A Federated Multi-Cloud PaaS Infrastructure

Fawaz Paraiso, Nicolas Haderer, Philippe Merle, Romain Rouvoy, Lionel Seinturier

▶ **To cite this version:**

Fawaz Paraiso, Nicolas Haderer, Philippe Merle, Romain Rouvoy, Lionel Seinturier. A Federated Multi-Cloud PaaS Infrastructure. CLOUD 2012 - 5th IEEE International Conference on Cloud Computing, Jun 2012, hawaii, United States. pp.392 - 399, 10.1109/CLOUD.2012.79 . hal-00694700

HAL Id: hal-00694700
https://inria.hal.science/hal-00694700

Submitted on 8 May 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Federated Multi-Cloud PaaS Infrastructure

Fawaz Paraiso, Nicolas Haderer, Philippe Merle, Romain Rouvoy, Lionel Seinturier
University of Lille & Inria Lille - Nord Europe
LIFL UMR CNRS 8022, France
firstname.lastname@inria.fr

***Abstract*—Cloud platforms are increasingly being used for hosting a broad diversity of services from traditional e-commerce applications to interactive web-based IDEs. However, we observe that the proliferation of offers by cloud providers raises several challenges. Developers will not only have to deploy applications for a specific cloud, but will also have to consider migrating services from one cloud to another, and to manage distributed applications spanning multiple clouds. In this paper, we present our federated multi-cloud PaaS infrastructure for addressing these challenges. This infrastructure is based on three foundations: i) an *open service model* used to design and implement both our multi-cloud PaaS and the SaaS applications running on top of it, ii) a *configurable architecture* of the federated PaaS, and iii) some *infrastructure services* for managing both our multi-cloud PaaS and the SaaS applications. We then show how this multi-cloud PaaS can be deployed on top of thirteen existing IaaS/PaaS. We finally report on three distributed SaaS applications developed with and deployed on our federated multi-cloud PaaS infrastructure.**

***Keywords*-Federation; PaaS; SaaS; SCA; interoperability;**

I. Introduction

Cloud computing is a major trend in current research for building scalable distributed computing environments. In particular, Cloud computing emerged as a way for *"enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."* [1]. Several layers of cloud computing exist, including the infrastructure, platform, and application layers, which provide to end-users functionalities referred to as IaaS, PaaS, and SaaS, respectively [2]. Amazon Elastic Compute Cloud (Amazon EC2), Windows Azure, and Google App Engine are three of most well-know cloud platform providers, yet the offer has increased rapidly over the last months and tens of solutions are now available[1]. Besides, many key players in the IT business are also offering private cloud solutions for their data centers. Nevertheless, this proliferation of solutions raises several key challenges:

Portability: To avoid the vendor lock-in syndrome, SaaS must be portable on top of various cloud PaaS and IaaS providers. This portability allows the migration from one provider to another in order to take advantage of cheaper prices or better qualities of services (QoS). However, SaaS portability requires the runtime support provides a common model to hide the diversity of underlying PaaS and IaaS.

Interoperability: The diversity of offers combined with cloud services going mainstream will lead to scenarios of distributed SaaS applications whose parts are hosted on different cloud platforms, and will therefore need to interoperate and cooperate through efficient and reliable protocols.

Heterogeneity: Various protocols can be used to support interactions between services distributed on the clouds, such as *Simple Object Access Protocol* (SOAP), *REpresentational State Transfer* (REST), *JavaScript Object Notation* (JSON), *Google Web Toolkit* (GWT) RPC, just to name a few of them. PaaS will then need to deal with the heterogeneity of these service-oriented protocols and let SaaS to select the best fitting protocol according to business requirements.

Geo-diversity: Finally, [2] advocates that small data centers, which consume less power, may be more advantageous than large ones, and that geo-diversity tends to better match user demands. This has led to the idea that federated cloud platforms, so-called intercloud solutions [3], are required.

In this paper, we present our federated multi-cloud PaaS infrastructure for addressing these challenges. This paper elaborates on a previously published version [4] where we reported on an initial experiment that has consisted in deploying our infrastructure on eleven existing cloud systems: Amazon EC2, Amazon Elastic Beanstalk, BitNami, CloudBees, Cloud Foundry, Dot-Cloud, Google App Engine, Heroku, InstaCompute, Jelastic, and OpenShift. This paper goes beyond this first experiment and details the architecture and the underlying concepts of our infrastructure as well as three examples of SaaS exploiting this infrastructure.

The remainder of this paper is therefore organized as follows. Section II presents the software architecture promoted by our solution. In Section III, we report on the way our infrastructure can be deployed to federate existing cloud systems. Section IV describes three SaaS applications that have been designed and implemented to validate our infrastructure. Section V compares our infrastructure with the state-of-the-art, while Section VI concludes this paper and sketches some of future works we intend to address.

[1] http://tinyurl.com/6ntl968

II. Architecture

The architecture of our federated multi-cloud PaaS relies on three foundations: *i)* an *open service model* (cf. Section II-A) that is used to design and implement both our multi-cloud PaaS and the SaaS applications that run on top of it, *ii)* a *configurable architecture* of the federated PaaS (cf. Section II-B), and *iii)* some *infrastructure services* (cf. Section II-C) for managing both our multi-cloud PaaS and the SaaS applications.

A. Open Service Model

Service-Oriented Computing (SOC) promotes the idea of assembling software components into a network of loosely coupled services [5]. SOC has proved to be an adequate solution for building flexible and agile software systems that are resilient to changes. In this context, the OASIS consortium specifies the *Service Component Architecture* (SCA) standard [2] for designing and running service-oriented distributed applications. SCA promotes a vision of SOC where services are independent of implementation languages (Java, Spring, BPEL, C++, COBOL, C, etc.), remote communication and service access technologies (Web Services, JMS, etc.), interface definition languages (WSDL, Java, etc.) and non-functional properties. SCA thus provides a framework that can accommodate many different forms of SOC systems, then addressing *portability*, *interoperability* and *heterogeneity* challenges identified in Section I.

The different layers of a cloud environment (IaaS, PaaS, SaaS) provide dedicated services. Although their granularity and complexity vary, we believe that a principled definition of these services is needed to promote the interoperability and federation between heterogeneous cloud environments. Given the properties stated in the previous paragraph, we believe that SCA is an adequate substrate to address these challenges. Hence, our multi-cloud infrastructure uses SCA both for the definition of the services provided by the federated PaaS layer and for the services of the SaaS applications that run on top of this PaaS.

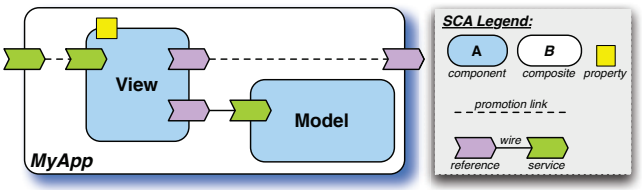

Figure 1. Overview of an SCA application.

As illustrated in Figure 1, the basic SCA building blocks are software *components*, which have *services* (or provided interfaces), *references* (or required interfaces) and expose *properties*. The references and services are connected by means of *wires*. SCA specifies a hierarchical component model, which means that components can be implemented either by primitive language entities or by subcomponents. In the latter case the components are called *composites*. Both component references and services can be exposed at the composite level by means of *promotion links*. To support service-oriented interactions via different communication protocols, SCA provides the notion of *binding*. For SCA references, a binding describes the access mechanism used to invoke a remote service. In the case of services, a binding describes the access mechanism that clients use to invoke the service. Our FraSCAti platform [6], [7] provides a reference implementation of this open service model.

B. Configurable Federated Multi-PaaS Infrastructure

Our federated multi-PaaS infrastructure relies on a configurable kernel. This kernel can be specialized to fit the characteristics of concrete cloud environments. Section III shows how this specialization has been performed for thirteen existing cloud environments.

This kernel draws inspiration from our FraSCAti [6], [7] platform for reconfigurable SOA and from the domain of *Software Product Line* (SPL) design. An SPL can be defined as *"a set of software-intensive systems that share a common, managed set of features and that are developed from a common set of core assets in a prescribed way"* [8]. A Feature Model is used to compactly define all features in an SPL and their valid combinations [9].

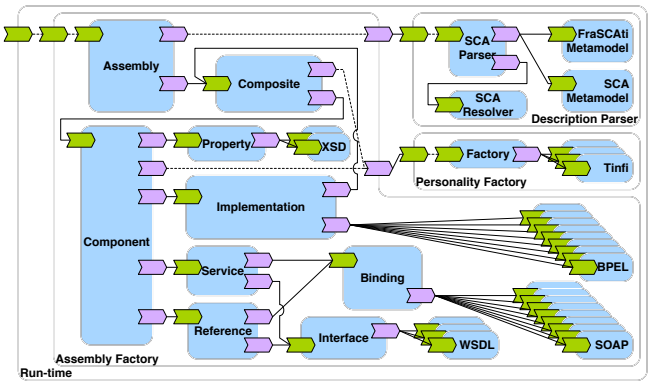

Figure 2. Architecture of the PaaS kernel.

Basically, the idea is *i)* to define an SPL that captures the common characteristics and the points of variability of cloud environments, and *ii)* to implement this SPL as an assembly of SCA components with the FraSCAti platform. The points of variability are captured as alternate components, so-called plugins, at some given locations of the architecture. Figure 2 illustrates the approach. The multi-lines that originate from some of the components, e.g. `Binding`, symbolize these alternative implementations.

As a matter of illustration, 19 plugins are available to support different component implementation languages:

[2] http://www.oasis-opencsa.org

Java, BPEL, Spring, Fractal ADL, OSGi (Equinox, Felix, JBoss OSGi or Knopflerfish), Scala, scripting languages (BeanShell, FScript, Groovy, JavaScript, JRuby, Jython, Xquery), Apache Velocity, and Web resources. Similarly, 11 plugins support different binding technologies: HTTP, JMS, JSON-RPC, REST, Java RMI, SOAP, JGroups, SLP, UPnP, OSGi, and JNA. Additional plugins exist to support different interface definition languages (Java, WSDL, UPnP service description). An API is provided for developing new plugins in order to address unforeseen requirements. Furthermore, several other functionalities of the platform (remote REST management, Web Explorer, dynamic code generation and compilation, SCA metamodel parsing, etc.) are also plugin-based leading to a set of 63 existing plugins in the FRASCATI code base[3] at the date of the writing of this paper. For further details, readers can refer to [10] and to the FRASCATI user manual[4] where the full SPL for the FRASCATI platform is described.

Overall, the plugin-based architecture and the SPL provide a way to address both *interoperability* and *heterogeneity* challenges identified in Section I.

C. Infrastructure Services

In addition to the generic architecture presented in the previous section, this section presents the set of common services that are provided to assist cloud application developers in using our infrastructure.

Basically, our infrastructure enables to federate distributed nodes, where each node is a particular cloud environment that hosts SaaS applications and an instance of our generic kernel specialized for this particular cloud. Each SaaS application corresponds to a so-called SCA domain and the required mechanisms for remote communications between these domains are provided by the kernel. Let remind also that both the kernel and its hosted SCA-domain are assemblies of SCA components. The challenge is then to be able to manage the interconnection topology between the different distributed nodes. We enumerate below the services that are provided by our infrastructure.

Cloud Node Provisioning: Our infrastructure provides some facilities for provisioning resources on cloud environments. The idea is to be able to allocate resources prior to the deployment phase. The purpose is twofold. First, the service acts as a cache for cloud resources. Provisioned nodes can be used later on, on-demand, when they will be needed.

This speeds up the deployment by skipping the provisioning, which will have been performed in advance. Second, the service provides a preparation phase for distributed deployment. To ensure that a distributed application can be deployed, the service first allocates resources on all concerned nodes, and then deploy the kernel and the applications on each node. Of course, such a scheme can lead to well-known issues of deadlock, and a mechanism is provided to release resources if all nodes cannot be provisioned.

PaaS Deployment Service: The second service consists in being able to deploy an instance of the configurable kernel and an instance of a SaaS application onto a particular node/-cloud. Based on our previous work that deals with a generic solution for deployment in distributed environments [11], we identified six main features that needs to be captured: *packaging*, *upload*, *install*, *start*, *stop*, *remove*. *packaging* refers to the way the SaaS application and the kernel need to be bundled for a particular node. *upload* refers to the operation which will remotely let the package be available on the node. *install* consists in preparing for execution an uploaded package. *start* puts the application into operation. *stop* discontinues the operation of the application. *remove* uninstalls the application from the node. Overall, our infrastructure provides, via some dedicated scripts, these 6 features for the 13 cloud environments that are supported (cf. Section III).

SaaS Deployment Service: This service allows to dynamically deploy/undeploy SaaS applications running on PaaS nodes. SaaS applications are packaged as ZIP or Java archives (respectively .zip or .jar files). Each archive contains a list of SCA composites to deploy, and their associated implementation artifacts like .class, WSDL, BPEL, scripts, etc. files. Each PaaS node provides an API to upload SaaS archives and then deploy their contained SCA composites. This API allows also to control the scope of the deployment: only on the called node, on all the nodes of a federation, or on a set of nodes matching some constraints.

Federation Management Service: Once deployed, the PaaS nodes and the SaaS applications need to be managed. Two main tasks are envisioned for this service: supervision and reconfiguration. In both cases, these tasks can be applied for a single node and for the federation. Since both the PaaS kernels and the SaaS applications are SCA-based, these tasks are universal in the sense that they can be applied indifferently either on the application or on the kernel levels (or both if required). The supervision task enables to interact with the SCA components deployed on a node. The supported interactions consist in triggering operations on a component and in retrieving the software architecture defined between components. The reconfiguration task deals with the update of the topology of the federation and of the software architecture on a particular node.

Three complementary means are provided by our infrastructure for achieving these tasks: a RESTful API, the FRASCATI scripting *Domain Specific Language* (DSL), and a workflow-based approach. The RESTful API is a low level API defining 15 basic operations for supervising and reconfiguring any SCA component. The FRASCATI scripting DSL takes advantage on this low level API to provide an higher-level and user-friendly syntax for performing supervision and reconfiguration on a given node, on all nodes

[3] http://websvn.ow2.org/listing.php?repname=frascati

[4] http://frascati.ow2.org/doc/1.4/ch12.html

or a selected set of nodes. Finally, when the operations need to span several different nodes, a BPEL workflow can be deployed to orchestrate the invocations of the RESTful API on the different nodes. Readers interested in obtaining more information about the API and DSL may refer to [7].

All these services are therefore implemented as SCA components. They are exposed as both SOAP and REST services allowing third-party client applications to simply invoke them, or orchestrate them via complex BPEL processes. All these management services are plugins of the PaaS kernel, and thank to our SPL approach, they could be deployed on demand according to management requirements. Finally, this open approach allows administrators to easily develop new management services addressing new management requirements.

III. Supported Cloud Environments

The previous section described our generic multi-PaaS infrastructure. This section reports on the concrete cloud environments on which this infrastructure has been deployed. For that, we need to *i)* configure our generic architecture to match the characteristic of the target cloud, and *ii)* deploy our infrastructure on this target cloud. Overall, as reported in the remainder of this section, we are supporting, thirteen target cloud environments.

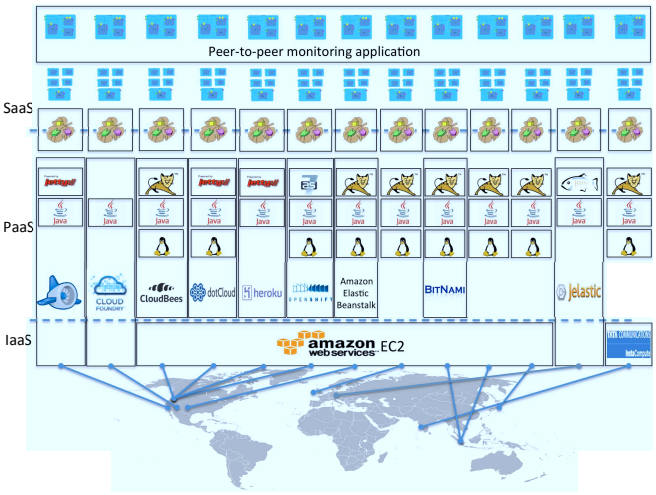

Figure 3. The FraSCAti multi-cloud platform.

This deployment has been conducted with IaaS/PaaS providers offering free trial accounts. In particular, we provisioned IaaS resources (*i.e.*, CPU, memory, storage, network, load balancer, firewall, and public IP address) from *Amazon EC2*[5] and Tata Communications's *InstaCompute*[6]. After configuring these IaaS resources, we installed a PaaS stack composed of a Linux distribution, a Java virtual machine, and a Web application container.

[5]http://aws.amazon.com/ec2
[6]http://iaas.tatacommunications.com

We also provisioned PaaS resources (*i.e.*, PaaS stacks deployed on top of provisioned IaaS resources) from *Amazon Elastic Beanstalk*[7], *BitNami*[8], *CloudBees*[9], *Cloud Foundry*[10], *DotCloud*[11], *Google App Engine* (GAE)[12], *Heroku*[13], *Jelastic*[14] , and Red Hat's *OpenShift*[15]. Let us note that Amazon Elastic Beanstalk, BitNami, CloudBees, DotCloud, Heroku, and OpenShift provision IaaS resources from Amazon EC2. All these PaaS provide a Linux distribution, a Java virtual machine, and a Web application container off-the-shelf.

Table 4 summarizes the characteristics of the thirteen nodes we provisioned for our federated multi-cloud PaaS infrastructure. The reader may have noticed that, with CloudBees, Cloud Foundry, DotCloud, GAE, Heroku, Jelastic, and OpenShift, descriptions of IaaS resources are not available because these PaaS provision them automatically and hide them to the end-user. The resulting multi-cloud platform therefore exhibits a wide diversity of geolocations (Germany, India, Ireland, Japan, US, Singapore) and SaaS containers (GlassFish, Jetty, Tomcat, JBoss) as depicted in Figure 3. This multi-cloud infrastructure is publicly accessible from the FraSCAti web site[16].

IV. Validation

In order to validate our federated multi-PaaS infrastructure, we report in this section on three SaaS applications that have been developed and deployed.

A. P2P Monitoring Network

The first SaaS application consists on a distributed peer-to-peer monitoring network application deployed on the thirteen concrete cloud environments supported by our infrastructure. This application can be accessed online from the FraSCAti web site[17].

This application is composed of 13 peers, one for each of the 13 cloud environments. Each peer is connected to the 12 other peers to build a mesh topology. Each peer periodically monitors and reports on the activity of its peers.

Each peer is implemented by an SCA composite of three SCA components: a *sensor* exposes local monitoring data (peer name, url, geolocation, hostname, IP address, current date, available processors, free/total/max memory) as a REST resource, an *aggregator* collects monitoring data from all the peers and computes network latency, and the

[7]http://aws.amazon.com/elasticbeanstalk
[8]http://bitnami.org
[9]http://www.cloudbees.com
[10]http://www.cloudfoundry.com
[11]http://www.dotcloud.com
[12]http://code.google.com/appengine
[13]http://www.heroku.com
[14]http://jelastic.com
[15]http://openshift.redhat.com
[16]https://wiki.ow2.org/frascati/Wiki.jsp?page=Cloud
[17]http://frascati.ow2.org

Cloud		IaaS hardware			PaaS software stack		
Provider	Location	CPU	RAM	Storage	OS	JRE	Web container
Amazon EC2	Asia Pacific (Singapore)	1 EC2 virtual core	613MB	8GB	Amazon Linux	OpenJDK 1.9.1	Apache Tomcat 7.0.20
Amazon EC2	Asia Pacific (Tokyo)	1 EC2 virtual core	613MB	8GB	Amazon Linux	OpenJDK 1.9.1	Apache Tomcat 7.0.20
Amazon EC2	EU North (Ireland)	1 EC2 virtual core	613MB	8GB	Amazon Linux	OpenJDK 1.9.1	Apache Tomcat 7.0.20
Amazon Elastic Beanstalk	US East (Virginia)	1 EC2 virtual core	613MB	8GB	Amazon Linux	OpenJDK 1.9.1	Apache Tomcat 7
BitNami	Amazon EC2 Singapore	1 EC2 virtual core	613MB	3GB	Amazon Linux	JVM 6	Apache Tomcat 6.0.29
CloudBees	Amazon EC2 US West	Data not available				JVM 6	Apache Tomcat 6.0.32
Cloud Foundry	US Utah	Data not available					
DotCloud	Amazon EC2 US West	Data not available				Oracle JRE 1.6.0 24-b07	Jetty 6.1.22
Google App Engine	US	Data not available					Google Jetty
Heroku	Amazon EC2 US West	Data not available				JVM 6	Jetty 7.4.5
InstaCompute	India	1 core 1.00 GHz	1GB	20GB	CentOS 5.4 X64	Oracle JRE 1.6.0 26-b03	Apache Tomcat 7.0.20
Jelastic	Germany	Data not available				JVM 6	GlassFish 3.1.1
OpenShift from Red Hat	Amazon EC2 US West	Data not available					JBoss AS 7.0

Figure 4. The provisioned multi-cloud platform

view component produces a dynamic HTML page showing a Google Map geolocating the thirteen peers, network latencies between peers, and all collected monitoring data (cf. Figure 5). To summarize, this monitoring application is composed of 13 SCA composites, 39 SCA components, and uses two external services computing peer geolocation[18] and maps, respectively.

Figure 5. The multi-cloud P2P network.

In terms of configuration, this application uses 22 FRASCATI plugins: 2 component implementation languages (Java and Apache Velocity), 3 binding technologies (HTTP, REST, SOAP), 2 interface languages (WSDL, Java), 3 metamodels (OSOA SCA, FRASCATI SCA, FRASCATI Web), 8 core plugins (SCA parser, Assembly Factory, Component Factory, Binding Factory, Remote REST Management, FRASCATI Web Explorer), and 4 plugins for dynamic code generation and compilation. However, the last 4 plugins have not been deployed on GAE since writing on the filesystem is forbidden by this platform. This applications plus all the required FRASCATI features were packaged as a Web ARchive (WAR) of a size of 15.5MB. On the CloudBees PaaS, we observed a memory consumption of 95MB including the Java virtual machine, the Web application container, FRASCATI and this SaaS application. This application shows that our federated multi-cloud PaaS infrastructure fully addresses the challenges of *Portability*, *Interoperability*, *Heterogeneity*, *Geo-diversity* introduced in Section I, and our infrastructure could be tailored to meet constraints of each targetted PaaS/IaaS thank to our PaaS kernel SPL approach (cf. Section II-B).

[18]http://freegeoip.net

B. DiCEPE

This section reports on the DICEPE SaaS, another application developed with our federated multi-cloud PaaS infrastructure. DICEPE is a distributed *complex event processing* (CEP) platform, which is designed to integrate different complex event engines [12] and to interact via various remote communication protocols (*e.g.*, HTTP, WS-Notification, JMS).

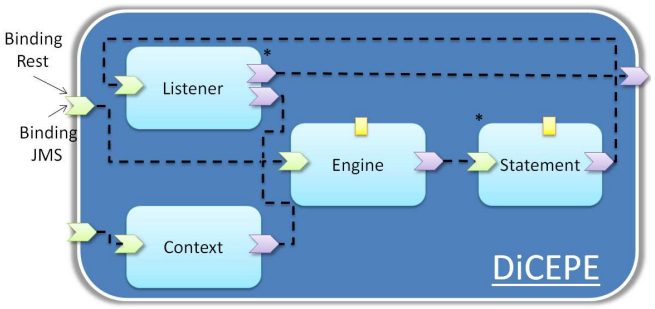

Figure 6. Overview of DICEPE architecture.

The architecture of DICEPE, which is illustrated in Figure 6, is composed of four SCA component types: *Engine*, *Statement*, *Listener*, and *Context*. The *Engine* component encapsulates a given CEP engine, and then performs operations on event objects, including creating, reading, transforming, aggregating, correlating or removing them. A *Statement* component implements a trigger when a given complex event query is matched by the CEP engine. The *Engine*

component is wired to all the *Statement* components. *Listener* component generates another complex event when an action is detected and *Context* component collects statements deployed in the engine component at runtime. Each of them is wired to the *Engine* component.

To evaluate the performance of DICEPE, we wrote a benchmark that mines weather data. For that, we took advantage of the existing database of the National Climatic Data Center (NCDC web site[19]). The database for year 2010 stores 6GB of events collected by weather sensors around the globe. We then fed 1GB of these events to the DiCEPE SaaS application, in order to determine the largest temperature. This dataset represents 4001495 weather events to process by the CEP engine. All the experiments were performed on an HP Z4000 Workstation with a 2.67 GHz Intel(R) Xeon processor, 16 GB RAM with Ubuntu Server 3.0.0-12 64 bit and Oracle java 1.6. We evaluated three implementations of this scenario where events are processed: *i)* natively with the Esper CEP engine[20], *ii)* with DiCEPE using Esper and implemented on top of our FRASCATI 1.5 PaaS kernel, and *iii)* with DiCEPE using Esper and implemented on top of Apache Tuscany 1.6 [21]—another implementation of the SCA standard. The scenario was executed ten times on each of the three implementations. For each, the following table reports on the mean execution time in seconds and the mean overhead introduced by the used SCA runtime.

Implementation	Execution time	SCA overhead
Esper	118.862 s	-
DiCEPE (Esper + FraSCAti)	139.513 s	17.37%
DiCEPE (Esper + Tuscany)	154.079 s	29.63%

Overall, this big data benchmark shows that the overhead introduced by SCA (17.37% and 29.63%) is negligible compared to the benefits that SCA provides (cf. Section II-A). Even if SCA runtimes could be still optimized in order to reduce their overhead, we can already consider that SCA is ready for intensive real-time distributed complex event processing applications. Moreover, this benchmark also validates both our open service model (cf. Section II-A) and the FRASCATI-based PaaS kernel (cf. Section II-B) as it provides a smaller overhead than the one imposed by using Apache Tuscany, which is the SCA reference implementation embedded into the industrial IBM's WebSphere Application Server product.

C. AntDroid

Scientific communities extensively exploit simulations to validate their theories. However, the relevance of the obtained results highly depends on the accuracy of the dataset they use. This statement is particularly true when considering human mobility traces, which tend to be highly unpredictable. ANTDROID is a federated SaaS delivered to scientists for sensing the activities of mobile users [13].

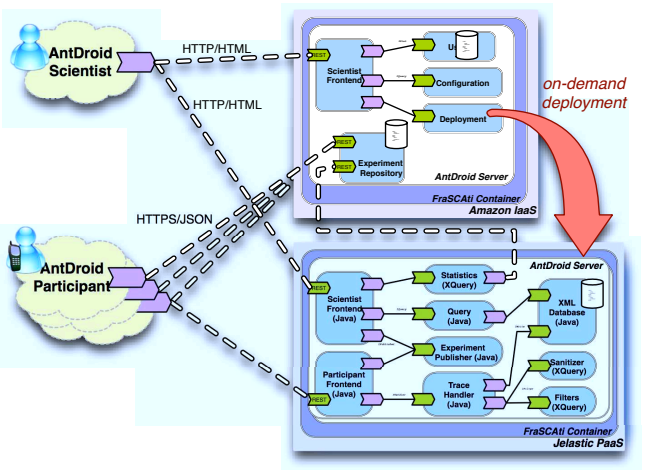

Figure 7. On-Demand SaaS Deployment.

The originality of ANTDROID lies in its multi-cloud orientation and its modular infrastructure, which can be configured and customized according to the scientist requirements. To achieve this degree of modularity, ANTDROID adopts a star topology whose central node consists in a factory of edges (cf. Figure 7). New scientists connect to the central node in order to create a new account and to configure their own instance of SaaS. The configured SaaS is a dedicated environment, which can be used by the scientist to create and publish new experiments, and then process the collected mobility traces. ANTDROID therefore reflects the features of the environment (real-time processing, data visualization, data exportation, etc.) as a *software product line*, which is used by the scientist to build her/his own data collection SaaS product. ANTDROID deals with the vendor lock-in syndrome by offering the scientist the opportunity to select the IaaS or the PaaS she/he prefers to use depending on her/his preferences or to ethical and privacy issues related to the storage of collected traces. From the central node, the scientist can trigger the deployment (or the download) of the configured SaaS as a web archive (war) or a virtual appliance [14]. ANTDROID SaaS are therefore deployed over a wide diversity of IaaS and PaaS and are connected to an experiment directory service exposed by the central node. Once deployed, the scientist can connect to her/his SaaS to create a new experiment, public experiments are automatically registered within the central directory service. ANTDROID participants connect to the central directory service to retrieve the list of proposed experiments. When enrolling in an experiment, the participant retrieves the URL of the SaaS that will be used to upload the collected data.

This multi-cloud application architecture therefore provides solution that better scales than multi-tenant architec-

[19] http://www.ncdc.noaa.gov
[20] http://www.espertech.com
[21] http://tuscany.apache.org

tures and does not impose a single cloud provider to its users. This validates provisioning and deployment services provided by our federated multi-cloud PaaS infrastructure (cf. Section II-C).

V. Related Work

Virtual IaaS solutions: BitNami, CloudBees, DotCloud, Heroku, and OpenShift are cloud computing providers acting as cloud intermediaries by providing developers with an application server provisioned on top of the Amazon EC2 IaaS. These providers are therefore offering a dedicated PaaS on top of an existing IaaS, but these solutions do not cross the boundaries of Amazon EC2 and cannot be used to deploy a multi-cloud system. The solution we propose in this paper rather builds on virtual IaaS platforms to provide a unified open programming model for SaaS, based on the SCA standard.

Provisioning of heterogeneous IaaS: Most of the current research activities focus on the provisioning of web applications in heterogeneous clouds [15], [16]. These approaches are resource allocation algorithms that match the resources of the node provisioned by the IaaS (CPU, I/O) in order to maximize the performances of the deployed PaaS. Although these approaches do not apply to multi-cloud systems, we believe that they can be used for considering the automatic provisioning of clouds according to pricing and latency dimensions. Several open source libraries exist to manage heterogeneous IaaS—*i.e.*, deploy images and create/start/stop/destroy virtual machines, such as Apache Deltacloud[22], Apache Libcloud[23], jclouds[24], and SimpleCloud[25]. Each of them provides its own API abstracting the heterogeneity between underlying IaaS management API. Nevertheless, these abstractions are too low level as they provide an imperative programming model instead of a declarative model.

SCA-based SaaS: To avoid the vendor lock-in syndrome, [17] proposes an SCA-based format for packaging and deploying multi-tenant aware configurable composite SaaS applications. This format extends SCA with variability descriptors and SaaS multi-tenancy patterns. We think that this SCA-based format can be reused or extended for multi-cloud systems. To avoid lock-in to a specific cloud application platform, Apache Nuvem[26] defines an open SCA-based application programming interface for common cloud application services, allowing applications to be easily ported across the most popular cloud platforms. [18] shows how to build and integrate SCA-based composite applications using Apache Tuscany, the Eucalyptus open source cloud framework, and OpenVPN to create an hybrid composite application. Technical talks at JavaOne 2010[27] and ApacheCon NA 2010[28] presented how to develop composite applications for the cloud using Apache Tuscany. Our experiment is deployed as a larger multi-cloud platform and FraSCAti brings reflective capabilities to SCA, which is not the case for Apache Tuscany.

VI. Conclusion

Numerous cloud computing environments, either PaaS or IaaS, are now available. They open many perspectives for next generations of applications and services. This proliferation of offers raises new challenges which require to federate these cloud environments. This paper proposes a step towards this direction with a federated multi-cloud PaaS infrastructure. Our solution is composed of an open service model and a generic kernel infrastructure. The open service model is used both for the PaaS infrastructure and the SaaS applications hosted on top of it. The infrastructure has been specialized and deployed on thirteen existing cloud environments: Amazon EC2, Amazon Elastic Beanstalk, BitNami, CloudBees, Cloud Foundry, Dot Cloud, Google App Engine, Heroku, InstaCompute, Jelastic, and OpenShift. To our knowledge, our infrastructure is the largest worldwide federated multi-cloud PaaS infrastructure. Dedicated infrastructure services are provided to provision nodes, deploy applications and services, and manage a federation. We reported on three applications that have been designed and implemented with this infrastructure. Readers may refer to our web site[29] where the P2P monitoring application can be accessed.

Many perspectives are open by this work. Some of them consists in extending to the intercloud case the existing properties of cloud environments. For example, intercloud elasticity is one of them. The idea would be, in case of activity bursts that cannot be handled by a single cloud, to migrate and balance this activity between parts of the application hosted on different clouds. In addition, one can note that our infrastructure introduces a layer that abstracts the services provided by IaaS and PaaS environments. Yet in some cases a finer grained interoperability between the services provided by these environments would be desirable to fine tune and tailor applications. Furthermore, the diversity of pricing strategies of cloud solution providers will make it relevant for third party actors to come up with offers proposing the highest level of resources for the best price. As an economic optimum may vary over time, or may differ depending on the kind of resources requested, we will investigate the integration of brokering intermediaries to move opportunistically between these cloud solutions. Finally, federated multi-cloud security and SaaS multi-tenancy

[22] http://incubator.apache.org/deltacloud
[23] http://libcloud.apache.org
[24] http://www.jclouds.org
[25] http://simplecloud.org
[26] http://incubator.apache.org/nuvem
[27] http://tinyurl.com/6oanleg
[28] http://tinyurl.com/7bl7o84
[29] `http://frascati.ow2.org`

are also challenging perspectives. Globally, the challenge is to design a solution for accessing the specificities offered by each cloud environment, while keeping the common model that enables designing and implementing for all these clouds.

ACKNOWLEDGMENT

This work is partially funded by the French Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council and FEDER through the Contrat de Projets Etat Region Campus Intelligence Ambiante (CPER-CIA) 2007-2013, and the ANR (French National Research Agency) ARPEGE SocEDA project.

REFERENCES

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," tech. rep., National Institute of Standards and Technology, 2009. `http://www.nist.gov/itl/cloud/upload/cloud-def-v15.pdf`.

[2] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud Computing: State-of-the-art and Research Challenges," *Journal of Internet Services and Applications*, vol. 1, pp. 7–18, May 2010.

[3] R. Buyya, R. Ranjan, and R. Calheiros, "InterCloud: Scaling of Applications across multiple Cloud Computing Environments," in *10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'10)*, vol. 6081 of *LNCS*, pp. 13–31, May 2010.

[4] P. Merle, R. Rouvoy, and L. Seinturier, "A Reflective Platform for Highly Adaptive Multi-Cloud Systems," in *10th International Workshop on Adaptive and Reflective Middleware (ARM)*, pp. 1–7, Dec. 2011.

[5] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *IEEE Computer*, vol. 40, pp. 64–71, Nov. 2007.

[6] L. Seinturier, P. Merle, D. Fournier, N. Dolet, V. Schiavoni, and J.-B. Stefani, "Reconfigurable SCA Applications with the FraSCAti Platform," in *6th IEEE International Conference on Service Computing (SCC'09)*, pp. 268–275, Sept. 2009.

[7] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, and J.-B. Stefani, "A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures," *Software: Practise and Experience (SPE)*, 2012. To appear, online preview available on the SPE web site.

[8] P. Clements and L. Northrop, *Software Product Lines Practices and Patterns*. Addison-Wesley, 2002.

[9] D. Batory, "Feature models, grammars, and propositional formulas," in *Proceedings of SPLC'05*, vol. 3714 of *LNCS*, pp. 7–20, 2005.

[10] M. Acher, A. Cleeve, P. Collet, P. Merle, L. Duchien, and P. Lahire, "Reverse Engineering Architectural Feature Models," in *Proceedings of 5th European Conference of Software Architecture (ECSA'11)*, Sept. 2011.

[11] A. Flissi, J. Dubus, N. Dolet, and P. Merle, "Deploying on the Grid with DeployWare," in *Eighth IEEE International Symposium on Cluster Computing and the Grid*, (France), pp. 177–184, 2008.

[12] D. Luckman, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, 2001.

[13] N. Haderer, R. Rouvoy, and L. Seinturier, "AntDroid: A Distributed Platform for Mobile Sensing," Research Report RR-7885, Inria, Feb. 2012.

[14] C. Quinton, R. Rouvoy, and L. Duchien, "Leveraging Feature Models to Configure Virtual Appliances," in *2nd International Workshop on Cloud Computing Platforms (CloudCP'12)*, (Bern, Switzerland), Apr. 2012.

[15] J. Dejun, G. Pierre, and C.-H. Chi, "Resource Provisioning of Web Applications in Heterogeneous Clouds," in *2nd USENIX conference on Web Application development (WebApps'11)*, (Berkeley, CA, USA), USENIX Association, June 2011.

[16] G. Lee, B.-G. Chun, and R. H. Katz, "Heterogeneity-Aware Resource Allocation and Scheduling in the Cloud," in *3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'11)*, (Berkeley, CA, USA), USENIX Association, June 2011.

[17] R. Mietzner, F. Leymann, and M. P. Papazoglou, "Defining Composite Configurable SaaS Application Packages Using SCA, Variability Descriptors and Multi-tenancy Patterns," in *International Conference on Internet and Web Applications and Services*, (Los Alamitos, CA, USA), pp. 156–161, IEEE Computer Society, 2008.

[18] R. Bhose and K. C. Nair, "Integrating Composite Applications on the Cloud Using SCA," Mar. 2010. Dr. Dobb's, available at `http://drdobbs.com/cpp/223800269`.