PaaS-independent Provisioning and Management of Applications in the Cloud

Mohamed Sellami, Sami Yangui, Mohamed Mohamed and Samir Tata
Computer Science Departement
Institut Mines-Telecom, Telecom SudParis, CNRS UMR Samovar, Evry, France
Email: Firstname.Lastname@telecom-sudparis.eu

***Abstract*—The study we have conducted of existing cloud platforms shows that their operating requires the use of specific and proprietary APIs. This PaaS providers' policy is hampering the interactions between different clouds. If appropriate solutions are not considered, this issue would for instance slow down the democratization of clouds federation and cooperation. In this paper, we propose (i) a unified description model that allows the representation of applications independently of the targeted PaaS for their hosting and (ii) a generic PaaS application provisioning and management API (called COAPS API). Our proposed solution applies the separation of concerns principle by separating the provisioning and the management API from the defined description model. We motivate our solution with real use case scenarios and an implementation to show its feasibility.**

***Keywords*-Application model; Environment model; PaaS; Provisioning; Management; REST API**

I. Introduction

Cloud Computing is a new supplement, consumption, and delivery model for IT services based on Internet protocols. More and more companies are adopting the new economic model offered by cloud computing. For instance, in a Mckinsey Quarterly survey [1] conducted in 2010 on 332 companies, 75% believe that the use of cloud computing could drive value at their companies. Among these companies, 68% says that they are currently adopting the cloud to set up electronic collaboration and 82% are planning to do it in the 18 coming months. In this new world of business, electronic cooperation, collaboration and/or federation are inevitable.

Cloud computing offers innovative solutions and services to companies. In this paper, we focus on the Platform-as-a-Service (PaaS) layer and more particularly on application provisioning and management. To allow cloud cooperation and federation, each company's cloud must be able to seamlessly interact with different and heterogeneous PaaS (e.g. Cloud Foundry[1], Openshift[2], etc.). However, our study on existing cloud platforms (see Section VI) shows that their handling requires the use of specific and proprietary APIs. For example, to interact with the Force.com PaaS Apex REST API is provided [2], Cloud Foundry is delivered with a proprietary API (i.e. the Cloud Foundry core REST API), etc. Each existing PaaS exposes a different interface and no standardized (or generic) API is offered for PaaS consumers. Thereby, the actual PaaS Provider's policy makes a seamless interaction with their PaaS very difficult, if not impossible.

In this paper, we provide a PaaS-independent solution for PaaS application provisioning and management. We define a model, called the PaaS Resources and Applications Description Model, for the description of PaaS applications independently from their targeted PaaS. We also propose a generic API, called COAPS API, that allows human and/or software agents to provision and manage PaaS applications. This API provides an abstraction layer for existing PaaS allowing PaaS application provisioning in a unified manner. By using our description model to describe PaaS applications, and the COAPS API as a middleware with available PaaS offers, application providers can easily switch from one PaaS to another.

The remainder of this paper is organized as follows. In Section II, we present two realistic use cases for our PaaS application provisioning and management solution as motivating cases for our work. In Section III, we propose a model to describe PaaS resources and applications. Section IV presents our generic cloud application provisioning and management API (i.e. COAPS API). We discuss related work and show contributions of our solution in Section VI, before discussing our results and presenting directions for future work in Section VII.

II. Use Cases and Motivation

To show the use and utility of our cloud Application provisioning and management solution in realistic situations, we present in this section two real use cases: the CompatibleOne cloud broker project[3] and the EASI-CLOUDS project[4].

A. CompatibleOne

As a first use case for our solution, we present in this section the work done in the CompatibleOne project. CompatibleOne is an open source project which provides, among others, a platform (i.e. ACCORDS) for the description of and interoperability between different cloud resources provisioned by heterogeneous cloud service providers [3]. The ACCORDS

[1]http://www.cloudfoundry.com/
[2]https://openshift.redhat.com
[3]http://www.compatibleone.org
[4]http://easi-clouds.eu/

978-0-7695-5028-2/13 $26.00 © 2013 IEEE
DOI 10.1109/CLOUD.2013.105

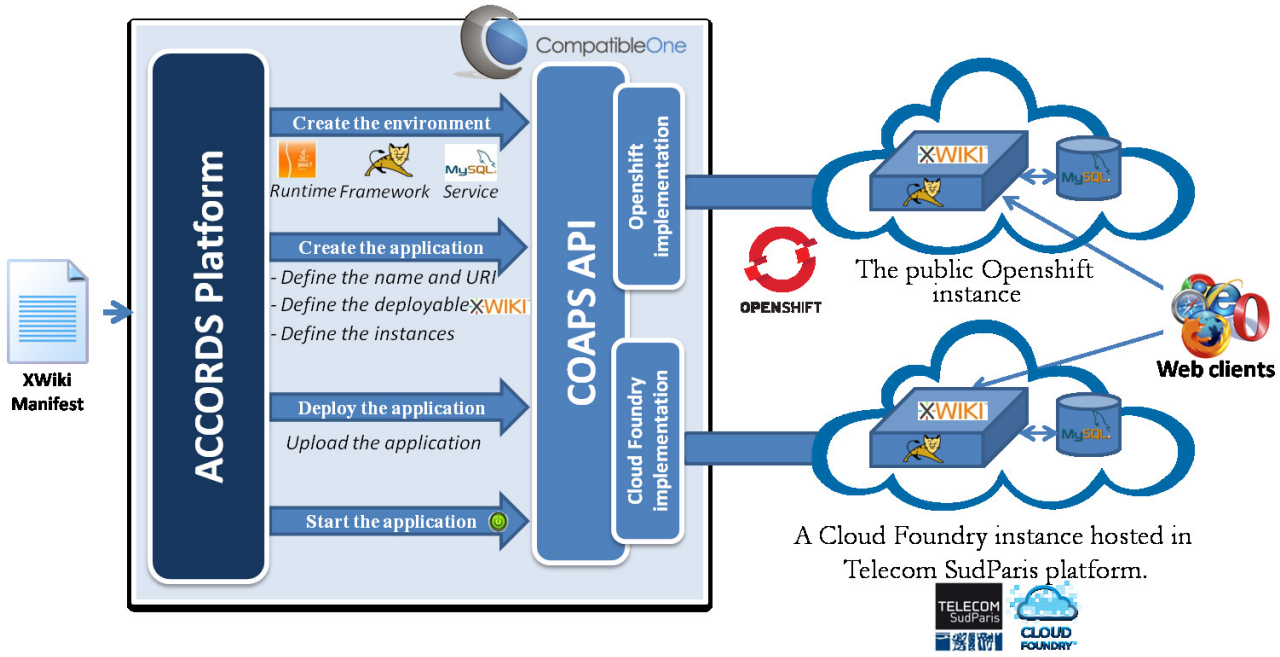

Fig. 1. COAPS demo presented in the CompatibleOne final review

platform authorizes application developers to choose the runtimes and frameworks of their choice to deploy their applications. The developers are not supposed to consider proprietary characteristics related to a specific PaaS (e.g. Google App Engine[5], Cloud Foundry, etc.). To ensure this interoperability requirement, the initial specifications and implementations of our solution (i.e. the PaaS resources and application description model and the COAPS API) were proposed. Describing applications using a generic model and using a unified API for their provisioning enabled us to meet the interoperability challenge. The related CompatibleOne module (COAPS[6] specifications and implementations) is available at http://gitorious.ow2.org/ow2-compatibleone/coaps.

As a proof of concept for the COAPS module, the demonstration illustrated in Fig. 1 was presented at the project's final review. This demonstration shows how the interoperability of the ACCORDS platform is ensured through our solution. We provisioned two different PaaS instances, a Cloud Foundry instance and an Openshift instance, with a same application (XWiki Enterprise[7]) using the same application descriptor (according to our model) and the same actions (defined in our API). XWiki Enterprise is a light and powerful development platform that allows users to customize the wiki to their specific needs (e.g. sharing documents, monitoring project progress, etc.). For our demonstration, the XWiki company provided us with a "Cloudified" version of the application. To provision the Cloud Foundry instance or the Openshift instance with the XWiki Enterprise application, the ACCORDS platform performs the same sequence of COAPS API operation calls to create the hosting environment of the application, upload its source archives and start it.

[5]https://developers.google.com/appengine/
[6]COmpatible Application Platform Service
[7]http://enterprise.xwiki.org/xwiki/bin/view/Main/WebHome

B. EASI-CLOUDS

EASI-CLOUDS stands for Extendable Architecture and Service Infrastructure for Cloud-Aware Software. This project aims to offer novel and beneficial solutions for both cloud consumers and providers. The major expected outcome is an open-source cloud platform, the EASI-CLOUDS platform (see Fig. 2), "*that can be instantiated to set up an application type-specific cloud (e.g., e-learning, HPC-on-demand, storage marketplace) for a private, public, or hybrid usage, and implementing a given level of security, privacy and QoS*"[4]. An EASI-CLOUDS platform must also provide the required facilities for intra-cloud cooperation and federation.

In this project, one of our objectives is to provide the required facilities promoting EASI-CLOUDS platforms federation. Our PaaS-independent application provisioning and management solution can be used in this context to enable application provisioning/management (i) between EASI-CLOUDS platforms of a same federation and also (ii) with other commercial PaaS (see Fig 2). Currently, our solution is used in two different PaaS application delivery scenarios:

- A cloud application provider connects to an EASI-CLOUDS platform federation, to deploy its already created application on PaaS. In this scenario, the generic interface provided by the COAPS API, part of the REST APIs circle in Fig. 2, allows an EASI-CLOUDS platform

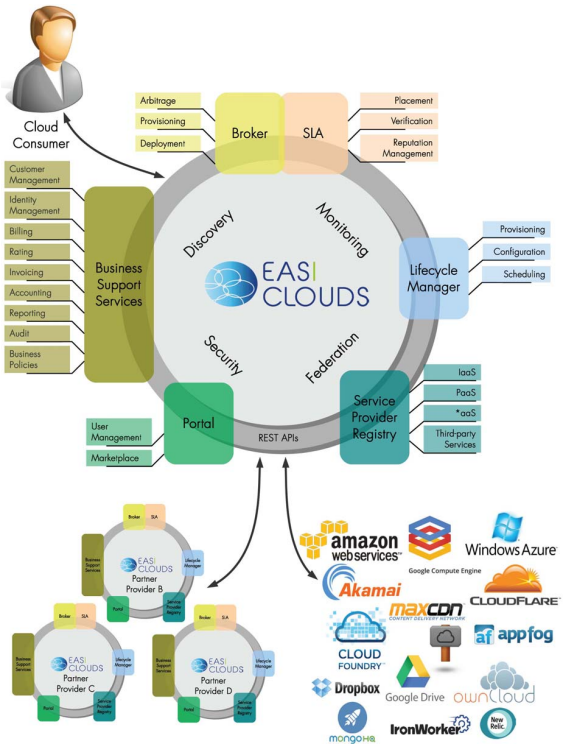

Fig. 2. Conceptual diagram of the EASI-CLOUDS platform [5]

to seamlessly interact with different and heterogeneous PaaS, including other EASI-CLOUDS platforms, to deploy the application.

- A cloud application provider connects to an EASI-CLOUDS platform, to create and deploy an application on PaaS. In this case, the provider uses an IDE-as-a-Service [6] provided by the EASI-CLOUDS platform to develop and deploy its application. In this context, EASI-CLOUDS partners from the Tampere University of Technology are currently extending their web-based collaborative real-time editor for software development (CoRED) [7] to integrate cloud application provisioning and management functionalities. This extension will be ensured by using our Application description model and the COAPS API.

III. PaaS Resources and Applications Description

In order to provision and manage applications on a PaaS through our COAPS API, one has to provide the application's deployable (source archives) and the corresponding application manifest. By application manifest, we mean an application descriptor detailing the application properties, requirements and its hosting environment (See Fig. 3).

As part of our work, we defined description models for both application and environment. These models are detailed respectively in Sections III-A and III-B. Then, to better explain these models, we present and comment in Section III-C

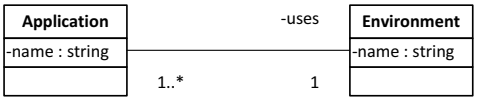

Fig. 3. Application manifest model.

the XWiki application manifest corresponding to our XWiki illustrative example introduced in Section II-A.

A. Application description model

The diagram illustrated in Fig. 4 represents the various entities that compose an application.

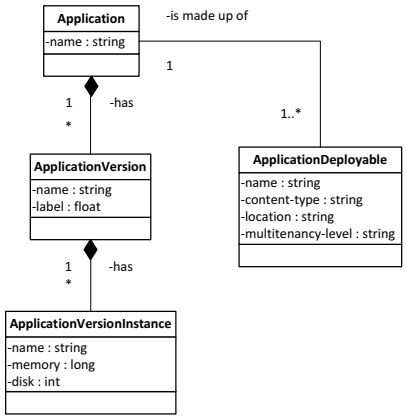

Fig. 4. Application description model.

The *application* entity is characterized by a unique *name* and has a set of *applicationVersion* entities. Each *applicationVersion* can be instantiated into *applicationVersionInstance* entities. Based on this, one can define as many running application instances as needed. Each one of these instances has its own *memory* and *disk* attributes. These attributes describes nonfunctional requirements of *applicationVersionInstance* entities. Furthermore, to each *application*, a set of *applicationDeployable* entities are associated. The *applicationDeployable* entity's *content-type* can be a artifact or a configuration file. The *location* attribute contains their URL. The *multitenancy-level* attribute indicates the degree of the application tenancy to apply by the PaaS once the application is deployed.

B. Environment description model

The diagram illustrated in Fig. 5 represents the various entities that compose an environment. By environment, we mean all platform components/resources needed to host and execute the application to deploy.

Each environment is characterized by a unique *name* and is instantiated from an *environmentTemplate* entity. Each *environmentTemplate* is described by a *name*, a *memory* value and a *disk* capacity. *Memory* and *disk* attributes are used to fix the

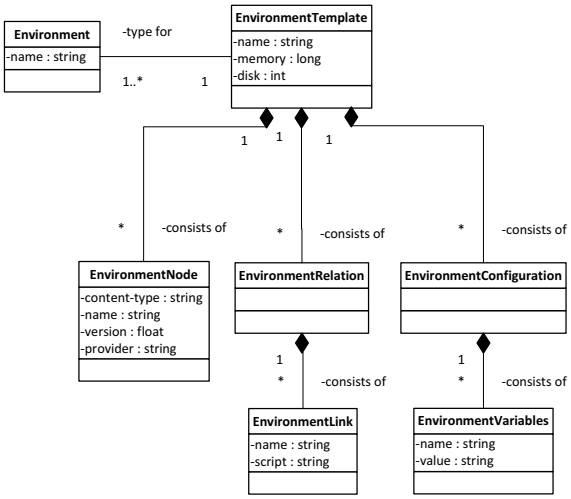

Fig. 5. Environment description model.

environment elasticity limit managed by the PaaS. In addition to that, the *environmentTemplate* is built from a set of entities:

- *EnvironmentNode* components: They represent platform resources associated to the *environmentTemplate*. They can take different *content-type* values in accordance with our OCCI PaaS resources extension defined in a previous work [8]. The possible values are:
  - *container* which are engine resources to host and run services (e.g. Apache Tomcat),
  - *database* which are storage resources for applications processing persistent data (e.g. MySQL),
  - *router* which are resources that provide protocols, messages format transformation and routing (e.g. Apache/JK).
- *EnvironmentRelation* components: They define relations between *EnvironmentNode* components,
- *EnvironmentConfiguration*: a set of actions/scripts necessary for the configuration and execution of requested *environmentNode* and *environmentRelation* resources.

Concrete entities associated to *EnvironmentRelation* and *EnvironmentConfiguration* components are respectively *EnvironmentLink* and *EnvironmentVariable*. *EnvironmentLink* components allow expressing bindings between allocated *EnvironmentNodes* (e.g. a database binding between database and container nodes). *EnvironmentVariable* components allow the specification of a set of variables necessary for the configuration and execution of requested *EnvironmentNode* components (e.g. an environment variable to configure a container node).

This description model is extensible and can follow the evolution of PaaS features. Indeed, one can add, if needed, new entities related to *EnvironmentRelation* and *EnvironmentConfiguration* components. Besides, any application and/or environment description in accordance to this model will be transferred through our API to the PaaS which interpret it according to its features and capabilities.

C. Example: the XWiki application manifest

In this section, we provide an example of an application description manifest. We consider the XWiki application example (See Fig. 1) and provide its manifest in Listing. 1. This manifest describes the XWiki application (Listing. 1, line 4-11) and its hosting environment (line 12-24). The name of the XWiki application (line 4) and the label of the version to deploy (line 6) are specified. Content-type of XWiki 1.0 deployable is a Web application archive (line 7). There is a set of XWiki instances to run on the targeted PaaS (line 8-9): *XWikiInstance1* is defined as the default instance (line 8).

All these instances have to be hosted and executed in a Java Web environment (line 12) instantiated from *JavaEnvTemplate* (line 13-23). The link between the application and its environment is expressed in the environment attribute of the application element (line 4). The defined template *JavaEnvTemplate* is composed of two PaaS resource nodes: An Apache Tomcat as Web container (line 15) to host the XWiki application and a MySQL database instance (line 16) for storing persistent data. A script to set a binding between these resources is also specified (line 18). In addition to that, an environment variable required by the container is provided (line 21).

IV. Overview of the COAPS API

The COAPS API is based on the Representational State Transfer (REST) architecture which is an architectural style for building distributed systems. It provides a simple and powerful model for organizing complex applications into simple resources [9]. The REST architectural style is based on resources associated to unique identifiers (e.g. URI). The interactions with these resources are based on a standardized communication protocol (e.g. HTTP).

COAPS API handles application and environment resources which are described in accordance to our defined description models (See Section III). It exposes a set of generic and RESTful HTTP operations (i.e. GET, POST, PUT and DELETE) for cloud applications management and provisioning. We organize these operations into two categories: application management operations and environment management operations. By application we mean any computer software or program that can be hosted and executed by a PaaS. The source archives of the application is provided by the COAPS API human or software agent. By environment we mean the set of required software components needed by an application: i.e. runtimes (e.g. java 6, java 7, etc.), frameworks/containers (e.g. Spring, Tomcat, etc.), services (e.g. databases, messaging, etc.), etc. In the following, we present the different environment and application management operations offered by our API respectively in Section IV-A and IV-B.

A. The Environment Management Resource

A resource based representation of the proposed environment management operations is provided in Fig. 6. Each box represents an environment resource (or sub-resource), the title text (e.g. `/environment`, `/environment/envId`, etc.)

```
<?xml version="1.0" encoding="UTF8"?>
<manifest name="XWikiApplicationManifest" xmlns="http://www.compatibleone.fr/schemes/paasmanifest.xsd">
  <description>This manifest describes The XWiki Servlet.</description>
  <application name="XWikiApplication" environement="JavaWebEnv">
    <description>XWiki application description.</description>
    <application_version name="version1.0" label="1.0">
     <deployable name="XWiki.war" content_type="artifact" location="Folder/URL" multitenancy_level="
         Shared"/>
     <application_version_instance name="XWikiInstance1" initial_state="1" default_instance="true"
         memory="256" disk="1"/>
     <application_version_instance name="XWikiInstance2" initial_state="1" default_instance="false"
         memory="128" disk="1"/>
    </application_version>
  </application>
  <environment name="JavaWebEnv" template="JavaEnvTemplate">
    <environment_template name="JavaEnvTemplate" memory="2048" disk="2">
      <description>TomcatServerEnvironmentTemplate.</description>
      <environment_node content_type="container" name="tomcat" version="6.0.36" provider="CF"/>
      <environment_node content_type="database" name="mysql" version="4.2" provider="OS"/>
      <environment_relation>
        <environment_link name="dbBinding" script="bind.sh"/>
      </environment_relation>
      <environment_configuration>
        <environment_variable name="CATALINA_HOME" value="$catalina_home"/>
      </environment_configuration>
    </environment_template>
  </environment>
</manifest>
```

Listing 1. XWiki application manifest

represents the resource identifier and the body text lists the offered operations by this resource (e.g. `FindEnvironments`, `CreateEnvironment`, etc.) and its associated REST methods (e.g. GET, POST, etc.).

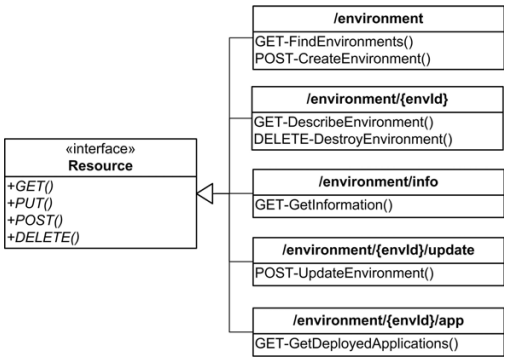

Fig. 6. The COAPS API environment management operations

In our specifications, we consider the basic operations for an application's environment creation and management. The environment management resource offers the following operations:

- *Create Environment*: creates a new environment using the environment element of the application manifest (see Section III). The operation returns, among others, an environment ID.
- *Update Environment*: updates an existing environment. An environment ID must be provided and the updates must be specified in a new application manifest.
- *Destroy/Describe Environment*: destroys/describes an environment given its ID.
- *Find Environments*: lists all available environments.
- *Get Deployed Applications*: lists all deployed applications in an environment given its ID.
- *Get information*: lists the runtimes, frameworks and services supported by the targeted PaaS.

B. The Application Management Resource

As for the environment management resource, we consider the basic operations for an application provisioning and management. Our application management resource is represented in Fig. 7.

The application management resource offers the following operations:

- *Create Application*: creates a new application using the application element of the application manifest (see Section III). The operation returns, among others, an application ID.
- *Deploy Application*: deploys an application identified by its ID on an existing environment identified by its environment ID.
- *Start/Stop/Restart/Un-deploy/Destroy Application*: starts/stops/restarts/un-deploys/destroys a deployed application given its ID.
- *Update Application*: updates an existing application. The application ID must be provided and the updates have to be specified in a new application manifest.
- *Describe Application*: returns an application description given its ID.
- *Find Applications*: lists the available applications.

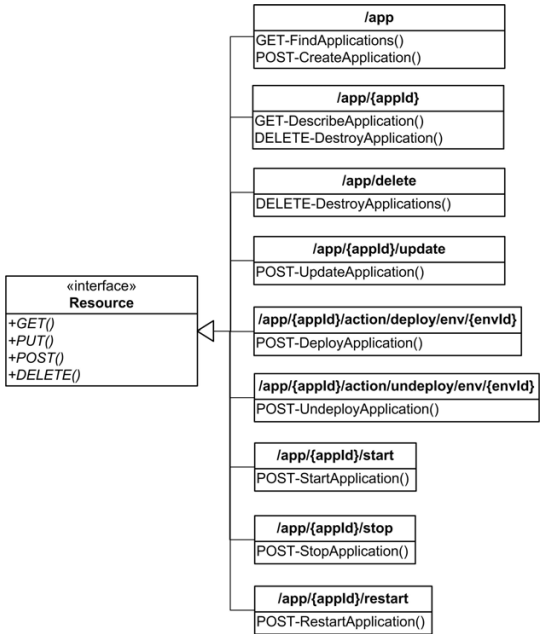

Fig. 7. The COAPS API application management operations

- *Destroy Applications*: destroys all existing applications.

The full version of the COAPS API specification is available at: http://www-inf.it-sudparis.eu/SIMBAD/tools/COAPS.

V. Implementation

Currently, we provide two implementations of our API: a Cloud Foundry implementation (CF-PaaS API) and an Openshift implementation (OS-PaaS API). Both implementations are developped in Java and provided as RESTful[8] Web applications. We also developed a generic Web client (see Fig. 8) for application provisioning and management on PaaS with an implementation of our API. The Web client acts as an access point for the API implementations and allows a user to call the environment/application management operations (see Section IV). Through this client, we show that our API allows a seamless (i.e. using (1) the **same** application/environment manifests and (2) the **same** actions) PaaS application provisioning.

We recall our XWiki provisioning example (see Section II-A) to illustrate how our solution works in practice. To deploy the XWiki application, we start by creating its manifest, written in XML, according to our PaaS application description model. The used XWiki application description manifest is provided in Listing 1. We recall that this description is independent from the targeted PaaS (in our case Cloud Foundry) and can be used to provision another PaaS with the same XWiki application.

This manifest and the application's deployable (a WAR file in this example) are the only inputs required by the

[8]In our implementations we use the Jersey JAX-RS (JSR 311) implementation.

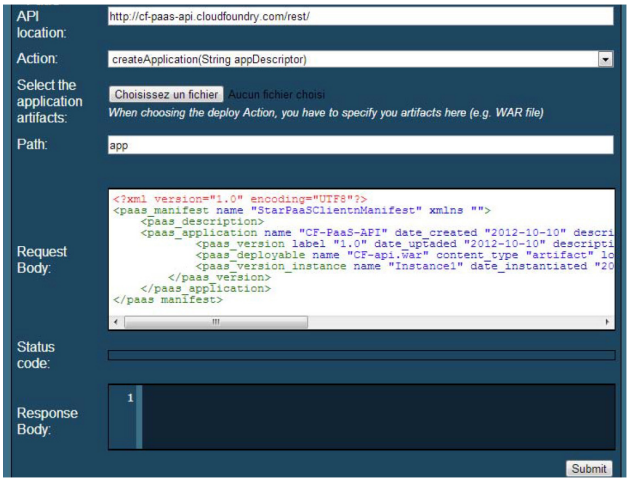

Fig. 8. The COAPS API Web client

COAPS API for the application provisioning. Using our generic Web client (see Fig. 8), the XWiki provisioning occurs as follows: First of all, we specify the COAPS API implementation that will be used. In this example we adopted a Cloud Foundry implementation acting as a middleware to a hybrid Cloud Foundry instance hosted by our institute (i.e. Telecom SudParis). Next, we execute one by one the required environment and application operations (listed by the Web client in a drop-down list). To provision the XWiki application required operations are: `create Environment`, `create Application`, `deploy Application` and `start Application`. Finally, the application is running and accessible on the URL returned by the `start Application` operation. At this stage, our Web client can be used to manage (i.e. stop, restart, update, etc.) the provisioned application.

Both COAPS API implementations, the Web client implementation and a video illustrating the XWiki deployment (see Section II-A) are available at the COAPS API page[9]. Our generic Web client is also available on line at: http://star-paas-client.cloudfoundry.com/ and can be used to test our CloudFoundry-PaaS API implementation deployed at: http://cf-paas-api.cloudfoundry.com/. A user guide and test resources are available in the project's folder[10].

VI. Related Work and Comparison

There are many attempts to provide a description model that covers the PaaS resources (i.e. containers, runtimes, frameworks and applications), and to offer efficient APIs to manage them. We provide in Section VI-A an overview of the existing approaches of PaaS resources description models. Then, we investigate the different PaaS resources management APIs in Section VI-B.

[9]http://www-inf.it-sudparis.eu/SIMBAD/tools/starPaaS/
[10]http://www-inf.it-sudparis.eu/SIMBAD/tools/COAPS/GenericAPI.zip

A. PaaS Resources Description Models

The Amazon CloudFormation [10] allows specifying the resources needed by an application. Using this tool, an application developer can specify that its application requires a specific number of computing, storage and networking resources. CloudFormation proposes to use existing sample templates, or to create user specific templates. These templates describe the needed resources and their dependencies. They could be deployed and updated via a command line tool.

Topology and Orchestration Specification for Cloud Applications (TOSCA) [11] provides an XML based language to describe PaaS applications as a set of Nodes with well defined Relationships. The Nodes and Relationships are described in a Service Template document. This latter, contains the details needed to set up the environment and its artifacts. The Service Template contains a Topology Template that describes the Relationships between all the Nodes of the application. It contains also, a Plan element that describes the operational management behavior. All the needed elements for a TOSCA application are encapsulated in a predefined archive format called CSAR.

Cloud Application Management for Platforms (CAMP) [12] represents PaaS as a set of Application Components related to each others and using Platform Components via Assemblies resources. An Assembly resource represents a running application. In this specification, the Platform is described as a set of Platform Components offering a list of capabilities to be used by applications. An application is a set of Application Components having capabilities and requirements. An Application Component can be related to a Platform Component if this latter has the needed capabilities that could be associated to the requirements of the Application Component. The application is described in a Platform Deployment Package (PDP for short) containing all the descriptions and dependencies of the application (i.e. manifests, deployment plan, certificates, bundles, etc.).

In [13], authors propose to use existing approaches to describe applications and their deployment in the cloud. They use resource templates (as for TOSCA and CloudFormation) representing reconfigurable entities that can be reused for different applications. Automated deployment of the resources associated to templates description, can be possible using Deployment recipes (using DevOps technologies like Chef or Puppet).

Except CloudFormation, which is a proprietary solution, all other solutions are not used in the cloud community. They still subjects of discussions and modifications. TOSCA and CAMP perspectives are promising. Our proposed Platform Application Description Model contains all the information included in a TOSCA Service Template or in a CAMP Assembly Template. In addition, our description model is based on the OCCI standard. Finally, our solution is already adapted in national and international projects (see SectionII), that proves its usefulness and richness.

B. PaaS Applications Provisioning APIs

To provide a generic PaaS API, we studied different approaches. We cite among others Amazon PaaS API AWS Elastic Beanstalk [14], which allows the provisioning and management of applications running on Amazon Cloud instances. Azure Service Platform API [15] can be used to deploy applications on Azure Cloud instances. Google App Engine API [16] allows deploying and scaling applications on Google infrastructures. Salesforce API [17] provide a Development service that allows developing new services using storage or business process engines. Red Hat API [18] enables the management of PaaS resources (containers, storage services, business process, etc.) and applications.

All of these APIs, allow the description of an application and its environment in different manners. However, almost all of these attempts are proprietary APIs. They suffer from the vendor lock-in problem, because they just consider a specific model of resource representation. These solutions have many difference in resources modeling, used languages and frameworks.

In [19], author discussed the need for a generic API that enables cloud users to specify their requirements among providers' offers. Their investigations show that almost all the APIs use similar concepts with similar properties and actions but with different names and structures. The authors consider that the interoperability problems arise due to different modeling and notation of the same features across different cloud providers. To handle this issue, the author considers semantic technologies as a solution for interoperability in the cloud.

Other works are attempting to benefit from the similarities between cloud resources representation and management to provide a generic way to perform this management.

In [20], authors suggest that "*a common API should involve a set of core functionalities that will meet the basic needs of any Cloud Platform and will unify all different APIs (an API for all APIs).*" Accordingly, a common API is proposed with a common semantics for the needed PaaS resources and actions. This API communicates with PaaS providers via adapters, and any new provider has to adapt his offering following the same semantics (i.e. using the same models and structures or providing an adapter to transform its own representation to the common one). To deploy an application, developers should provide an application profile that describes the requirements of the application. A management module builds an application deployment descriptor according to the selected PaaS, and then, it initiates the application deployment via a standard API (Cloud4SOA) that uses the dedicated adapter for the selected PaaS offer.

CAMP [12], provides a restful solution to manage the lifecycle of an application. To deploy an application, the user needs to register the associated PDP (see Section VI-A) using a HTTP POST request to the Platform. This request must contain the URI of the concerned PDP. The platform creates an Assembly Template that represents the deployed application. To run the application, a client sends a HTTP POST request

to the corresponding Assembly Template. This later creates an Assembly instance that represents the running application. The client can update or suspend a running application by sending the new state in a HTTP POST request. To delete an application, the client sends a HTTP DELETE request to the Assembly.

Inspired by the idea suggested in [20], we studied the different PaaS offerings to benefit from the similarities between the existing APIs. Our COAPS API offers a generic interface for application provisioning and management independently from PaaS. New PaaS services could be consumed via our API by simply providing an associated implementation. Unlike other approaches, our API is independent from an application description model and other description models can be used.

VII. Conclusion

In this paper, we presented a PaaS-independent solution for PaaS application provisioning and management. We provide a unified description model that allows to represent an application and its requirements independently of the targeted PaaS. We also defined a generic PaaS application provisioning and management API (COAPS API). Our proposed solution applies the separation of concerns principle by separating the provisioning and management API from the application description model. This choice makes both contributions, i.e. the description model and the API self-contained and allows their independent use. This solution is currently used in two collaborative research projects (i.e. CompatibleOne and EASI-CLOUDS) involving academic and industrial partners to resolve research issues coming from real world cases.

As future work, we plan to propose a generic API for PaaS application monitoring and management. Monitoring includes retrieving information related to the current state of the PaaS resources and applications, while management includes fault tolerance, application migration or replication, and scaling up and down running applications. Such API is indeed challenging, because it has to face the complexity of monitoring and management tasks amplified by the heterogeneity of PaaS level and the huge number of parameters to take into account. In order to do that, we need to investigate the different monitoring and management solutions in the cloud to see whether it is possible to define a generic model that enables the retrieval of monitoring information and the execution of management operations.

Acknowledgment

This work was funded in part by CompatibleOne, a research project publicly funded by French Ministry (FUI), and in part by the ITEA 2 research project EASI-CLOUDS.

References

[1] The Mckinsey Quarterly, "How IT is managing new demands: McKinsey Global Survey results," 2010.

[2] Salesforce.com. (2013) Force.com Apex Code Developer's Guide. http://www.salesforce.com/us/developer/docs/apexcode/salesforce_apex_language_reference.pdf.

[3] J.-P. Laisne, I. J. Marshall, and P. Peiravi, "Next-Generation Cloud Management. The CompatibleOne Project," *Intel's Journey to Cloud*, vol. 2, no. 1, pp. 15–24, 2012.

[4] A. Thiele. (2012) The EASI-CLOUDS Project Description. http://easi-clouds.eu/2012/02/03/project-description/.

[5] EASI-CLOUDS. (2012) Project Poster. http://easi-clouds.eu/wp-content/uploads/2012/11/EASI-CLOUDS_poster_2012_v4.pdf.

[6] T. Aho, A. Ashraf, M. Englund, J. Katajamki, J. Koskinen, J. Lautamki, A. Nieminen, I. Porres, and I. Turunen, "Designing IDE as a Service," *Communications of Cloud Software*, vol. 1, pp. 1–10, 2011.

[7] J. Lautamäki, A. Nieminen, J. Koskinen, T. Aho, T. Mikkonen, and M. Englund, "CoRED: browser-based Collaborative Real-time Editor for Java web applications," in *CSCW '12 Computer Supported Cooperative Work, Seattle, WA, USA, February 11-15, 2012*. ACM, 2012, pp. 1307–1316.

[8] S. Yangui and S. Tata, "CloudServ: PaaS resources provisioning for service-based applications," in *The IEEE 27rd International Conference on Advanced Information Networking and Applications, AINA 2013, March 25-28, Barcelona, Spain*, 2013.

[9] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[10] Amazon. (2013) Amazon's CloudFormation. http://aws.amazon.com/cloudformation/.

[11] TOSCA Technical Committe, "Topology and Orchestration Specification for Cloud Applications," OASIS, Tech. Rep., November 2012.

[12] M. Carlson, M. Chapman, A. Heneveld, S. Hinkelman, D. Johnston-Watt, A. Karmarkar, T. Kunze, A. Malhotra, J. Mischkinsky, A. Otto, V. Pandey, G. Pilz, Z. Song, and P. Yendluri, "Cloud Application Management for Platforms," http://www.cloudspecs.org/paas/, OASIS, Tech. Rep., 2012.

[13] B. Satzger, W. Hummer, C. Inzinger, P. Leitner, and S. Dustdar, "Winds of Change: From Vendor Lock-In to the Meta Cloud," *IEEE Internet Computing*, vol. 17, no. 1, pp. 69–73, 2013.

[14] AWS Elastic Beanstalk. (2010) API Reference. http://www.wilsonmar.com/arc%5Caws%5Cec2-dg.2009-04-04.pdf.

[15] D. Chappell, "Introducing the Azure Services Platform an Early Look at Windows Azure, .Net Services, SQL Services, and Live Services ," *David Chappell & Associates*, 2008.

[16] Google App Engine. (2011) Developer's Guide. http://code.google.com/intl/el-GR/appengine/docs/.

[17] Salesforce. (2011) Web Services API Developer's Guide. http://www.salesforce.com/us/developer/docs/api/apex_api.pdf.

[18] Red Hat. (2010) Red Hat PaaS: Bringing Open Choice & Application Portability to the Cloud. http://www.jboss.com/pdf/RedHatPaaSWhitepaper.pdf.

[19] N. Loutas, V. Peristeras, T. Bouras, E. Kamateri, D. Zeginis, and K. Tarabanis, "Towards a Reference Architecture for Semantically Interoperable Clouds," in *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, 30 2010-dec. 3 2010, pp. 143–150.

[20] N. Loutas, "Cloud4SOA: Requirements Analysis Report," Cloud4SOA, Tech. Rep., February 2011.