in the future, given the capability of generating BPEL workflows directly from BPMN models.

In addition, we plan to extend the presented implementation (1) with the inclusion of new WRMS/DIMS and (2) with the integration of workflow engines to include both BPEL-based tools like Orchestra and tools like Taverna, which is not using BPEL.

Acknowledgments

The author acknowledges the Cybersar Consortium for the use of its computing facilities. This material is based upon work supported in part by the National Science Foundation under Grant no. 0910812 to Indiana University for "FutureGrid: An Experimental, High-Performance Grid Test-bed." Partners in the FutureGrid Project include University of Chicago, University of Florida, San Diego Supercomputer Center-UC San Diego, University of Southern California, University of Texas at Austin, University of Tennessee at Knoxville, University of Virginia, Purdue University, and Technische University Dresden.

References

[1] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-Science: an overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009.

[2] E. Elmroth, F. Hernández, and J. Tordsson, "Three fundamental dimensions of scientific workflow interoperability: model of computation, language, and execution environment," *Future Generation Computer Systems*, vol. 26, no. 2, pp. 245–256, 2010.

[3] T. McPhillips, S. Bowers, D. Zinn, and B. Ludäscher, "Scientific workflow design for mere mortals," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 541–551, 2009.

[4] A. Akram, D. Meredith, and R. Allan, "Evaluation of BPEL to scientific workflows," in *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID '06)*, pp. 269–272, IEEE Computer Society, May 2006.

[5] A. Bosin, N. Dessì, and B. Pes, "Extending the SOA paradigm to e-Science environments," *Future Generation Computer Systems*, vol. 27, no. 1, pp. 20–31, 2011.

[6] G. Mateescu, W. Gentzsch, and C. J. Ribbens, "Hybrid Computing-Where HPC meets grid and Cloud Computing," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 440–453, 2011.

[7] G. Fox and D. Gannon, "A survey of the role and use of web services and service oriented architectures in scientific/technical Grids," Tech. Rep. 08/2006, Indiana University, Bloomington, Ind, USA, 2006.

[8] I. Taylor, M. Shields, I. Wang, and A. Harrison, "Visual Grid workflow in Triana," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 153–169, 2005.

[9] D. Churches, G. Gombas, A. Harrison et al., "Programming scientific and distributed workflow with Triana services," *Concurrency Computation Practice and Experience*, vol. 18, no. 10, pp. 1021–1037, 2006.

[10] D. D. Pennington, D. Higgins, A. Townsend Peterson, M. B. Jones, B. Ludäscher, and S. Bowers, "Ecological Niche modeling using the Kepler workflow system," in *Workflows for eScience: Scientific Workflow for Grids*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds., pp. 91–108, Springer, Berlin, Germany, 2007.

[11] E. Deelman, G. Singh, M. H. Su et al., "Pegasus: a framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[12] T. Fahringer, R. Prodan, and R. Duan, "ASKALON: a development and Grid computing environment for scientific workflows," in *Workflows for eScience: Scientific Workflow for Grids*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds., pp. 450–471, Springer, Berlin, Germany, 2007.

[13] T. Oinn, M. Addis, J. Ferris et al., "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.

[14] T. Oinn, P. Li, D. Kell et al., "Taverna/myGrid: aligning a workflow system with the life sciences community," in *Workflows for eScience: Scientific Workflow for Grids*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds., pp. 300–319, Springer, Berlin, Germany, 2007.

[15] T. Dörnemann, E. Juhnke, and B. Freisleben, "On-demand resource provisioning for BPEL workflows using amazon's elastic compute cloud," in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*

(CCGRID '09), pp. 140–147, IEEE Computer Society, May 2009.

[16] R. Y. Ma, Y. W. Wu, X. X. Meng, S. J. Liu, and L. Pan, "Grid-enabled workflow management system based on Bpel," *International Journal of High Performance Computing Applications*, vol. 22, no. 3, pp. 238–249, 2008.

[17] I. Brandic, S. Pllana, and S. Benkner, "High-level composition of QoS-aware grid workflows: an approach that considers location affinity," in *Proceedings of the Workshop on Workflows in Support of Large-Scale Science (WORKS '06)*, pp. 1–10, Paris, France, June 2006.

[18] A. Slominski, "Adapting BPEL to scientific workflows," in *Workflows for eScience: Scientific Workflow for Grids*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds., pp. 208–226, Springer, Berlin, Germany, 2007.

[19] F. Leymann, "Choreography for the Grid: towards fitting BPEL to the resource framework," *Concurrency Computation Practice and Experience*, vol. 18, no. 10, pp. 1201–1217, 2006.

[20] K. M. Chao, M. Younas, N. Griffiths, I. Awan, R. Anane, and C. F. Tsai, "Analysis of grid service composition with BPEL4WS," in *Proceedings of the18th International Conference on Advanced Information Networking and Applications (AINA '04)*, pp. 284–289, March 2004.

[21] T. Dörnemann, T. Friese, S. Herdt, E. Juhnke, and B. Freisleben, "Grid workflow modeling using Grid-specific BPEL extensions," in *Proceedings of German e-Science Conference*, Baden-Baden, Germany, 2007.

[22] W. Emmerich, B. Butchart, L. Chen, B. Wassermann, and S. L. Price, "Grid service orchestration using the Business Process Execution Language (BPEL)," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 283–304, 2005.

[23] M. A. Murphy and S. Goasguen, "Virtual Organization Clusters: self-provisioned clouds on the grid," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1271–1281, 2010.

[24] C. Vázquez, E. Huedo, R. S. Montero, and I. M. Llorente, "On the use of clouds for grid resource provisioning," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 600–605, 2011.

[25] A. Bosin, N. Dessì, M. Bairappan, and B. Pes, "A SOA-based environment supporting collaborative experiments in E-science," *International Journal of Web Portals*, vol. 3, no. 3, pp. 12–26, 2011.

[26] I. Foster, K. Kesselman, J. M. Nick, and S. Tuecke, "The physiology of the grid—an open grid services architecture for distributed systems integration globus alliance," 2002, http://www.globus.org/alliance/publications/papers/ogsa.pdf.

[27] P. Tröger, "DRMAAv2—An Introduction," 2011, http://www.drmaa.org/drmaav2-ogf33.pdf.

[28] C. Aiftimiei, P. Andreetto, S. Bertocco et al., "Design and implementation of the gLite CREAM job management service," *Future Generation Computer Systems*, vol. 26, no. 4, pp. 654–667, 2010.

[29] AWS, "Amazon Web Services: Amazon Elastic Compute Cloud—API Reference," 2011, http://awsdocs.s3.amazonaws.com/EC2/latest/ec2-api.pdf.

[30] AWS, "Amazon Web Services: Amazon Simple Storage Service—API Reference," 2006, http://awsdocs.s3.amazonaws.com/S3/latest/s3-api.pdf.

[31] R. Nyren, A. Edmonds, A. Papaspyrou, and T. Metsch, "Open Cloud Computing Interface—Core," 2011, http://ogf.org/documents/GFD.183.pdf.

[32] D. Nurmi, R. Wolski, C. Grzegorczyk et al., "The eucalyptus open-source cloud-computing system," in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09)*, pp. 124–131, IEEE Computer Society, May 2009.

[33] I. Foster, K. Keahey, C. Kesselman et al., "Embedding community-specific resource managers in general-purpose grid infrastructure," Tech. Rep. ANL/MCS-P1318-0106, Argonne National Laboratory, Lemont, Ill, USA, 2006.

[34] K. Pepple, *Deploying OpenStack*, O'Reilly Media, Sebastopol, Calif, USA, 2011.

[35] E. Laure, S. M. Fisher, A. Frohner et al., "Programming the Grid with gLite," *Computational Methods in Science and Technology*, vol. 12, no. 1, pp. 33–45, 2006.

[36] LSF, "Platform Load Sharing Facility," 2005, http://www.platform.com/workload-management/high-performance-computing.

[37] SGE, "Oracle Grid Engine," 2009, http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html.

[38] O. V. Sukhoroslov, "JLite: a lightweight Java API for gLite," 2009, http://jlite.googlecode.com/files/jLite.pdf.

[39] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.

[40] JCLOUDS, "Jclouds multi-cloud library," 2011, http://code.google.com/p/jclouds.

[41] Libvirt, "The virtualization API," 2011, http://libvirt.org.

[42] OASIS, "Web Services Business Process Execution Language Version 2.0," 2007, http://docs.oasis-open.org/wsbpel/2.0/ws-bpel-v2.0.html.

[43] W3C, "Web Services Description Language 1.1," 2001, http://www.w3.org/TR/wsd.

[44] ORACLE, "Oracle BPEL Process Manager," 2011, http://www.oracle.com/technetwork/middleware/bpel/overview/index.html.

[45] AVOS, "ActiveVOS platform," 2011, http://www.activevos.com.

[46] OW2, "Orchestra User Guide," 2011, http://download.forge.objectweb.org/orchestra/Orchestra-4.9.0-UserGuide.pdf.

[47] ODE, "Apache Orchestration Director Engine," 2011, http://ode.apache.org.

[48] A. Bosin, N. Dessì, and B. Pes, "A cost-sensitive approach to feature selection in micro-array data classification," in *Proceedings of the 7th international workshop on Fuzzy Logic and Applications: Applications of Fuzzy Sets Theory (WILF '07)*, vol. 4578 of *Lecture Notes in Computer Science*, pp. 571–579, Springer, 2007.

[49] E.-J. Yeoh, M. E. Ross, S. A. Shurtleff et al., "Classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling," *Cancer Cell*, vol. 1, no. 2, pp. 133–143, 2002.

[50] SOAPUI, "Eviware SoapUI," 2011, http://www.soapui.org.

[51] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[52] D. Gilbert, "JFreeChart Java chart library," 2011, http://www.jfree.org/jfreechart.

[53] A. Bosin, M. Dessalvi, G. M. Mereu, and G. Serra, "Enhancing eucalyptus community cloud," *Intelligent Information Management*, vol. 3, no. 4, pp. 52–59, 2012.

[54] Cybersar, "Cybersar consortium for supercomputing, computational modeling and management of large databases," 2006, http://www.cybersar.com.

[55] FutureGrid, "FutureGrid: a distributed testbed for Clouds, Grids, and HPC," 2009, https://portal.futuregrid.org.

[56] AXIS2, "Apache Axis2," 2011, http://axis.apache.org/axis2/java/core/.

[57] LHA, "Linux-HA," 2011, http://www.linux-ha.org/wiki/Main_Page.

[58] W. Gentzsch, "Porting applications to grids and clouds," *International Journal of Grid and High Performance Computing*, vol. 1, no. 1, pp. 55–77, 2009.