Elastic Service Management in Computational Clouds

Clovis Chapman*, Wolfgang Emmerich*,Fermín Galán Márquez †,Stuart Clayman‡, Alex Galis‡
*Department of Computer Science, UCL, Malet Place, London, UK {c.chapman, w.emmerich}@cs.ucl.ac.uk
†Telefónica I+D, Emilio Vargas 6, Madrid, Spain fermin@tid.es
‡Department of Electrical Engineering, UCL, Gower Street, London, UK {s.clayman, a.galis}@ee.ucl.ac.uk

***Abstract*— Cloud computing [1], [2] is rapidly changing the landscape of traditional IT service provisioning. It presents service providers with the potential to significantly reduce initial capital investment into hardware by removing costs associated with the deployment and management of hardware resources and enabling them to lease infrastructure resources *on demand* from a virtually unlimited pool. This introduces a great degree of flexibility, permitting providers to pay only for the actual resources used per unit time on a "pay as you go" basis and enabling them to optimise their IT investment whilst improving the overall availability and scale of their services. In this manner, the need for overprovisioning of services to meet potential peaks in demand can be considerably reduced in favour of driving resource allocations dynamically according to the overall application workload. This however requires means of defining rules by which the service should scale and mechanisms must be provided by the infrastructure to monitor this state and enforce the rules accordingly.**

We discuss in this paper an elastic service definition language and service management that are developed in the context of the RESERVOIR project to facilitate dynamic service provisioning in clouds. This language builds on the Open Virtualisation Format, a DMTF standard for the packaging and deployment of virtual services, and introduces new abstractions to support service elasticity. In addition, we detail the functional requirements that the the cloud infrastructure must meet to handle these abstractions.

I. Introduction

Cloud Computing has rapidly established itself as a paradigm for the provisioning of IT services as utilities over the Internet. With the boom of Web 2.0, and the development of virtualisation technologies, a market opportunity has emerged in de-coupling the provisioning of physical hardware from the application level services they provide.

Companies supplying services over the Internet often need to over-provision their servers by as much as 500 percent to handle potential peaks in demand or cater for very high growth rates [3]. In addition, the deployment of a service is expensive and time consuming; machines must be set up with an operating system, configured with network, storage capabilities, and applications must be deployed. Entry costs for a service provider can hence be very steep, often serving as a barrier to the entrance of SMEs.

By harnessing virtualisation and grid technologies, Infrastructure-as-a-service (IAAS) Clouds [1] remove such barriers by providing a foundation for the hosting and deployment of web-based services, theoretically relieving providers from the responsibility of provisioning the computational resources needed to support these services. Virtualisation technologies introduce a layer of abstraction between physical hardware resources and virtualised operating systems, enabling multiple systems to share single physical servers, the suspension and resumption of execution of these systems, and even their live migration from one physical server to another.

Such capabilities enable the creation of a cost-effective service-based online economy where resources and services can be transparently and flexibly provisioned and managed as utilities. Service providers can lease resources on a pay-as-you-go basis and dynamically adjust their resource consumption to accommodate increased loads.

But providing support for such activities requires a clear understanding of the requirements of service providers to be communicated to the cloud infrastructure. Indeed software systems may be composed of numerous loosely coupled components whose capacity requirements may need to evolve individually to accommodate varying workloads. Further more dependencies and constraints regarding deployment and placement may exist, which must be taken into account when migrating services from one host to another in a cloud. The service provider must hence be given the tools to control the provisioning process not just at deployment time, but throughout the lifetime of a service. This requires specific language abstractions that will be used to detail the provider's requirements in the form of a *service manifest*. In addition, the underlying framework must present functional capabilities that will enable these requirements to be enforced at run-time.

In this paper, we examine how we have handled the problem of service definition in the RESERVOIR project, a European Union FP7 funded project involving 13 academic and commercial organisations in Europe to develop an Open Cloud Computing Infrastructure building on open standards. The principal contribution of this paper is a discussion of the problem of dynamic resource allocation in clouds, and the solutions we have proposed to expose application-level state to service management components in order to ensure that appropriate adjustment decisions are made in a timely manner to meet service level objectives. The paper details the structure of our service definition language and the syntactic elements that we have adopted to support notions such as elasticity.

This research has been partly funded by the RESERVOIR EU FP7 Project. http://www.reservoir-fp7.eu/.

The paper is structured as follows: in Section II, we explore the background to this research and give an overview of the RESERVOIR architecture. In Section III, we examine key requirements of a service definition language and detail our proposed abstractions to handle service elasticity. In order to demonstrate the usability of our approach we detail in Section IV two use cases explored in the context of RESERVOIR, an SAP Enterprise Resource Planning system and a grid based computational chemistry application. Finally we conclude with a discussion of related work in Section V and an overview of future work.

II. Background

Virtualisation technology, such as Xen [4] and VMWare [5], is a key enabler of cloud computing. Whereas operating systems would traditionally be responsible for managing the allocation of physical resources, such as CPU time, main memory, disk space and network bandwidth to applications, virtualisation allows a hypervisor to run on top of physical hardware and allocate resources to isolated execution environments known as *virtual machines*, which run their own individual virtualised operating system. Hypervisors manage the execution of these operating systems, booting, suspending or shutting down systems as required. Some hypervisors even support replication and migration of virtual machines without interruption.

As the unit cost of operating a server in a large server farm is lower than in small data centres, such technologies have been exploited to form the basis of cloud computing. Commercial services, such as Amazon's Elastic Compute Cloud (EC2) [6] or IBM's Blue Cloud [7], have been built on aggregating physical resources and providing management technologies enabling the co-ordinated allocation of resources to virtual machines across numerous hypervisors.

However single-provider proprietary clouds are by nature limited in their flexibility and scale. As the number of users and online services hosted increase considerably, it is not possible to provide a seemingly endless compute utility without partnering with other infrastructure providers to achieve the appropriate economy of scale.

The *Resources and Services Virtualisation without Barriers* (RESERVOIR) project aims to address these issues by building on open standards and existing virtualisation products to define a reference model and architecture to support the federation and interoperability of computational clouds. RESERVOIR aims to satisfy the vision of service oriented computing by distinguishing the needs of *Service Providers* and *Infrastructure Providers*. Service providers understand the operation of particular businesses and will offer suitable *Service* applications, and will lease resources in order to host these services. Infrastructure providers on the other hand will own these resources and provide the mechanisms to deploy and manage self contained services. They may themselves lease additional resources from their peers to form a seemingly infinite pool.

The overall RESERVOIR architecture is illustrated in Figure 1. In the context of this architecture, a *service* is a set of software components, each with it's own software stack (OS, middleware, application, configuration and data) in the form of one or more virtualised images. Each component executes in a dedicated *Virtual Execution Environment* (VEE) and these may present different requirements. Service providers deploy applications by communicating these requirements to a single infrastructure provider.

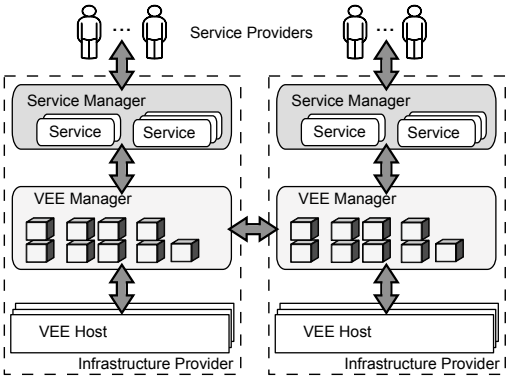

Fig. 1. RESERVOIR architecture

The layers of the RESERVOIR architecture are briefly described as follows.

- **The Virtual Execution Environment Host (VEEH)** is the lowest layer of the architecture. It represents a virtualised resource that can host one or more VEEs. It provides a standard interface to different hypervisors, translating requests from upper layers into commands specific to the virtualisation platform.
- **The Virtual Execution Environment Manager (VEEM)** is responsible for distributing services amongst a collection of VEE Hosts, controlling the creation, activation and de-activation of VEEs, migration and resizing. It will place VEEs generally according to policies defined by the infrastructure provider, such as security and cost. In addition it is this layer that is responsible for the federation of remote sites, through cross site interactions with different VEEMs.
- **The Service Manager** acts as the primary interface of the service provider. It will be responsible for the instantiation of the service application, managing the overall collection of components on behalf of the service provider and monitoring the execution of the application in order to drive potential capacity adjustments. This may require adding or removing new instances of service components and/or changing the requirements of a service dynamically. The Service Manager also performs other service management tasks, such as accounting and billing.

A full implementation of the architecture is under development (www.reservoir-fp7.eu), which includes the VEEM open source component OpenNebula, available from OpenNebula.org.

It is the service manager layer that we are concerned with in this paper. In order for the service manager to deploy and monitor an application consisting of several components, these must be bound by a description of the overall software system architecture and the requirements of the underlying components, that we refer to as the *Service Manifest*. The

manifest will describe issues such as grouping instructions, topology, etc. in addition to the functional and capacity needs of each component.

We aim in this paper to identify the abstractions that should be provided by a language used to specify a manifest, particularly with regards to elasticity, and define the language syntax by building on existing standards. We will also examine how these syntactic elements relate to the underlying infrastructure.

III. Elastic Service Definition

A. Requirements

In this section, we break down the core requirements that a service provider must describe in order to ensure the correct provisioning of a service on a cloud computing infrastructure. This enables us to determine the essential syntactic elements that the service definition language should present. In the following sections we will then identify how these requirements are met by existing standards, in particular the Open Virtualisation Format, the extensions required, and provide an overview of our approach.

1) Service Specification: A software application may be composed of several loosely coupled components must to be managed jointly. We can consider a standard 3 tier web application, consisting of a single database, web servers and a load balancer. Whilst independent, they present dependencies and constraints that must be respected by the cloud infrastructure. The service provider must hence be able to specify:

- **SD1 Software architecture:** The overall structure of the service, and characteristics of each component.
- **SD2 Hardware:** The physical hardware requirements of individual components (e.g. CPU, memory, etc.).
- **SD3 Software environment:** An explicit specification of the software required to be run on the component, such as the OS and kernel, or a reference to a pre-configured virtual disk image with the appropriate application level components.
- **SD4 Network topology:** The physical interconnections of components and external interfaces required.

2) Elasticity Specification: In order to be able to automate the scaling of applications to meet variations in workload, the service provider must be able to describe the conditions within which this scaling takes place. Referring to the web application example, it may be necessary to increase the number of web servers available, though the load balancer would continue to serve as a single entry point.

- **SD5 Elasticity rules:** The service provider can define conditions related to the state and operation of the service, and associated actions to follow should these conditions be met. The actions may involve the resizing of a specific service component (e.g. increase/decrease allocated memory) or the deployment/undeployment of specific service instances.
- **SD6 Key performance indicators:** Providing support for elasticity requires the definition of indicators of performance that will be monitored to trigger adjustments accordingly. These may be infrastructure level indicators such as current disk use, but application level performance indicators may prove necessary to maximise the optimisation and response.

3) Virtualisation: The underlying use of virtualisation technologies requires a specification to be able to indicate constraints that may exist with regards to the deployment of services on hypervisors. In particular:

- **SD7 Migration constraints:** The service provider must specify which components can be moved from one host to another in a transparent manner and the optimum conditions that have to be met to minimise disruption (e.g. maximum unavailability).
- **SD8 Portability:** The overall service may be tied to specific hypervisor technology, or virtual machine images may require conversion to be migrated from one host to another. Catering for a heterogeneous hosting environment implies providing the means to specify such constraints.

4) Component dependencies: The inter-relationships between components may not favour simultaneous deployment of all components; some components may be required to be available before others and configuration data may be dependent on specific deployment instances. We identify the following:

- **SD9 Deployment:** There may exist a need to specify the order in which components are deployed (e.g. the database before the web server). In addition some configuration parameters may only be known at deployment time, such as the IP address of a specific component instance.
- **SD10 Undeployment:** Similarly, dependencies should be taken into account when shutting down particular components.

5) Component Customisation: The manifest should serve as a template for easily provisioning instances of particular components. Multiple instances of web servers for example may be created from a same basic template and virtual image and may require instance specific configuration data, such as dynamically allocated IP addresses. The manifest language must provide constructs to support the automatic generation of instance specific values, and deployment time customisation of images (**SD11 Customisation**).

6) Constraint Policy: Federation provides the ability to seamlessly deploy services across multiple physical and administrative domains, but doing so means allowing service providers to control the spread of the application by defining clear constraints on the distribution of services across sites (**SD12 Location constraints.**). Placement constraints may be of a technical nature (e.g. deploy certain components on a same host to minimise latency) or administrative (avoid untrusted locations).

7) Non-functional Requirements: In addition, we may take into account several non-functional requirements. In particular, the manifest should be specified in a language that is platform independent (**SD13**), based on open standards (**SD14**) and generally be supported by existing implementations to ease development and adoption (**SD15**). It should also provide

abstractions to handle security and authentication (**SD16**).

B. Adopting open standards

In order to achieve the goal of interoperability for the federation of clouds it is important to examine existing standards that may be supported by the virtualisation community. These standards should enable the distribution of services across clouds and meet the requirements detailed above. We are not however solely concerned with interoperability but also with addressing issues for IaaS clouds that are not completely solved, in particular with regards to service elasticity.

Numerous software architecture description languages exist, though the Open Virtualisation Format (OVF) [8], a DMTF standard backed by VMWare and XenSource, is undoubtedly the most suited as a building block for our manifest language. OVF aims to offer a portable packaging mechanism for virtual appliances in a platform neutral way.

In previous work [9], we have discussed a number of extensions to the OVF syntax to support clouds, including attribute and section changes to incorporate support for service components IDs in elastics arrays, cross virtual machines reference, IP dynamic addresses and elasticity rules and bounds. However, we must review the implications of these language abstractions on the design of the RESERVOIR infrastructure, and identify the level of support required to ensure the correct provisioning of elastic services.

Indeed, there must exist a clear understanding of how we derive from the language used to express the requirements of the Service Provider a *management cycle*, which will consist of several actions being taken throughout the lifetime of a service to ensure a certain service quality being obtained. Using the RESERVOIR framework as a reference, and examining specifically issues related to dynamic capacity adjustment and service deployment, we refine our OVF-based language service definition language and syntax to incorporate these abstractions, focusing specifically on elasticity and application domain description.

We briefly describe OVF as follows: OVF allows multiple virtual machines to be packaged as a single entity containing an OVF descriptor, which will be in XML, any resources which may be referred to in the descriptor, such as virtual disk, ISO images, etc., and finally X.509 certificate files to ensure integrity and authenticity. It is the OVF descriptor that we will focus on here and will form the core syntax of the manifest we have described.

It is an XML document composed of three main parts: a description of the files included in the overall architecture (disks, ISO images, etc.), meta-data for all virtual machines included, and a description of the different virtual machine systems. The description is structured into various "Sections". Focusing primarily on the most relevant, the `<DiskSection>` describes virtual disks, `<NetworkSection>` provides information regarding logical networks, `<VirtualHardwareSection>` describes hardware resource requirements of components and `<StartupSection>` defines the virtual machine booting sequence.

In itself, OVF addresses requirements SDL1 to SDL4 by providing means of specifying various loosely coupled components, hardware and software requirements and network topologies as a single logical entity. In addition, it addresses SD9 and SD10 by providing means of describing the deployment and undeployment dependencies. The OVF specification also addresses the issue of customisation (SDL11) by describing a communication protocol between host and guest (the target virtual machine) that allows the deployment-time configuration of a virtual machine instance. The `<ProductSection>` of the descriptor allows various properties to be specified as key-value pairs. Though various transport mechanisms may be used to communicate these properties, the sole defined approach at time of writing, the "iso" transport, consists of dynamically generating CD/DVD images to be used as boot disks during the start-up process.

As an open standard, it also addresses requirements SDL13 to SDL15, being supported by a number of currently available tools to manipulate and create OVF packages, such as that provided by VMWare. Finally, SDL16 is met through the inclusion of X.509 certificates as previously described.

There are a number of requirements however that OVF in itself does not meet. In particular it is focused solely on initial deployment of fixed size services, and does not provide measures to handle potential changes in requirements over the lifetime of a service (SDL5 and SDL6). It also does not provide measures to handle migration across hosts (SDL7 and SDL8) nor does it provide any measure to deal with federated environments (SDL12).

Such language abstractions must hence be defined with respect to the expected capabilities of cloud infrastructures. This allows us to identify key functional characteristics that the infrastructure should present. In the context of this paper, we will focus on elasticity capabilities, refining and extending the OVF descriptor to form a service manifest suited to a dynamic cloud provisioning environment.

C. Abstract syntax of the Manifest Definition Language

We describe in this section the abstract syntax of the *Manifest Language* and the relationship between syntactic elements of the language and the underlying cloud infrastructure. The abstract syntax covers the core elements of the language and their accompanying attributes, and is modelled here using the *Essential Meta-Object Facility* (EMOF), an OMG standard part of the Model Driven Architecture initiative [10] for describing the structure of meta-data. EMOF models are very similar to UML class diagrams, in that they describe classes, the data they contain and their relationships, but are at a higher level of abstraction: they describe the constructs, rules and constraints of a model. As such, EMOF is typically used to define the syntax of languages. We describe this syntax with respect to the domain elements that model the operation of cloud infrastructure components.

Though outside of the scope of this paper, we can further express the behavioural semantics of the language as constraints between the abstract syntax and domain elements in the model denotational style to define semantics that we introduced in [11]. This provides a formalisation of these semantics, ensuring that we limit ambiguities when it comes to the interpretation of the manifest. This is crucial where

financial liabilities may exist; a formal understanding of the nature of the service being provided is required in order to ensure that the service is provisioned as expected by both parties, and in a way that both can evaluate to be correct, either through run-time monitoring capabilities or logs.

The core syntax relies upon, as previously stated, OVF. Incorporating the OVF standard provides the manifest language with a syntactic model for the expression of physical resource requirements and hardware configuration issues. We introduce new abstractions in the form of extensions to the standard rather than create new independent specifications, as doing so ensures continued compatibility with existing systems.

1) Application description language: The automated scaling of service capacity to support potential variations in load and demand can be implemented in numerous ways. Application providers may implement such scaling at the application level, relying on an exposed interface of the cloud computing infrastructure to issue specific reconfiguration requests when appropriate. Alternatively, they may have a desire to keep the application design free of infrastructure specific operations and opt instead to delegate such concerns to the infrastructure itself. With a sufficient level of transparency at the application level for workload conditions to be identified, and through the specification of clear rules associating these conditions with specific actions to undertake, the cloud can handle dynamic capacity adjustment on behalf of the service provider.

It is the latter approach that we have chosen to adopt in the context of RESERVOIR. By providing a syntax and framework for the definition and support of elasticity rules, we can ensure the dynamic management of a wide range of services with little to no modification for execution on a cloud. With respect to the syntax, we can identify the two following subsets of the language that would be required to describe such elasticity: service providers must first be able to describe the application state as a collection of *Key Performance Indicators* (KPIs), and the means via which they are obtained in the manifest. These will then serve as a basis for the formulation of the rules themselves, described in the following subsection.

Though it is possible to build elasticity rules purely based on infrastructure level performance indicators, this may prove limiting as will be detailed in the use cases. Indeed, the disk space, memory or CPU load may not accurately reflect the current needs of the application, nor suitably allow scaling to be anticipated.

Because we do not want the manifest language to be tied to any specific service architecture or design, it is necessary to decouple the KPI descriptions from the application domain via the use of an *Application Description Language* (ADL). This language will allow the description of services in terms of components, parameters and their monitoring requirements.

Alongside the syntactic requirements, a suitable monitoring framework must exist. A service provider is expected to expose parameters of interest through local *Monitoring Agents*, responsible for gathering suitable application level measurements and communicating these to the service management infrastructure. Though communication protocols with the underlying framework are outside the scope of the model, there must exist a correlation between the events generated by the monitors and the KPIs described in the manifest. This is modelled in Figure 2.

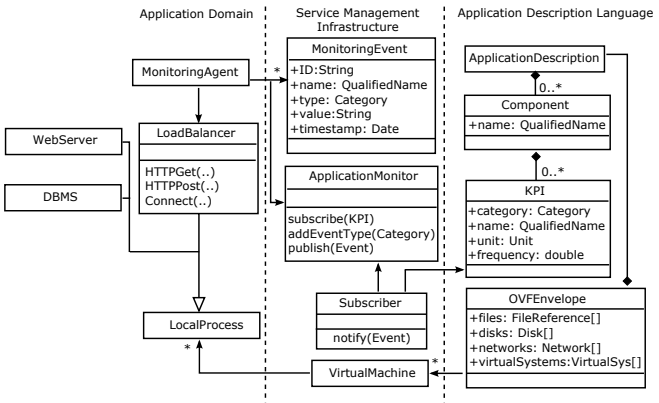

Fig. 2. Application Description Language

Using as an example a basic 3-tier web architecture, with load balancer, web server and database management system, the figure exemplifies the relationship between the ADL, the RESERVOIR application-level monitoring infrastructure, and the application domain. The syntax of the ADL consists of one or more named components, with a number of associated KPIs. These KPIs are identified using appropriate qualified names (e.g. `com.sap.webdispatcher.kpis.sessions`), that will allow the underlying infrastructure to identify corresponding events obtained from an application level monitor and forward these to subscribers responsible for enforcing elasticity rules.

We are concerned in this example with the number of simultaneous web sessions managed by the `LoadBalancer` component. We can conceive that this parameter may be used as a general indicator of load, and used to increase the number of deployed instances of web servers.

Our actual implementation of the RESERVOIR monitoring framework itself is fully described in [12]. Data sources, such as application level monitoring agents, encapsulate one or more probes responsible for the collection and publication of specific attributes and values. Communication between producers and consumers of data is divided between multiple communication planes: the control plane, to manage the execution of infrastructure level probes, such as turning them on and off and reconfiguration, the information plane, to facilitate the exchange of related meta-data, and the data plane, for the actual communication of KPI measurements. In order to minimise the number of connections established between end points a number of solutions are in place, including the use of IP multicasting, and intermediate data aggregation points, which will be responsible for collecting data packets and processing these if necessary to produce new performance indicators. This framework is well suited to the approach adopted here, providing means of retrieving KPIs published on the network from multiple sources including monitoring agents.

2) Elasticity Rules: With respect to the rule syntax, we adopt an *Event-Condition-Action* approach to rule specifica-

tion. Based on monitoring events obtained from the infrastructure, particular actions from the VEEM are to be requested when certain conditions relating to these events hold true. This requires the rules to be expressed with respect to the interface of the underlying VEEM and monitoring events.

A representation of the elasticity rules based on a general rule-base model and their relationship to monitoring events and the cloud infrastructure is illustrated in Figure 3. The syntax specifies conditions, based on monitoring events at the application layer or otherwise, which would lead to specified actions, based on a set of operations presented by the VEEM. The operations, modelled on the OpenNebula framework capabilities will involve the submission, shutdown, migration, reconfiguration, etc. of VMs and should be invoked within a particular time frame. The conditions are expressed using a collection of nested expressions and may involve numerical values, arithmetic and boolean operations, and values of monitoring elements obtained. The relationship between KPIs specified in the manifest and these events has been described in the previous section. The elasticity rules will be supervised by the cloud infrastructure at the Service Manager layer during the running of the software system and it is expected that a rule interpreter will receive events from the infrastructure or application monitors and trigger such operations accordingly. An example of a concrete OVF-based specification of an elasticity rule will be provided in Section IV.

It is worth briefly discussing the subject of time management. The service provider controls the timeliness of the response in multiple ways. Firstly the rate at which monitoring events are sent by the application level monitor is entirely dictated by the application and this should be carefully balanced against deployment or adjustment time to avoid duplicate responses. Secondly service providers can specify a time frame within which particular actions should take place, as described above. Finally, the current time can be introduced as a monitorable parameter if necessary.

Elasticity rules can be a powerful tool to express capacity constraints. This may be combined with predictive approaches for the underlying infrastructure to anticipate future allocation changes but in itself, an Event Condition Action model is highly intuitive and sufficiently expressive to handle the majority of service applications as will be seen in Section IV.

Our prototype implementation relies on an XML parser component to parse the rules and supply these to a rule engine component, which will check the rules periodically against monitoring records obtained from the network. Should the conditions hold, actions will be triggered accordingly.

IV. Use Cases

In this section, we delineate the use cases relied upon to identify the requirements specified in the previous section and whose scalability objectives we have used as a basis for the definition of elasticity rules. We present these scalability objectives and demonstrate how these would be formulated and met using the RESERVOIR framework.

We have relied on two cases, an enterprise grade system, the SAP Enterprise Resource Planning system, and an academic scientific grid application currently deployed at University College London. In both instances, manifests describing the overall architecture of the system were produced and these were deployed on the RESERVOIR infrastructure.

A. SAP ERP systems

SAP systems are used for a variety of business applications, such as Customer Relationship Management and Enterprise Resource Planning. They will typically consist of generic components customised by configuration used in combination with custom-coded elements. A high-level architecture of an SAP system is illustrated in Figure 4. SAP systems have a traditional multi-tiered software architecture with a relational database layer:

- **SAP Web dispatcher** The web dispatcher is responsible for the handling of requests and session management, and balancing workloads between multiple dialog instances.
- **Dialog Instance (DI):** The dialog instance is an application server responsible for handling business logic and generating HTTP-based dialogues that are shown in a browser. Several instances of the DI may be deployed to adapt to load.
- **Central Instance (CI):** The central instance provides central services such as application level locking, messaging and registration of DIs. Only a single instance of a CI will be deployed.
- **Database Management System:** A single DBMS will serve the SAP system.

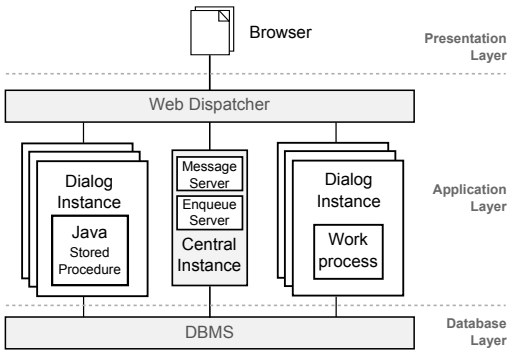

Fig. 4. SAP three-tiered architecture

To date SAP systems are hosted in substantial data centres, but organisations in the future might wish to deploy these in a compute cloud in order to avoid the significant capital investment. This however requires a number of architectural constraints to be obeyed by the cloud.

Though a complete manifest was produced to describe an SAP deployment on the RESERVOIR infrastructure, we can only briefly cover essential requirements. The manifest will specify a single template for each component alongside file references to disk images which will contain the appropriate software and configuration data. Components will have strongly differing hardware requirements with the DBMS having significantly higher storage requirements while the dialog instances will be more processor intensive. In addition, the

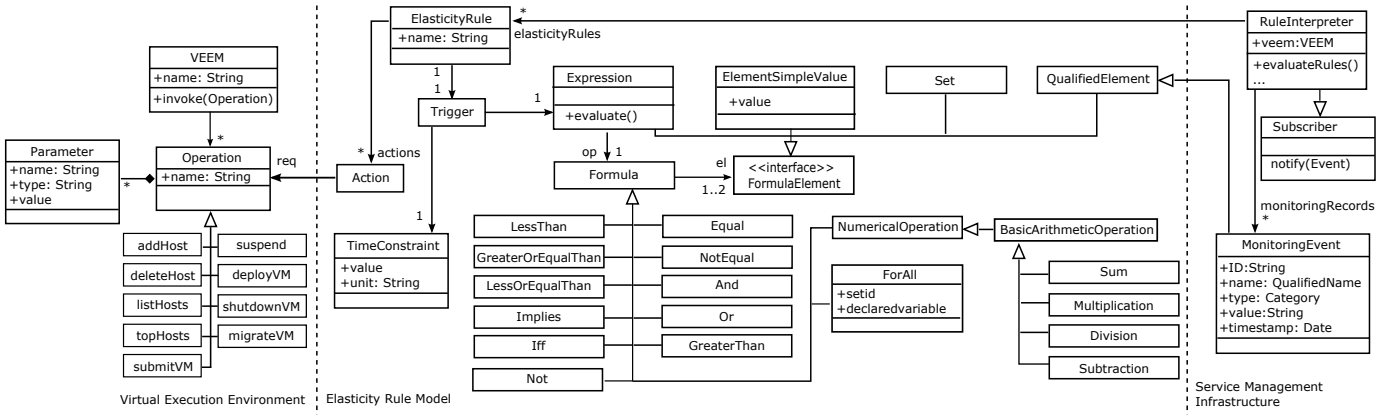

Fig. 3. Elasticity Rules Syntax

central instance will frequently make queries to the database and the Central Instance should be located on the same LAN subnet as the DBMS to minimise latency. Deployment dependencies exist, with the DBMS and central instance required to be active before other components.

In addition because we are generating potentially multiple instances of a component from a single disk image, replicas of the image must be customised to include both instance specific information and data obtained only at deployment time, such as the host name and IP of the CI.

With respect to service elasticity, there exists a direct proportional relationship between resource requirements and sessions. The number of simultaneous sessions will be used as a basis for scaling the number of dialog instances using elasticity rules. However, directly monitoring the traffic to and from the web dispatcher would be impossible as SAP uses proprietary protocols. The SAP system can nonetheless report the number of sessions provided an appropriate query is formulated. As such it is necessary to implement a monitoring agent to formulate that query and publish the answer onto the network. This may then be picked up by the service manager for the enforcement of elasticity rules.

An example of elasticity rule specified in XML as part of an OVF descriptor is as follows. Here we describe that a new dialog instance should be deployed for every 200 users up to a maximum of 5. A definition of the `kpis.totalUsers` should be described using the ADL, and the infrastructure will define a number of predefined KPIs such as the number of replicas of named components.

```
<elr:ElasticityRule name=''DI_Increase''>
 <elr:Trigger>
  <TimeConstraint unit=''ms''>5000</TimeConstraint>
   <elr:Expression>
    kpis.totalUsers / components.DI.replicas.amount > 200
    && components.DI.replicas.amount < 5
   </Expression>
 </Trigger>
 <elr:Action run=''deployVM(components.DI)''/>
<elr:ElasticityRule>
```

B. Scientific Grid Computing

Another use case we have tackled is the deployment on the RESERVOIR infrastructure of a grid based application responsible for the prediction of organic crystal structures from the chemical diagram [13]. The application operates according to a predefined workflow involving multiple web based services and Fortran programs. Up to 7200 executions of these programs may be required to run, as batch jobs, in both sequential and parallel form, to compute various subsets of the prediction. Web services are used to collect inputs from a user, coordinate the execution of the jobs, process and display results, and generally orchestrate the overall workflow. The actual execution of batch jobs is handled by Condor [14], a job scheduling and resource management system, which maintains a queue of jobs and manages their parallel execution on multiple nodes of a cluster. This is illustrated in Figure 5.

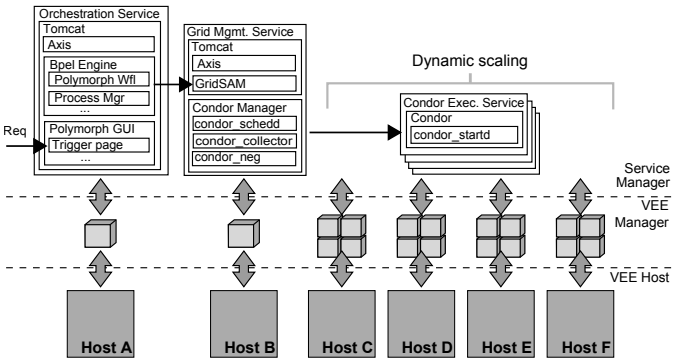

Fig. 5. Scientific grid application structure

The elasticity requirements are tied to the number of jobs currently in queue. Indeed, as jobs are created, the number of cluster nodes required to execute them will vary. Our goal in relying upon a cloud computing infrastructure will be to create a *virtualised cluster*, enabling the size of the cluster to dynamically grow and contract according to the needs of the application. This requires a monitoring agent to actively query the condor scheduler to obtain the size of the queue, which is deployed alongside the grid management node. The rules in use are similar to the example provided above.

The overall management process can hence be described as follows: upon submission of the manifest, the Service Manager will parse and validate the document, generating suitable individual deployment descriptors to be submitted to

the VEEM beginning with the Orchestration and Grid Management components. The VEEM will use these deployment descriptors to select a suitable host from the pool of known resources. These resources are running appropriate hypervisor technology, in this case Xen, from which the creation of new virtual machines can be requested. Upon deployment, the disk image is replicated and the guest operating system is booted with the appropriate virtual hardware and network configuration. When the Grid Management component is operational, a monitoring agent will begin the process of monitoring the queue length and broadcast the number of jobs in the queue on a regular basis (every 30 seconds) under the selected qualified name (`gridservice.queuelength`). These monitoring events, combined with appropriate service identifier information, will be recorded by the rule interpreter component of the Service Manager to enforce elasticity rules. When conditions regarding the queue length are met (i.e. there are more than 4 idle jobs in the queue), the Service Manager will request the deployment of an additional Condor Execution component instance. Similarly, when the number of jobs in queue falls below the selected threshold, it will request the deallocation of new virtual instances.

V. Related Work

It is worth briefly examining research developments related to service virtualisation, grid computing and component based software architecture description languages. There exists a number of software architecture description languages which serve as the run-time configuration and deployment of component based software systems. The CDDLM Component Model [15], for example, outlines the requirements for creating a deployment object responsible for the lifecycle of a deployed resource with focus on grid services. Each deployment object is defined using the CDL language. The model also defines the rules for managing the interaction of objects with the CDDML deployment API. However, with the focus being on the application level services provided, the type of deployment object is not suited to a virtual machine management, which is at a much lower level of abstraction.

With respect to current developments in production level cloud environments, auto-scaling has been introduced in Amazon EC2 to allow allocated capacity to be automatically scaled according to conditions defined by a service provider. These conditions are defined based on observed resource utilisation, such as CPU utilisation, network activity or disk utilisation. Whilst the approach laid out in this paper can be used to define elasticity rules based on such metrics, this can prove limiting. With respect to the use cases, scaling could only take place with respect to application level parameters.

VI. Conclusion and Future Work

In this paper, we have investigated the problem of service definition and dynamic provisioning of services in computational clouds. We have proposed a syntax and approach for a service manifest language which builds on the OVF standard and allows the management of service elasticity. Moreover, we have examined how our approach enables service elasticity to be expressed and handled with respect to two production level use cases. We believe the overall manifest language to be sufficiently expressive to tackle production level cases with appropriate support from the infrastructure.

The implications of such a capability are clear. Because the elasticity rules enable the service provider to flexibly expand and shrink their resource demands, these will only need to pay for the resources that they need. This provides an opportunity for service and infrastructure provider to optimise investment in hardware as well as the allocation process. This does however call to attention the strong need for service level guarantees that take elasticity into account. Indeed, an infrastructure provider must be able to accommodate future potential capacity adjustments when planning allocations in order to ensure that it can actually meet demand. Federation does provide the potential to mitigate this problem, and we will investigate these issues in future work.

References

[1] L. M. Vaquero, L. Rodero-Merino, J. Cáceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, 2009.

[2] B. Rochwerger, A.Galis, D. Breitgand, E. Levy, J. Cáceres, I. Llorente, R. Montero, Y. Wolfsthal, M. Wusthof, S. Clayman, C. Chapman, W. Emmerich, E. Elmroth, and R. S. Montero, "Design for future internet service infrastructures," in *Towards the Future Internet - A European Research Perspective*. IOS Press, Apr. 2009, p. 350.

[3] J. Meattle, "YouTube vs. MySpace growth," [Online] http://blog.compete.com/2006/10/18/youtube-vs-myspace-growth-google-charts-metrics/, Oct. 2006.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM Press, 2003, pp. 164–177.

[5] J. Sugerman, G. Venkitachalam, and B.-H. Lim, "Virtualizing IO/Devices on VMware Workstation's Hosted Virtual Machine Monitor," in *Proc. of the 2001 USENIX Annual Technical Conference*. Boston, Mass: Usenix, Jun. 2001.

[6] "Amazon Elastic Compute Cloud (Amazon EC2)," [Online] http://aws.amazon.com/ec2.

[7] "IBM blue cloud," [Online] http://www-03.ibm.com/press/us/en/pressrelease/22613.wss.

[8] "Open Virtualization Format Specification," Distributed Management Task Force, Specification DSP0243 v1.0.0, 2009.

[9] F. Galán, A. Sampaio, L. Rodero-Merino, I. Loy, V. Gil, L. M. Vaquero, and M. Wusthoff, "Service specification in cloud environments based on extensions to open standards," in *Fourth International Conference on COMmunication System softWAre and middlewaRE (COMSWARE 2009)*, Jun. 2009.

[10] Object Management Group, "Meta Object Facility Core Specification 2.0, OMG Document, formal/2006-01-01," 2006.

[11] J. Skene, D. D. Lamanna, and W. Emmerich, "Precise service level agreements," in *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 179–188.

[12] B. Rochwerger, A. Galis, E. Levy, J. Cáceres, D. Breitgand, Y. Wolfsthal, I. Llorente, M. Wusthoff, R. Montero, and E. Elmroth, "RESERVOIR: Management Technologies and Requirements for Next Generation Service Oriented Infrastructures," in *The 11th IFIP/IEEE International Symposium on Integrated Management,(New York, USA)*, 2009, pp. 1–5.

[13] 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. [Online]. Available: http://dx.doi.org/10.1007/s10723-005-9015-3

[14] D. Thain, T. Tannenbaum, and M. Livny, "Condor and the Grid," in *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and T. Hey, Eds. John Wiley & Sons Inc., December 2002.

[15] J. Tatemura, "CDDLM Configuration Description Language Specification 1.0," Open Grid Forum, Tech. Rep., 2006.