CompatibleOne: The Open Source Cloud Broker

Article *in* Journal of Grid Computing · November 2013

Impact Factor: 1.51 · DOI: 10.1007/s10723-013-9285-0

Sami Yangui
Concordia University Montreal
23 PUBLICATIONS **97** CITATIONS

Samir Tata
Institut Mines-Télécom
115 PUBLICATIONS **722** CITATIONS

Noname manuscript No.
(will be inserted by the editor)

CompatibleOne: The Open Source Cloud Broker

Sami Yangui · Iain-James Marshall · Jean-Pierre Laisne · Samir Tata.

Received: date / Accepted: date

Abstract The adoption of Cloud computing as a new business model has induced the proliferation of several Cloud service providers. Cloud end users are then faced with choosing the appropriate provider offers in terms of supported technologies, geographic locations, security, access rules, billing, etc.

In this paper, we propose a new Cloud broker called CompatibleOne which provides solutions to assist Cloud end users in their providers choice. The CompatibleOne broker is based on open standards, mainly CDMI and OCCI, and uses our new defined object-based description model called CORDS. CORDS serves to model and manage the various Cloud resources that manipulates the main CompatibleOne platform called ACCORDS. We motivate our solution with real use case scenarios and an implementation to show its feasibility.

The work presented in this paper was partially supported by the French FUI CompatibleOne project.

Sami Yangui
Institut MINES-TELECOM, TELECOM SudParis, UMR CNRS Samovar.
9, rue Charles Fourier 91011 Evry Cedex, France.
Tel.: +33-0-160764502
Fax: +33-0-160764780
E-mail: Sami.Yangui@it-sudparis.eu

Iain-James Marshall
PROLOGUE.
ZA de Courtaboeuf. 12, Avenue des Tropiques 91943 Les Ulis Cedex, France.

Jean-Pierre Laisne
BULL SAS.
2, Rue Galvani, Massy 91300, France.

Samir Tata
Institut MINES-TELECOM, TELECOM SudParis, UMR CNRS Samovar
9, rue Charles Fourier 91011 Evry Cedex, France.

Keywords ACCORDS · Cloud broker · Cloud service provisioning · CORDS · OCCI.

1 Introduction

Cloud Computing economic model became mature and is increasingly used [1]. It consists on a new model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, services, etc.) that can be rapidly provisioned and released with minimal management effort and lower cost. It obeys to the pay as-you-go model of Cloud Computing [36]. The completion and the achievement of this new model has induced the proliferation of not only Cloud end users but Cloud service providers also [2]. The diversification of Cloud service providers has generated the diversification of their offers in terms of provider resources, security and access rules, billing models, etc. To deal with this, end users are faced to a real challenge to choose the appropriate Cloud provider. This choice should be motivated mainly by the provider features i.e. security rules, cost optimization and/or compatibility with the end users' requested technologies, etc.

Gartner VP Thomas Bittman stated at the 2008 Gartner Data Center Conference in Las Vegas USA that Cloud computing will eventually support thousands of specialized providers, creating the need for a cottage industry of specialists to assemble client solutions from a panoply of Cloud offerings. *"In the future, we expect to see thousands of providers in the Cloud,"* with services being *"put together like Lego blocks"*, Bittman affirmed. *"We are moving from this monolithic, one provider does everything model to an ecosystem. We are moving toward a more distributed, open world, and to-*

ward more customized services. We believe there will be a large number of mid-sized providers".

In this context, Cloud brokers act as intermediaries between Cloud providers and Cloud end users. Brokers help companies to choose the right providers, deploy services across multiple Clouds, and even provide Cloud arbitrage services that allow end users to shift between platforms to capture the best pricing for example.

In this paper, we present CompatibleOne[1], an open source broker, which provide interoperable middleware for the description and federation of heterogeneous Clouds and resources provisioned by different Cloud providers. CompatibleOne allows users and developers to combine different services available from different suppliers. It supports different kinds of Cloud resources (e.g. infrastructure, platform, application, etc.) that can be provided as-a-service. It helps developers and users to avoid vendor lock-in, enforce SLAs and reduce costs. Unlike existing brokerage systems, CompatibleOne could be considered as an advanced Cloud resource management and automatic provisioning software environment. In fact, some works have focused on the realization of Cloud brokers. We can cite among others SpotCloud for intermediation use case [20] and Unified Cloud Interface Project, for intermediation, aggregation and arbitrage use cases [14]. However, these works imposes strong constraints to providers by forcing them to integrate brokering mechanisms and/or do not provide a common resources description model as does CompatibleOne.

Indeed, CompatibleOne is a model and an execution platform. On the one hand, the model called CompatibleOne Resource Description System (CORDS for short), is an object based description of Cloud applications, services and resources. On the other hand, the execution platform called Advanced Capabilities for CORDS (ACCORDS for short), is a Cloud application provisioning and deployment control system. Furthermore, CompatibleOne obeys to a communication model based on OCCI categories and REST interfaces.

A Gartner report outlined three primary roles for a Cloud broker (i.e. aggregator, integrator and customizer) [13]. In this report, three main use cases for brokerage service [12] are idetified:

- Intermediation: intermediation for multiple services to add value-adds like identity management or access management,
- Aggregation: static aggregation of multiple Cloud services into one or more new services,
- Arbitrage: dynamic version of aggregation which aims to provide flexibility and opportunistic choices for the service aggregator.

CompatibleOne checks these three use cases. Moreover, it is aligned with the Cloud Computing Reference Architecture of the National Institute of Standards and Technology (NIST) [3]. It is powered by service brokering capabilities that offers services from Cloud providers based on open standards, mainly Cloud Data Management Interface (CDMI) and Open Cloud Computing Interface (OCCI) [17].

CDMI is a SNIA standard that specifies a protocol for self-provisioning, administering and accessing Cloud storage. It defines the functional interface that applications will use to create, retrieve, update and delete data elements from the Cloud. As part of this interface the client will be able to discover the capabilities of the Cloud storage offering and use this interface to manage containers and the data that is placed in them. In addition to that, metadata can be set on containers and their contained data elements through this interface [16].

OCCI is a specification that defines a meta-model for Cloud resource and a RESTful protocol for management tasks. It offers a flexible API with a strong focus on interoperability while still offering a high degree of extensibility. OCCI is suitable to serve many other models in addition to IaaS, including PaaS and SaaS. To enhance modularity and extensibility, OCCI is released as a suite of complimentary documents such as OCCI core [24], OCCI renderings [4] and OCCI extensions (e.g. OCCI Infrastruture extension [23]).

The rest of this paper is organized as follows. Section 2 discuss related work and show contributions of our solution. Section 3 presents the CompatibleOne logical model for describing Cloud resources at the IaaS and PaaS levels. Section 4 presents a high-level overview of the CompatibleOne platform architecture in four quadrants which represent the four steps of the functional cycle for Cloud service provisioning. Section 5 enumerates CompatibleOne services that have been developed around the broker. Section 6 introduces a realistic CompatibleOne use case (i.e. XWiki application deployment). Section 7 conclude the paper and describe directions for future work.

2 Related work

Cloud computing providers can setup several data centers at different geographical locations over the Internet in order to optimally serve needs of their customers around the world. However, existing systems do not

[1] http://www.compatibleone.org/bin/view/Main/

support mechanisms and policies for dynamically coordinating load distribution among different Cloud-based data centers in order to determine optimal location for hosting application services to achieve reasonable QoS levels for example. Thus, Cloud brokers are needed for such a federated Cloud computing environment to enable just-in-time, opportunistic and scalable provisioning of applications, services, resources and networks.

In this context, some works have focused on the realization of a recomandation system to assist and advice Cloud end users to select the appropriate provider. In [30], the authors presents their realized recommendation system (RS) for Cloud computing platforms. This approach is most suitable for design-time decisions as it is used statically to provide a ranking of available Cloud providers.

In [31], the authors introduces STRATOS broker service. STRATOS supports applications deployment and runtimes management. It allows the application deployer to specify what is important to them in terms of KPIs, transform these requirements on a set of resource provisioning requests before considering these requests against all providers' offerings to select the best offer which is aligned with the objectives. However, to express application requirements, STRATOS uses a rudimentary way which consists on providing simply the application configuration specifications at deployment time.

In [28], the authors proposes a broker implementation with agent-based simulation. This broker processes a form of financial derivative contract delivered by Cloud providers called an option. It uses the uptake of the option contracts to decide if he should invest by buying resource access, during determining periods, from this provider or not. The resources can then subsequently be provided to clients who demand them. This system does not really broker demands to Cloud providers but it invests in leasing resources to provide to consumers the best option obeying to their criteria. This work is based on a WZH brokering model introduced in [27]. The WZH model uses a third-party intermediary, called the coordinator, which uses a variety of Cloud assets to deliver resources to end users at a reduced price, while making a profit and assisting providers in resource forecasting. The coordinator acts then as a broker.

In addition to academic works, we can mention several research projects which proposed initiatives for Cloud services aggregation and intermediation such as InterCloud [19], contrail project [21], mOSAIC project [32] and BonFIRE [26].

The scope of InterCloud is to provide open interfaces devoid of any proprietary vendor version control at operational level such that there is a facility for all interconnected Clouds to have a common control of how applications should be deployed. That implies that there could be a common protocol and exchange mechanism for each Cloud, based on requisite parameters such as key resources including applications, data, bandwidth, load balancing, costs of processing, security, service level and user location [19].

The Contrail project aims to design, implement, evaluate and promote an open source computational Cloud wherein users can limitlessly share resources [22]. The Contrail vision is a federation of resources provided by public and private Clouds. Offered Cloud resources should be integrated into a single homogeneous federated Cloud that users can access seamlessly. Any organization should be able to be both a Cloud provider, when its IT infrastructure is not used at its maximal capacity, and a Cloud customer in periods of peak activity [21].

The mOSAIC project defines an ontology [37] and an open source API [35] to describe and manage Cloud resources offered by providers. mOSAIC attempts to improve interoperability among existing Cloud providers for both Cloud developers and end users.

The BonFIRE (Building service testbeds for Future Internet Research and Experimentation) project built a multi-site Cloud prototype FIRE facility to support research across applications and services on future Internet [25]. BonFIRE introduces experimenters and test bed tools to provide statistics access to heterogeneous Cloud resources with advanced low-level control and monitoring APIs. Based on these statistics, Clouds users can find out which Cloud provider satisfies their constraints and therefore with which they should interact [26].

The analysis of these studies reveals all the efforts for the establishment of reliable and standard brokering mechanisms in the Cloud to facilitate consumers access to resource providers and assist them in their choice. However, some works, such as BonFIRE and Contrail, imposes strong constraints to providers by forcing them to integrate brokering mechanisms which hinders their adoption. In addition to that, the Cloud provider recommendation systems do not provide any mechanisms to automate the resource acquisition process and offer advice only. The STRATOS broker assumes that all users may provide application configuration specifications in an a common and understandable way but the authors have not defined a common description model and/or a template to use when providing the configuration specifications.

CompatibleOne is an open source broker (sources are available at [29]) that provides interoperable middleware for the description and federation of heterogeneous Clouds and resources provisioned by different Cloud providers. CompatibleOne broker does not force to change anything on the side of provider, it intervenes on the side of the consumer, taking into account its vision and its specific application requirements. CompatibleOne is "application-oriented" and even "business-oriented" Cloud system. CompatibleOne fulfills the three primary roles for a Cloud service broker: aggregator, integrator and customizer identified by Gartner [13]. In addition to that, by using our defined CORDS description model to describe IaaS and PaaS resources, we can easily swhich brokerage and provisioning from one Cloud provider to another regardless supported implementations, technologies and communication protocols of both Cloud providers and Cloud end users. CompatibleOne is then a key element for federation and interoperability in Cloud systems.

3 CORDS: CompatibleOne logical model

The CompatibleOne Resource Description System is an object-based description model of Cloud resources. The CORDS model is based on the OCCI[2] standard. We designed CORDS model in order to be consistent and compliant with OCCI interfaces. Our choice of this standard was motivated by the simple approach of OCCI to describe Cloud resources at different levels and its ability to ensure systems interoperability. Indeed, we needed to design a model which would help to make abstraction of any provisioning system, of any cloud service. Such a model would enable a detailed description of complex workloads in order to provision them in an automated fashion on heterogeneous providers. Moreover, we needed that the CompatibleOne platform be interoperable with other services in a way to allow other CompatibleOne ecosystem actors to inter operate, to integrate and possibly to create added value services. To achieve this goal, we needed an object-oriented model specifically designed for cloud computing and flexible enough to be enriched with new classes. For all of these reasons, we chose the OGF OCCI open standard. In fact, to the best of our knowledge, OCCI is the only standard model compliant with these criteria. In addition to that, it offers a discovery capability that allows any client to discover services' categories that OCCI-based servers are capable to deliver.

Fig. 1 and Fig. 2 shows respectively IaaS and PaaS logical descriptive data models. They shows respectively how IaaS and PaaS resources can be described through CORDS model. Each box on these views schematises a potential resource; each resource belongs to an OCCI category. Based on this, we can make CORDS manifests which describes IaaS (respectively PaaS) resources. An IaaS (respectively PaaS) manifests are descriptors which describes and/or references IaaS (respectively PaaS) resources in accordance to the categories and hierarchy defined in the CORDS model.

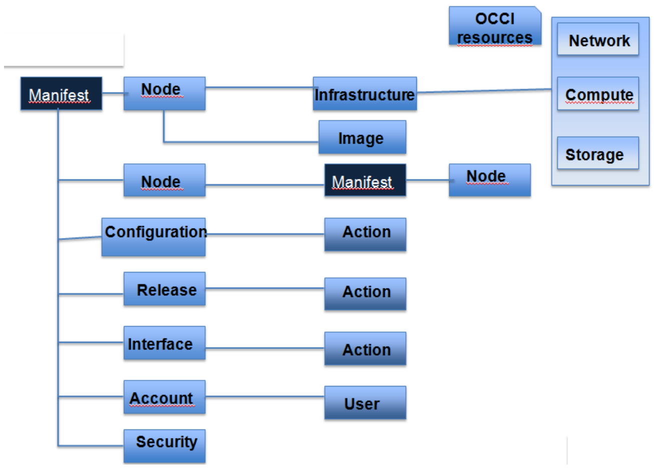

Fig. 1 CompatibleOne Resource Description System: IaaS logical data model.

Specifically, an IaaS CORDS manifest is described using *Nodes*. Each *Node* is defined as a unit, which is described in terms of its *Infrastructure* and services *Image* (See Fig. 1). *Infrastructure* entity provides *Network* (e.g. bandwidth), *Compute* (e.g. CPU power) and *Storage* (e.g. disk capacity) requirements in accordance to the OCCI IaaS extension [23]. The *Image* entity provides the operating system description (e.g. Ubuntu, Fedora, etc.) and a set of required packages.

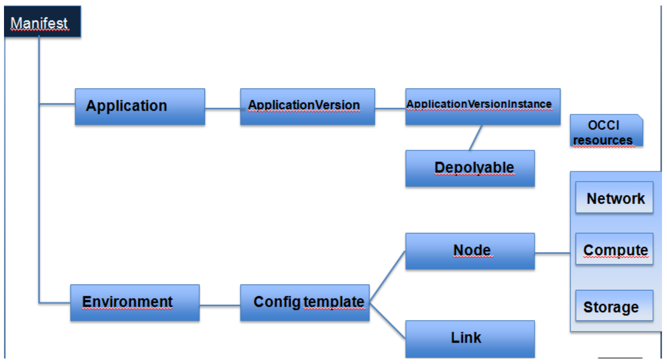

Fig. 2 CompatibleOne Resource Description System: PaaS logical data model.

A CORDS PaaS manifest describes an *application* and its hosting *environment* (See Fig. 2). By application we mean any computer software or program that

[2] http://occi-wg.org/

can be hosted and executed by a PaaS. Application *versions*, *instances* and *deployable* (source archives) are also described in this manifest. By hosting 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.

4 ACCORDS: Global architecture and functional cycle

The Advanced Capabilities for CORDS model (ACCORDS for short) is the CompatibleOne execution platform. ACCORDS offers services to provide Cloud resources from different IaaS and PaaS providers. Resource provisioning is performed in four steps (See Fig. 3). These steps are detailed in Section 4.1. Communication support facilities for intercations between these steps are detailed in Section 4.2.

4.1 Steps for service provisioning

Fig. 3 provides a high-level overview of the CompatibleOne ACCORDS platform architecture schematised in four quadrants. These quadrants represent the four steps of the functional cycle:

– Step One : Handling the user's requirements,
– Step Two : Validation and provisioning plan,
– Step Three : Execution of the provisioning plan,
– Step Four : Delivering the Cloud services.

Each one of these steps is necessary to provision both IaaS and PaaS resources. Each one of these steps are detailed in the following.

4.1.1 Step One: Handling the user's requirements

The first step deals with the user interactivity (first quadrant of Fig. 3). ACCORDS users specifies in this step their requirements in terms of IaaS and PaaS resources. Following this, CORDS manifests will be created. These manifests describes in detail required services to be delivered by Cloud providers, technical and economical criteria, specifications and constraints that are to be taken into consideration.

The Service Level Agreement Manager (SLAM) of the ACCORDS Platform provides an OCCI category model for the management of "negotiated" agreements. The negotiation process that leads to the creation of a negotiated agreement is currently to be outside of the scope of the ACCORDS Platform. Agreement documents are created using an SLA template production tool where by service description specificities, placement conditions and business valued guarantees are described using a combination of the standard WS Agreement and the CORDS service description model.

4.1.2 Step Two: Validation and provisioning plan

In the second step, CORDS manifests are transferred to the ACCORDS Parser (second quadrant of Fig. 3) which process and validate these documents. The parser builds a provisioning plan providing a precise blueprint for the description of required services. The Parser interact also with the ACCORDS publisher to ask if there is published Cloud providers capabilities that would satisfies the required services. More details according to the functioning of the ACCORDS publisher are provided in Section 6.1.

4.1.3 Step Three: Execution of the provisioning plan

The ACCORDS Broker is responsible of the execution of the provisioning plan (third quadrant in Fig. 3). The broker provides arbitrage and negotiation services useful to select appropriate providers in order to satisfy all requirements and constraints expressed in the provisioning plan (e.g. geographic locations, security, access rules, performance and/or billing, etc.). Specifically, the ACCORDS broker is responsible of processing the provided service level agreement to create an instance of service corresponding to the description provided by the blueprint or plan. The placement conditions guide the broker in its selection of the most appropriate providers for the provisioning of the described resources.

Aside from the broker, there are additional ACCORDS services that may be involved in this process. This depends on the requirements expressed in the service level agreement at step one. These services are developed by other CompatibleOne project partners. A non exhaustive list of these services is detailed in the following:

– CompatibleOne Security Service (COSS for short) secures the platform by implementing Transport Layer Security (TLS 1.0) to secure all access points with client and server certificate exchanges,
– CompatibleOne MONitoring Service (COMONS for short) : a REST OCCI server that starts just after COSS,
– CompatibleOne Placement Service (COPS) : allows the broker to determine the optimal placement of a treatment, required SLA and information provided by COMONS,

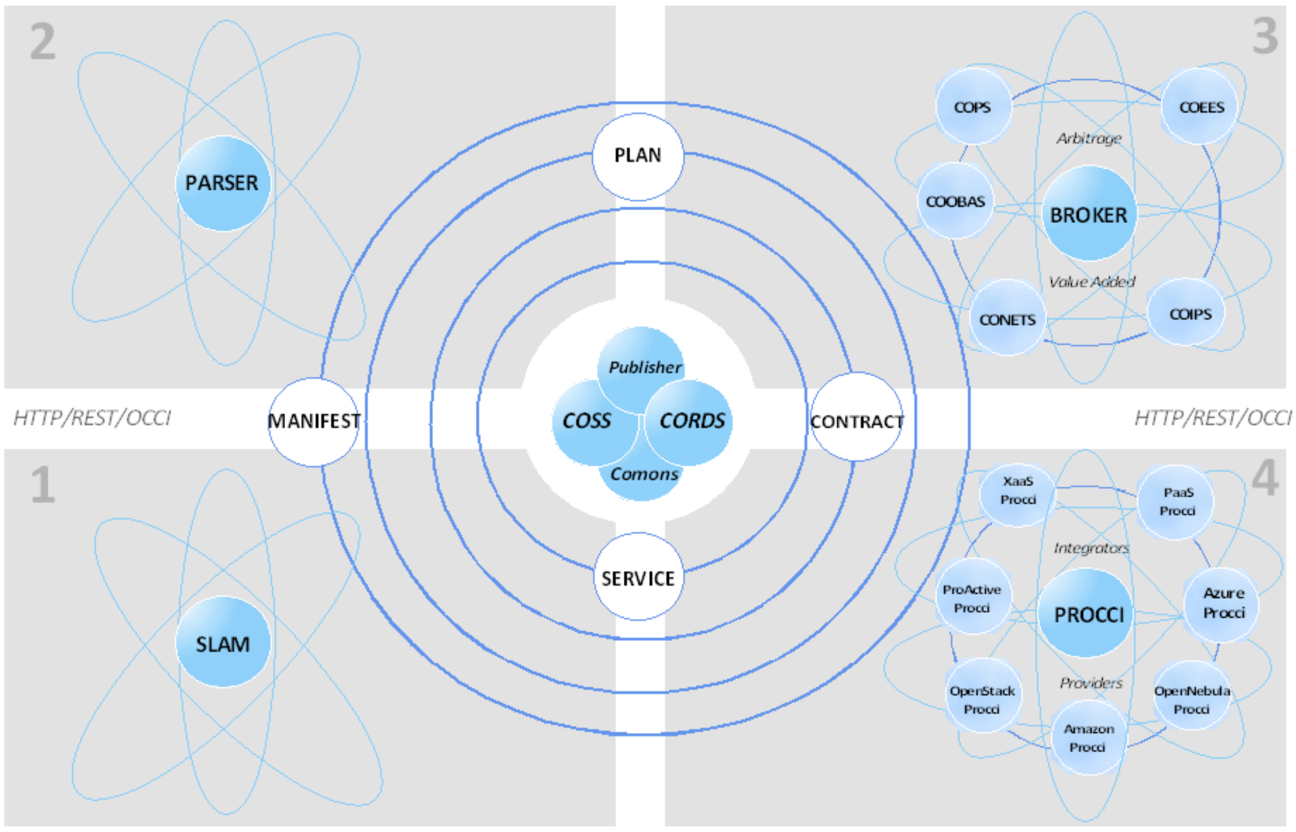

Fig. 3 ACCORDS: the Cloud brokerage platform.

– CompatibleOne Energy Efficiency Service (COEES) : allows the acquisition and the storage of consumption data to optimize energy consumption,
– CompatibleOne Ordering, Billing and Accounting Service (COOBAS) : allows commercial exploitation (costing, billing, etc.) and/or accounting information provided by COMONS,
– CompatibleOne Network Service (CONETS) : provides secure interconnection between resources and Cloud services. It allows allocation of dynamic address to connect consumers and suppliers and provides load balancing between providers via virtual networks,
– CompatibleOne Image Production Service (COIPS) : produces any required image format by a Cloud provider selected by the Broker and COPS.

More details about these services are provided in Section 5. The broker uses one or a set of these services, according to the end user demand, to aggregate provision plans to a set of a provisioning contracts. A contract is a document, derived from the plan, which describes the transaction to operate between the user who provide the initial CORDS manifest and the Cloud provider which handle the resource provisioning action through ACCORDS platform.

4.1.4 Step Four: Delivering the Cloud services

In the fourth step, ACCORDS Broker communicates obtained contracts to ACCORDS Proxies. Proxies in CompatibleOne are written Proccis in reference to their strong use of OCCI categories. The broker ensures the swhich and communicates each specific contract to its appropriate Procci implementation. For example, IaaS provisioning contracts will be redirected to Windows Azure Procci if the choice of the broker was carried on a Windows Azure provider in the second step.

4.2 Communication architecture and support facilities

CompatibleOne operates without service bus. Many options with service buses were being evaluated until it became evident for us that service buses were not necessary the best option and were redundant. The following three communication support facilities provide support to all interactions:

– Knowledge Base : A collection of information maintained within the system. The elements stored in the knowledge base are CORDS manifests, plans, contracts and services. A unique and universal identifier is attributed to each element in the knowledge base and may be used to reference the element for use in subsequent plans.

– CompatibleOne Security Services (COSS) : A highly security conscious system where everything is performed using Transport Layer Security (TLS). Each component in the system is required to obtain and present its own identity. Each component must also be authenticated and authorized to use the system.
– Publisher : Provides publication services, or repository services, for the connection between the components. All interactions within ACCORDS are performed in conjunction with the Publication Service offered by the Publisher, in compliance with COSS and using OCCI over HTTP [4].

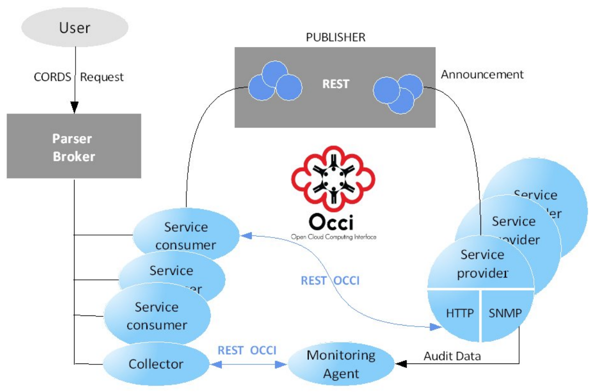

Fig. 4 ACCORDS Communication Architecture overview.

Fig. 4 shows the ACCORDS communication architecture. The end user submits a CORDS Manifest to the parser, which arrives then to the broker. Each required resource is considered as-a-service consumer. An arbitrage and a negotiation procedures are the engaged close to the publisher. The monitoring agent is under the control of a monitoring collector, or consumer, which collects the various audit and monitoring data in order to ensure that the provisioning is correctly performed and operated in compliance with the terms described in the CORDS Manifest or the SLA. Mapping and communication between service consumers and selected service provided is ensured through OCCI-based HTTP REST interfaces.

5 CompatibleOne services

The services listed around the broker in the third quadrant of Fig. 3 are detailed in this section. The above services assist the ACCORDS Broker in the provisioning operations required when using the plan.

5.1 CompatibleOne Security Service

CompatibleOne Security Service (COSS for short) secures the platform by implementing Transport Layer Security (TLS 1.0) to secure all access points with client and server certificate exchanges. An identity management can be integrated into the platform. This module is also responsible of data exchange security to allow secure transfer in peer-to-peer mode.

5.2 CompatibleOne MONitoring Service

CompatibleOne MONitoring Service (COMONS for short) is a REST OCCI server that starts just after COSS. It manages Agents, Probes and Consumers allowing the platform to monitor all required services and provisioned services. COMONS provides an information support to make decisions to other CompatibleOne services such as COPS, COEES and COOBAS services detailed in what follows.

5.3 CompatibleOne Placement Service

CompatibleOne Placement Service (COPS for short) allows the Broker to determine the optimal placement of a treatment based on the information contained in the CORDS Manifest, required SLA and information provided by COMONS.

Placement algorithms can be customized by the ACCORDS platform operator according to its needs and / or its business model. CompatibleOne is able to assure to the consumer that would, its treatments can be deployed on confidence Clouds *i.e.* implementing Trusted Compute Node technologies.

5.4 CompatibleOne Energy Efficiency Service

CompatibleOne Energy Efficiency Service (COEES for short) is provided by a software package allowing the acquisition and the storage of consumption data to optimize energy consumption. An experimental cluster is deployed and is instrumented to measure consumption with different solutions (OmegaWatt, Eaton, Schleifenbauer). With COEES and COPS, a consumer may require that these treatments are carried out on Clouds respecting "green computing" standards.

5.5 CompatibleOne Ordering, Billing and Accounting Service

CompatibleOne Ordering, Billing and Accounting Service (COOBAS for short) allows commercial exploitation (costing, billing, etc.) and/or accounting information provided by COMONS.

Thus, CompatibleOne manages not only technical aspects but also economic aspects demonstrating its relevance to the Cloud, which induces not only new technologies but also new business models.

5.6 CompatibleOne Network Service

CompatibleOne Network Service (CONETS for short) provides secure interconnection between resources and Cloud services. It allows allocation of dynamic address to connect consumers and suppliers, provide load balancing between providers via a virtual network, migrate service from a provider to another via a secure virtual network, and finally offer public IP addresses scalability.

5.7 CompatibleOne Image Production Service

CompatibleOne Image Production Service (COIPS for short) covers the entire image production within the same service. With COIPS, CompatibleOne platform is capable of producing any required image format by the Cloud provider selected by the Broker and COPS.

5.8 CompatibleOne Elasticity Service

CompatibleOne Elasticity Service (COES for short) implements a set of Cloud resources management, control and optimization algorithms. This service deals in accordance the user requirements expressed using CORDS and monitoring data provided by COMONS. These algorithms are able to adjust and optimize Cloud resources (compute, network and storage resources) based on users, applications and Cloud environment requirements.

In addition to these services, we can also mention EZVM VM interperability service which was also developed in CompatibleOne project. This service provides necessary technology to exploit ACCORDS components for instantiating virtual machines from different Cloud providers. Based on this, we can say that compatibleOne broker allows Cloud providers federation by ensuring a complete abstraction of Cloud platforms portability issues for the end user.

6 Use case : xWiki application deployment

To show the use and utility of the CompatibleOne broker in realistic situations, we present in this section a real use case i.e. XWiki[3] application provisioning. XWiki application 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.). This use case was performed as a demonstration at the project's final review. For the demonstration, the XWiki company, a project partner, provided us with a "Cloudified" version of the application. After starting the ACCORDS platform, the provisioning XWiki system is divided into two provisioning scenarios: An IaaS provisioning scenario for IaaS resources required to host XWiki application and then a PaaS provisioning resources for container and environment required to deploy and execute it.

6.1 ACCORDS platform authentification and startup

The ACCORDS Platform imposes an abstraction of identity in that a user is required to authenticate with credentials specific to the brokering platform. These credentials identify the payment account which may be shared between multiple federated brokering platforms. Provisioning credentials in the form of a subscription to a particular provider may be defined for and associated with a particular account. When private subscription credentials have not been attributed in this way then the default subscription credentials of the platform operator, for a particular provider, will be used. In the first case provisioning costs will be incurred by the original users. In the second case provisioning costs will be incurred by the provider to be subsequently billed to the end user.

ACCORDS platform starts through a dedicated script which brings up all the platform services and components (e.g. Parser, Broker, Proccis, etc.). Each component 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 [15]. These components are implemented as OCCI servers communicating through HTTP protocol (See Fig. 9). On the one hand, the choice of this architecture design allows us to have distributed platform components. These components can then be instantiated as needed to ensure the scalability and the good performance of the ACCORDS platform. On the other hand, the OCCI implementation of the

[3] http://enterprise.xwiki.org/xwiki/bin/view/Main/WebHome

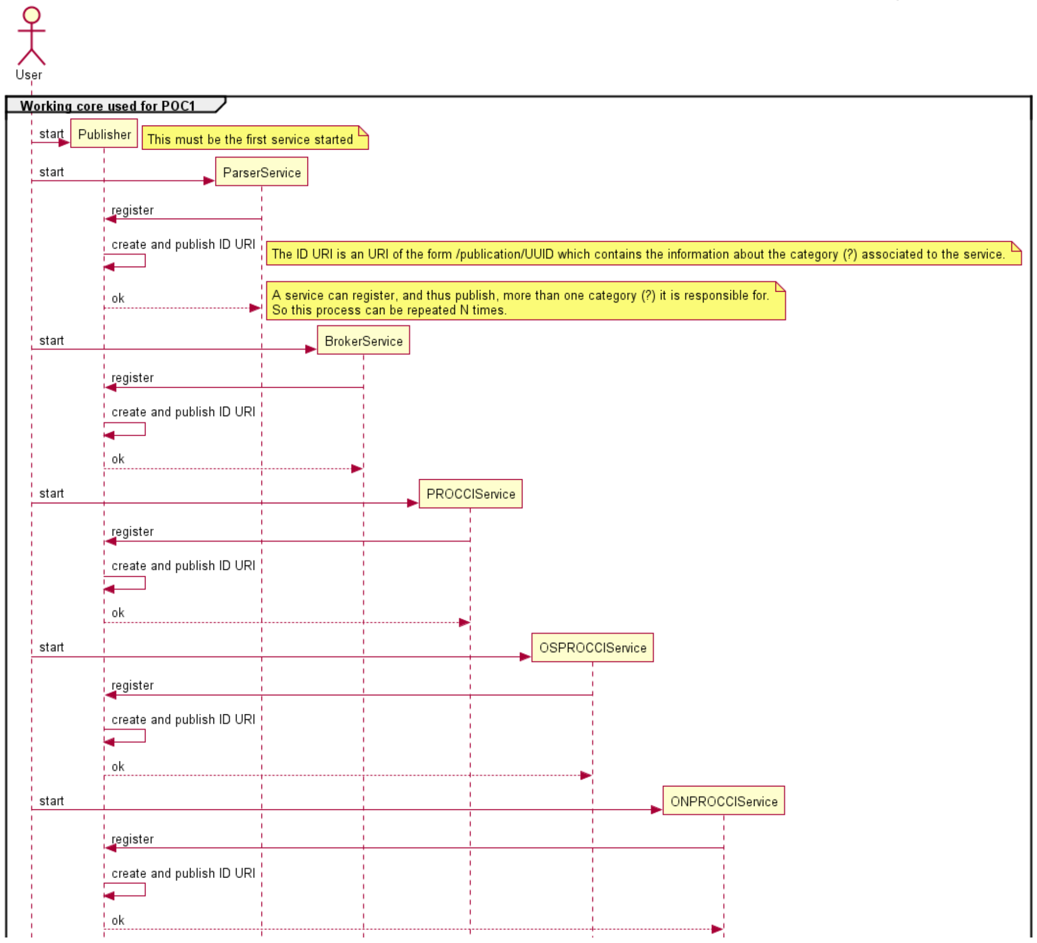
User
Working core used for POC1
start
Publisher
This must be the first service started
start
ParserService
register
create and publish ID URI
The ID URI is an URI of the form /publication/UUID which contains the information about the category (?) associated to the service.
ok
A service can register, and thus publish, more than one category (?) it is responsible for.
So this process can be repeated N times.
start
BrokerService
register
create and publish ID URI
ok
start
PROCCIService
register
create and publish ID URI
ok
start
OSPROCCIService
register
create and publish ID URI
ok
start
ONPROCCIService
register
create and publish ID URI
ok

Fig. 5 ACCORDS platform startup diagram.

platform servers ensures the extensibility and the interoperability of the platform in accordance to our OCCI choice motivations introduced in Section 3.

Fig. 5 shows a sequence diagram that details how the ACCORDS platform startup is performed. This diagram highlights the order of launch of platform services. The first service that should be started is the publisher which will be the place where all the other services can publish the categories they are able to handle. So, all platform's services, once started, must be registered on the publisher. For each registration request, the publisher creates an ID URI and publish it. It should be noted that in this diagram we take into account only the ACCORDS platform core. The other compatibleOne services may be launched only if needed.

Publication of OCCI category management described service is performed by all components of the ACCORDS Platform during the second part of their start up procedure, after authentication, authorization and accountability requirements have been satisfied and before activation of their OCCI/REST/HTTP Server Interface.

In [34], there is a reference to a demonstartion video which shows the procedure and reasonable response time of the Accords platform to provision requested Cloud resources.

6.2 IaaS resources provision scenario

The IaaS resources provisioning scenario is detailed according to the CompatibleOne functional cycle steps detailed in Section 4.1. Such provisioning consists on a the delivery of a Virtual Machine (VM) by the selected IaaS provider through a resources manager solution. In this VM, required operating system and infrastructure services are installed and configured.

6.2.1 Handling the user's requirements

Fig. 6 details the XWiki IaaS CORDS manifest. Two IaaS nodes to allocate from providers are described in this manifest (Fig. 6, lines 3-6, lines 7-10). The first node concerns a database resource that will be instantiated in a dual core machine template (Fig. 6, line 4)

```
1 <?xml version="1.0" encoding="UTF8"?>
2 <manifest name="POCXWIKI" xmlns="http://www.compatibleone.fr/schemes/cords.xsd">
3   <node name="sqldatabase" provider="any">
4     <infrastructure name="dualcore"/>
5     <image name="mysql"/>
6   </node>
7   <node name="xwiki1" provider="any">
8     <infrastructure name="quadcore"/>
9     <image name="xwiki"/>
10  </node>
11  <configuration name="pocxwiki">
12   <action name="connectxwiki1" type="cordscript" expression="configure(database.hostname);"/>
13  </configuration>
14  <account name="compatibleOnePocXwiki">
15    <user name="user1"/>
16  </account>
17  <security name="public" level="public"/>
18 </manifest>
```

Fig. 6 XWiki IaaS CORDS manifest.

from a mysql image (Fig. 6, line 5) when the second node concerns an application image node to instantiate in a quadcore machine template (Fig. 6, line 8) from the already published XWiki image (Fig. 6, line 9). All category defined element instances, used in manifest descriptions of resources and their configuration, are identified by a unique and universal identifier. Each instance may also have a unique name by which they may be identified. These names may be composed using a local or global naming scheme.

In (Fig. 6, lines 11-13), a configuration actions to apply on these nodes are described (i.e. execution of a CORDS script (Fig. 6, line 12)). Configuration actions can be defined in order to describe the precise operations that are to be performed in order that the collection of nodes may be correctly configured for the desired purpose. Configuration actions are synonymous with object constructors. In (Fig. 6, lines 14-16), the user platform account authorized to allocate these resources and the security level to apply to the provisioned services (Fig. 6, line 17) are described.

6.2.2 Validation and provisioning plan

The validation step performs a parsing of the IaaS CORDS manifest and annotates it by URLs and bindings of published services that are able to satisfy expressed XWiki requirements. The result of the parsing is the creation of different "category instance descriptions“ on the services that satisfies the required resources which were in turn transformed to categories. Each one of these categories is identified and can be accessed via an URL returned by the publisher (See Fig. 7). By starting from the manifest description, and navigating through these URIs, one can eventually reach all the descriptions that have been generated by parsing the manifest.

The obtained plan is detailed in Fig. 8. The plan keeps the same structure of the initial CORDS manifest with minor differences. These differences consists on the addition of a new *id* attribute (Fig. 8, lines 2-7-11-12-14, 19, 21, 24, 26, 28, 31) for each manifest element (i.e. category). *id* attributes targets the location where the corresponding category instances has been provisioned (i.e. the configuration action category to apply is stored and published in the URI mentionned in Fig. 8, line 24. Elements that have a well defined set of children (e.g. node that can have an *infrastructure* and *image* nodes) are also enriched with attributes containing the location of the category instances for each one of the children. This information can also be seen in the *id* of the child elements (Fig. 8, lines 7-13). This duplication is due to the mapping between the XML element and attribute representation and the OCCI category attribute and *Link* representation. Singleton sub elements are processed by the Accords Platform Parser and create independent instances of the corresponding category. The unique and universal identifier of this newly created category instances is affected to the attribute of the parent element of the same name. This is specific to the operation performed by the Accords Platform and required in order to be able to construct the resulting fully connected OCCI graph.

During the contract negotiation phase of service brokering, node descriptions are presented to the placement engine for the selection of suitable providers on both commercial and technical terms able to offer the required types of resources. Matching is performed by the placement algorithm as defined in the service level

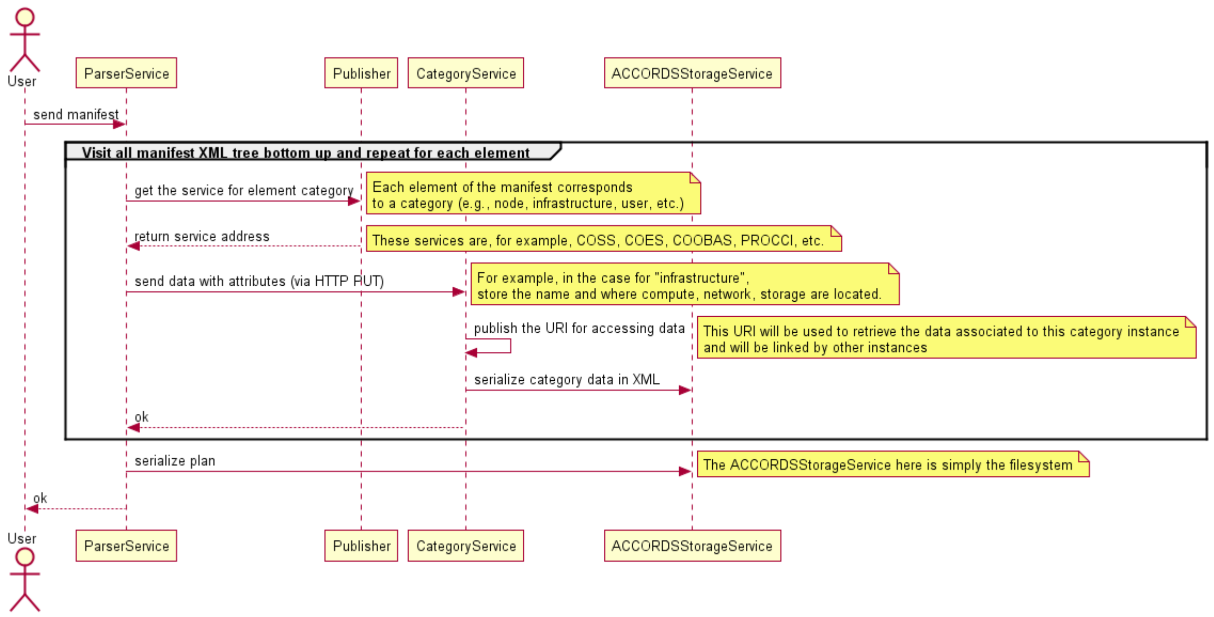
User
ParserService
Publisher
CategoryService
ACCORDSStorageService
send manifest
Visit all manifest XML tree bottom up and repeat for each element
get the service for element category
Each element of the manifest corresponds to a category (e.g., node, infrastructure, user, etc.)
return service address
These services are, for example, COSS, COES, COOBAS, PROCCI, etc.
send data with attributes (via HTTP PUT)
For example, in the case for "infrastructure", store the name and where compute, network, storage are located.
publish the URI for accessing data
This URI will be used to retrieve the data associated to this category instance and will be linked by other instances
serialize category data in XML
ok
serialize plan
The ACCORDSStorageService here is simply the filesystem
ok

Fig. 7 CORDS manifest parsing and provisioning plans generation sequence diagram.

agreement. Placement Algorithms may be defined, implemented and offered as-a-service, as required to satisfy a commercial usage of the platform.

When no suitable provider is available then the particular resource placement operation will fail and the initiator of the request will be notified of the failure through the corresponding commercial application software interface. Technical requirements, described for the infrastructure elements of the manifests nodes may be overloaded to provide alternative values through placement conditions defined through the service level agreement.

6.2.3 Execution of the provisioning plan

Each element of the obtained provisioning plan represents an allocation contract with a Cloud provider. They contracts may be provisioned via ACCORDS Proccis. CompatibleOne proccis ensures the connection between ACCORDS platform and IaaS providers by implementing communication and management actions exposed by their resources manager APIs (i.e. OpenStack[4], OpenNebula[5], etc.). For OpenStack (respectively OpenNebula) provisioning, the broker communicates with the publisher and also with the OpenStack (respectively OpenNebula) Procci.

The Accords Platform does not require Providers to support OCCI. The ACCORDS Platform provides an OCCI Standard Interface implementation (See Fig. 9) for use with Open Stack but this is an alternative to the

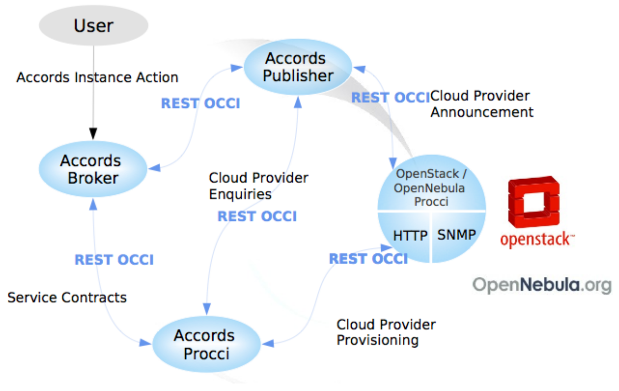

Fig. 9 CompatibleOne ACCORDS for OpenStack and/or OpenNebula Provisioning.

more common NOVA interface. The Generic PROCCI component provides an abstraction by which the Provider Specific Interface is encapsulated in the corresponding Provider Specific PROCCI for a particular provider technology type. Provider Specific PROCCI components are currently available for the OpenStack NOVA API, the OpenNebula OCCI 0.9 API, the Amazon EC2 API, the DeltaCloud API, the ComputeNext API and the WindowsAzure API.

6.2.4 Delivering the Cloud services

The result of this provisioning is the delivery of a VM instantiated using OpenNebula or OpenStack. In this VM, required OS and other infrastructure services are installed in accordance to resources described in IaaS CORDS manifest.

4 http://www.openstack.org/
5 http://opennebula.org/

```
<manifest name="POCXWIKI" xmlns="http://www.compatibleone.fr/schemes/cords.xsd"
  id="http://127.0.0.1:8092/manifest/8d5214b1-edff-487c-9dbc-5da077a1c72e"
  configuration="http://127.0.0.1:8093 configuration/34f60836-fa9c-4cf6-abd5"
  account="http://127.0.0.1:8091/account/1720ed48-0213-463b-8c95"
  security="http://127.0.0.1:8087/security/76d5ecb7 b5c5-403e-8a3d" nodes="2"
  plan="http://127.0.0.1:8093/plan/bf82b8ab-ec27-4395-93d7">
<node name="sqldatabase" provider="any" id="http://127.0.0.1:8094/node/97eaf6e5-fd10-47cb-a5ac"
      infrastructure="http://127.0.0.1:8094/infrastructure/d24b5e99-5ac0-4150-ad46"
image="http://127.0.0.1:8094/image/181a2825-a2d8-4ec8-9e76">
<infrastructure name="dualcore"
 id="http://127.0.0.1:8094/infrastructure/d24b5e99-5ac0-4150-ad46"/>
<image name="mysql" id="http://127.0.0.1:8094/image/181a2825-a2d8-4ec8-9e76"/>
</node>
<node name="xwiki1" provider="any" id="http://127.0.0.1:8094/node/d1e90660-d2de-4f12-be33"
      infrastructure="http://127.0.0.1:8094/infrastructure/71609251-6193-4177"
      image="http://127.0.0.1:8094/image/9597d559-d7cc-4be7-bfcb">
<infrastructure name="quadcore"
 id="http://127.0.0.1:8094/infrastructure/71609251-6193-4177"/>
<image name="xwiki" id="http://127.0.0.1:8094/image/9597d559-d7cc-4be7-bfcb"/>
</node>
<configuration name="pocxwiki" id="http://127.0.0.1:8093/configuration/34f60836-fa9c-4cf6-abd5"
actions="1">
<action name="connectxwiki1" type="cordscript" expression="configure(database.hostname);"
id="http://127.0.0.1:8093/action/30063fa1-66c6-49d0-9943"/>
</configuration>
<account name="compatibleOnePocXwiki" id="http://127.0.0.1:8091/account/1720ed48-0213-463b-8c95"
  users="1">
<user name="user1" id="http://127.0.0.1:8087/user/04ef149a-d191-4726-a9bd"/>
</account>
<security name="public" level="public"
  id="http://127.0.0.1:8087/security/76d5ecb7-b5c5-403e-8a3d"/>
</manifest>
```

Fig. 8 XWiki IaaS provisioning plan.

6.3 PaaS provision scenario

The PaaS resources provisioning scenario is detailed according to the CompatibleOne functional cycle steps detailed in Section 4.1. Such provisioning consists on the : (i) delivery, install and configuration of the hosting XWiki environment in the VM and (ii) the support of the XWiki sources archives upload and deployment.

6.3.1 Handling the user's requirements

Fig. 10 details the XWiki CORDS PaaS manifest. This manifest describes the XWiki application (Fig. 10, lines 4-11) and its hosting environment (Fig. 10, lines 12-24). The name of the XWiki application (Fig. 10, line 4) and the label of the version to deploy (Fig. 10, line 6) are specified. Content-type of XWiki 1.0 deployable is a Web application archive (Fig. 10, line 7). There is a set of XWiki instances to run on the targeted PaaS (Fig. 10, line 8-9): *XWikiInstance1* is defined as the default instance (Fig. 10, line 8).

All these instances have to be hosted and executed in a Java Web environment (Fig. 10, line 12) instantiated from *JavaEnvTemplate* (Fig. 10, lines 13-23). The link between the application and its environment is expressed in the environment attribute of the application element (Fig. 10, line 4). The defined template *JavaEnvTemplate* is composed of two PaaS resource nodes: An Apache Tomcat as Web container (Fig. 10, line 15) to host the XWiki application and a MySQL database instance (Fig. 10, line 16) for storing persistent data. A script to set a binding between these resources is also specified (Fig. 10, line 18). An environment variable required by the container is provided (Fig. 10, line 21).

6.3.2 Validation and provisioning plan

The validation of the PaaS CORDS manifest and the generation of the correspondent provisioning plan are performed in the same way as in the IaaS provision scenario (See Section 6.2.2). The obtained PaaS plan has the same structure as the initial PaaS CORDS manifest with injected *id* attributes for each PaaS CORDS element to reference location of their created categories.

For PaaS provisioning scenario, the broker communicates resulted contracts to the ACCORDS PaaS Procci.

```
<?xml version="1.0" encoding="UTF8"?>
<manifest name="XWikiApplication" 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"/>
      <application_version_instance name="XWikiInstance1" state="1" default="true"/>
      <application_version_instance name="XWikiInstance2" state="1" default="false"/>
    </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>
```

Fig. 10 XWiki PaaS CORDS manifest.

This specific procci is the unique CompatibleOne components which ensures communications of the CompatibleOne system and existing PaaS providers (e.g. Cloud Foundry[6], Jelastic[7], etc.). In fact, ACCORDS PaaS Proccis manages PaaS provision contracts with PaaS providers through the CompatibleOne Application Platform Service (COAPS for short). COAPS is a REST API that provides an abstraction layer for existing PaaS providers in order to allow PaaS application provisioning in a unified manner (See Fig. 11). Specifically, each COAPS operation call will be transformed to an understandable proprietary PaaS provider API call. By doing so, we always use the same set of operations on the COAPS side ignoring the specificities of proprietary PaaS providers APIs.

6.3.3 Execution of the provisioning plan

COAPS [8] exposes a set of generic operations for Cloud applications management and provisioning. These operations are organized into two categories: application management operations and environment management operations.

Currently, we provide two implementations of our API [9]: a Cloud Foundry implementation (CF-PaaS API) and an Openshift implementation (OS-PaaS API).

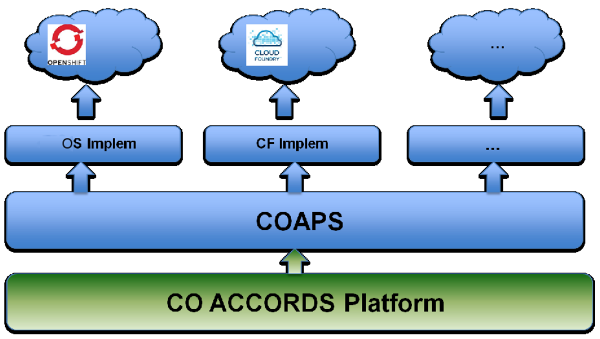

Fig. 11 COAPS : A middleware between CompatibleOne ACCORDS platform and PaaS providers.

To integrate a new PaaS provider to CompatibleOne, one can simply implements COAPS interfaces in accordance to proprietary PaaS providers operations (See Fig. 11).

To provision the XWiki application, the PaaS procci performs a sequence of COAPS API operation calls to create the hosting environment (i.e. an Apache tomcat service container) to host and run the application and a MySQL instance for persistent data (Fig. 12, *createEnvironment* operation).

6.3.4 Delivering the Cloud services

Once the hosting environment is created, we can upload XWiki source archives on it (Fig. 12, *deployApplication*

[6] http://www.Cloudfoundry.com/
[7] http://jelastic.com/

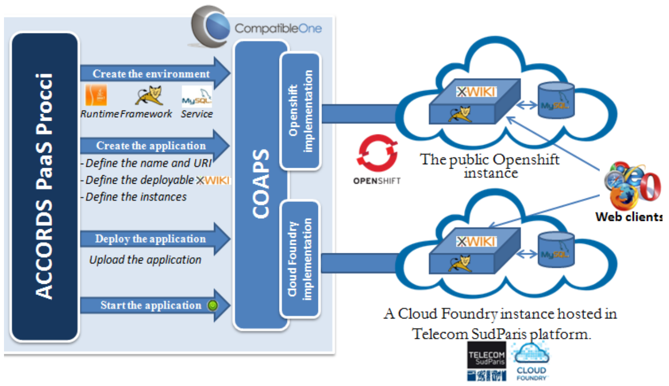

Fig. 12 XWiki application PaaS resources provisioning.

operation). This API operation supports the download of the application deployables from the referenced remote/local location and routing them to the PaaS platform. After that, the application can be started using the correspondant COAPS API call (Fig. 12, *startApplication* operation) and invoked through a browser using the public application URL returned by the PaaS provider. A demonstration video is available online at [33]. This video highlights the order of COAPS calls to deploy and execute an application and the reasonableness of the time required to COAPS when communicating with PaaS providers. There also a user guide and a tutorial to build and test this component [33].

7 Conclusion

In this paper, we presented CompatibleOne an open source broker that provides interoperable middleware for the description and federation of heterogeneous Clouds and resources provisioned by different Cloud provider. CompatibleOne is based on the object-based CORDS description model. We defined this model to describe IaaS and PaaS resources managed by our broker. In addition to that, we presented an overview of the CompatibleOne ACCORDS platform and we detailed the four steps of its functional cycle. We discussed also a realistic use case (i.e. XWiki application provisioning). We detailed needed IaaS (respectively PaaS) resources provisioned from IaaS (respectively PaaS) Cloud providers through CompatibleOne to deploy this application.

As future work, we aim to extend the CompatibleOne OCCI monitoring capabilities by adding a new feature to support the autonomic reconfiguration of provisioned resources to meet SLA at runtime based on monitoring data provided by COMONS. In other terms, we plan to add new components to ACCORDS platform that supports (1) the detection of the necessity of reconfiguring a provisioned resource to meet SLA (i.e. Analyzer component) and (2) mechanisms that support this reconfiguration (i.e. reconfiguration Manager component).

In another context, we plan also to use the ACCORDS brokering platform to build a federation Cloud system envolving several deployed ACCORDS instances. These instances can collaborate between them and federate resources provisioning of all providers that it has a deal with if needed. We aim to extend the ACCORDS publisher in order to support endpoint and credentials management of each one of the providers.

References

1. R. Harms, M. Yamartino. The economics of the Cloud. http://www.microsoft.com/en-us/news/presskits/Cloud/docs/the-economics-of-the-Cloud.pdf. (2010)
2. D. Dash, V. Kantere and A. Ailamaki. *An Economic Model for Self-Tuned Cloud Caching*. In The 25th IEEE International Conference on Data Engineering (ICDE'09), pp 1687-1693, ISBN: 978-0-7695-3545-6, Shanghai, China, 2009.
3. NIST report document. http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909505 (2012)
4. T. Metsch and A.Edmonds. Open Cloud Computing Interface - RESTful HTTP Rendering. OCCI-WG, GFD-P-R.185.(2011)
5. CompatibleOne Website. http://www.compatibleone.fr/occi/publisher/cords.htm (2013)
6. OpenStack Website. http://www.openstack.org/ (2013)
7. xWiki system Website. http://www.xwiki.com/xwiki/bin/view/Home/WebHome (2013)
8. COAPS specifications document. http://gitorious.ow2.org/ow2-compatibleone/coaps/trees/master/spec (2013)
9. COAPS source archives. http://gitorious.ow2.org/ow2-compatibleone/coaps/trees/master/api (2013)
10. Cloud Foundry Website. http://www.Cloudfoundry.com/ (2013)
11. OpenShift Website. https://openshift.redhat.com/app/ (2013)
12. Gartner report about Cloud Service Broker. http://www.gartner.com/it/page.jsp?id=1064712 (2013)
13. Daryl C. Plummer, Benoit J. Lheureux, M. Cantara, T. BovaCloud. Services Brokerage Is Dominated by Three Primary Roles, G00226509 (2011)
14. Unified Cloud Interface Project http://code.google.com/p/unifiedcloud/ (2013)
15. R. Fielding. Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine. (2000)
16. SNIA Website. http://www.snia.org/cdmi (2013)
17. CompatibleOne and OCCI. http://occi-wg.org/2012/07/15/occi-compatibleone/ (2013)
18. OpenNebula Website. http://opennebula.org/ (2013)
19. R. Buyya1, R. Ranjan, R.N. Calheiros. InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. ICA3PP'10 Proceedings of the 10th international conference on Algorithms

and Architectures for Parallel Processing - Volume Part I pp 13-31, ISBN:3-642-13118-2 978-3-642-13118-9. (2010)
20. SpotCloud Website. http://spotCloud.com/ (2013)
21. R G. Castella. Contrail: Bringing Trust in Clouds. CloudCP'11, Salzburg, Austria (2011)
22. http://contrail-project.eu/ (2013)
23. T. Metsch, A. Edmonds. Open Cloud Computing Interface - Infrastructure Extension. OCCI-WG, GFD-P-R.184. (2011)
24. R. Nyren, A. Edmonds, A. Papaspyrou,T. Metsch. Open Cloud Computing Interface - Core Technical report. OCCI-WG, GFD-P-R.183. (2011)
25. http://www.bonfire-project.eu/home (2013)
26. A. Hume, Y. Al-Hazmi, B. Belter, K. Campowsky, L. Carril, G. Carrozzo, V. Engen, D Garcia-Perez, J. Ponsat, R. Kubert, Y. Liang, C. Rohr, G Van Seghbroeck. BonFIRE: A Multi-Cloud Test Facility for Internet of Services Experimentation, TridentCom'12, Thessaloniki, Greece. (2012).
27. F. Wu, L. Zhang, B. Huberman. Truth-telling reservations. Lect Notes Comput Sci 3828:8091. doi:10.1007/11600930_9. (2005)
28. O. Rogers, D. Cliff. A financial brokerage model for Cloud computing. Journal of Cloud Computing: Advances, Systems and Applications 2012, 1:2 http://www.Cloud-casa.com/content/1/1/2 (2012)
29. ACCORDS Platform Repository. http://gitorious.ow2.org/ow2-compatibleone/accords-platform. (2013)
30. S.M. Han, M. M. Hassan, C.W. Yoon, E.N. Huh. Efficient service recommendation system for cloud computing market. Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, ser. ICIS'09. New York, NY, USA: ACM, pp. 839845. (2009)
31. P. Pawluk, B. Simmons, M. Smit, M. Litoiu, S. Mankovski. Introducing STRATOS: A Cloud Broker Service. IEEE 5th International Conference on Cloud Computing (CLOUD), pp 891-898 Honolulu, Hawaii, USA. (2012)
32. http://www.mosaic-cloud.eu/ (2013)
33. COAPS Web Page. http://www-inf.it-sudparis.eu/SIMBAD/tools/starPaaS/ (2013)
34. Accords platform demonstration video. http://www.youtube.com/watch?v=CKVtFdlJkcc (2013)
35. D. Petcu, C. Craciun, M. Rak. Towards a Cross Platform Cloud API - Components for Cloud Federation. The 1st International Conference on Cloud Computing and Services Science, pp 166-169, Noordwijkerhout, Netherlands. (2011)
36. R.L. Grossman. The Case for Cloud Computing. IT Professional, vol.11 no.2, pp 23-27 (2009)
37. F. Moscato, R. Aversa, B. Di Martino, T.Fortis, V.Munteanu. An analysis of mOSAIC ontology for Cloud resources annotation. Federated Conference on Computer Science and Information Systems (FedCSIS), pp 973-980 Szczecin, Poland. (2011)