View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Universidade do Minho: RepositoriUM

Computer Science and Information Systems 11(4):1209–1228 DOI: 10.2298/CSIS130828028C

PaaS Manager: A Platform-as-a-Service Aggregation Framework

David Cunha[1], Pedro Neves[1], and Pedro Sousa[2]

[1] Portugal Telecom Inovação, SA
3810-106 Aveiro, Portugal
davidgoncalvescunha@gmail.com, pedro-m-neves@ptinovacao.pt
[2] Centro ALGORITMI / Department of Informatics
University of Minho, 4710-057 Braga, Portugal
pns@di.uminho.pt

Abstract. The advent of Cloud Computing opened new opportunities in several areas, namely in the application development processes. As consequence, nowadays, PaaS (Platform-as-a-Service) service model allows simpler and flexible deployment strategies of applications, avoiding the need for dedicated networks, servers, storage, and other services. Within this context, several PaaS providers exist in the market, but each one having specific characteristics, proprietary technologies and Application Programming Interfaces (APIs). Based on such assumptions, this work addresses the challenge of devising a PaaS aggregation solution with the objective of unifying the information and management processes of applications created in PaaS environments. The proposed solution, denominated as PaaS Manager, take the form of a PaaS API aggregator aiming to struggle the existing lock-in in the PaaS market. In this perspective, this paper describes the specification, development and test of the proposed PaaS Manager solution. As result of this framework, end-users are able to select the most appropriate PaaS platform for an application, interacting with any supported vendor through a unique deployment and management interface.

Keywords: Platform-as-a-Service (PaaS), Interoperability, PaaS APIs.

1. Introduction

The advent of Cloud Computing technologies has greatly fostered the efforts to make real the vision of computing as a utility [19], thus being a very attractive market for the IT industry. In this perspective, cloud aware technological solutions allow that computational resources and applications be delivered through a pay-per-use model [1], making this approach more flexible and cost-effective from the customers perspective. In practice, this model allows end-users to have access to the services when needed, independently of their locations, also being only charged by the providers in accordance with the effective levels of use [2].

Generally, there are three commonly recognized service models for cloud computing: IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service). In the last years, several PaaS products appeared in the market, trying to offer attractive software development platforms to customers. This includes several polyglot PaaS alternatives, covering a wide range of technologies, but also specialized PaaS focused on very specific languages and developing frameworks. In this perspective, existent

PaaS platforms give ground to several technologies such as Java, Ruby, PHP, .NET, Ruby on Rails, JavaEE, MySQL, NoSQL, among many other examples. As obvious, in this mixed technological market, distinct vendors use distinct business models and underpin their products in distinct developments tools and inherent APIs. This variety of PaaS solutions supply developers with distinct alternative solutions for their particular needs [3] but, on the other hand, once a given vendor is selected the developer may somehow get locked to such platform and its proprietary features. Such divergent offerings with varying capabilities, configurations, and vendor-specific restrictions, can result in what is usually denominated as vendor lock-in problems [11][4][7]. As a simple example, developers may face a vendor lock-in problem when trying to deploy or migrate their applications to a distinct PaaS provider since the APIs used by each provider differ.

As result, interoperability between could service providers emerged has a relevant concern within the general topic of inter-cloud [12]. In fact, the deployment of cloud interoperability solutions will allow to attain the ability to manage, monitor and migrate applications among distinct vendors, thus abstracting the intrinsic differences from each PaaS provider [16]. All these capabilities will effectively contribute to foster the adoption of cloud aware platforms, also decreasing the reticences that some organizations might still have in the adoption of cloud solutions. The research efforts made in the could interoperability field started by mainly addressing the IaaS could layer, with approaches such as the OGF's OCCI or the Apache DeltaCloud API. Meanwhile, several R&D projects have also focused on some interoperability and portability aspects of Cloud Computing at the PaaS layer. As example, the 4CaaSt project [15] pursued an advanced PaaS Cloud platform which supports the optimized and elastic hosting of Internet-scale multi-tier applications. The CumuloNimbo [10] aims to attain a scalable PaaS Service enabling secure and un-partitioned data transactions resulting in consistent applications. The CONTRAIL project [9] allows that resources belonging to different operators be integrated into a single homogeneous federated Cloud that users can access seamlessly. The mOSAIC project [13], encompassing both IaaS and PaaS layers, aims to create, promote and exploit an open-source Cloud API and platform targeted for designing and developing multi-Cloud-oriented applications.

More recently, some players such as Oracle, Rackspace, RedHat, CloudBees, Huawei, Cloudsoft Corporation and Software AG, also fostered preliminary efforts for the specification of an open-source API, denominated as CAMP (Cloud Application Management for Platforms)[3]. The objective of such API is the creation, monitoring and management of applications and databases across multiple PaaS vendors. Even considering such efforts, it is not yet clear if such standards will be effectively adopted and implemented in the future by many relevant vendors in this competing market. Moreover, vendors are always trying to differentiate their products from competitors offering new functionalities and technological solutions. In this perspective, it is commonly accepted that heterogeneous PaaS solutions be always expected to exist in the market.

In this context, this work presents a framework focusing on the aggregation of several relevant public PaaS solutions based on their similarities. The framework, named as PaaS Manager, takes the form of a novel abstraction layer exposing a common API for developers [6], thus addressing the interoperability between providers and the portability of applications in order to reduce the vendor lock-in. Such harmonized API supported

[3] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=camp

by the PaaS Manager enables the management of applications across different PaaS offerings. Moreover, and as will be discussed later, it also allows the development of additional value-added services ruling the interactions between end users and PaaS platforms.

In the presented aggregation framework, four providers were selected (CloudBees, Cloud Foundry, Iron Foundry and Heroku) to belong to an interoperable PaaS ecosystem. As far as we know, this is one of the first works in this area presenting an implemented and tested system aggregating several PaaS offerings. Moreover, some of the assumptions and objectives underpinning this work are also corroborated and complemented by other recently proposed multi-PaaS application management approaches [18].

The remainder of this paper is organized as follows. Section 2 makes a brief summary of the Platform-as-a-Service platforms selected to integrate the proposed solution. Section 3 describes the proposed framework, highlighting the PaaS Manager architecture, which modules and operations are then described in detail in Section 3.3. After that, an example of an use-case scenario is described in Section 4. Section 5 overviews an illustrative performance analysis of the implemented PaaS Manager prototype. Finally, the conclusions and future work topics are discussed in Section 6.

2. Platform-as-a-Service

In the last years, the IaaS service model was recognized as a good example of the successful use of cloud aware solutions in enterprise computing environments. However, the inherent characteristics of the PaaS service model [14] also opened additional opportunities for the organizations in the area of application development processes. In fact, the PaaS model is by nature application oriented, allowing to abstract organizations from many configurations and maintenance tasks inherently associated with the application deployment processes. As consequence, simpler and flexible applications deployment strategies could be attained by organizations, allowing to effectively concentrate the efforts on improving the application quality and on delivering more value to end customers [3].

In general, a PaaS provider intends to supply an integrated and easy to use environment allowing to develop, test, monitor and host applications. As consequence, organizations using PaaS platforms take advantage of a reduced complexity and time allocated to the application deployment lifecycle, also being able to benefit from the auto-scalable features of cloud infrastructures. In practice, with PaaS aware solutions it is possible to speedup the deployment of service prototypes without a large initial investment. Thus, innovation could be easily pursued even by simple organizations without significant economic resources.

In the next sections, we overview and briefly characterize the PaaS platforms that were selected to integrate the devised PaaS Manager aggregation framework.

2.1. Selected PaaS Ecosystem

The PaaS market integrates several alternative platforms, including offers made by relevant players such as Amazon, Google and Windows. Besides such relevant titans, other interesting PaaS platforms also emerged in the last years, namely: *i)* Java-based PaaS CloudBees, *ii)* Salesforce.com's Heroku, and the open-source solutions: *iii)* VMware's Cloud Foundry and *iv)* Tier 3's Iron Foundry. These last four PaaS platforms were chosen

to integrate our ecosystem due to their heterogeneity and their relevance acquired within the developers' community. Furthermore, the API libraries of the selected PaaS are open-source, thus being able to be used and integrated in the development of other software products. A brief overview of such platforms is now provided.

CloudBees CloudBees[4] was founded in 2010 and it is a PaaS entirely directed to the development of Java-based applications. It provides two services, one being directed to the application development and testing (DEV@Cloud), and other targeted only to the application deployment and execution (RUN@CLoud). This PaaS platform offers an extensive range of add-ons and tools which can help developers during the application lifecycle, such as, Maven, Jenkins, New Relic, AppDynamics and others.

Heroku Heroku[5] came in 2007 as a Ruby oriented PaaS. After being acquired by Salesforce.com in 2010, Heroku supports much more technologies and it is becoming one of the most used PaaS in the market. Currently, it is estimated that more than 1.5 million of applications are hosted in Heroku. One reason for this success is the extensive catalog of add-ons that allows users to add various types of services: logging, billing, testing, monitoring etc. Unlike the aforementioned vendors, Heroku provides only Git[6] for source code deployment. Hence, Git is becoming a *de facto* in PaaS market being more and more supported by platforms' providers.

Cloud Foundry Cloud Foundry[7] is a PaaS platform that has a dissimilar approach in the market by being fully open-source and multi-cloud. Launched in 2011 by VMware, Cloud Foundry offers a polyglot environment without being attached to a single infrastructure vendor. Users have the opportunity to change the PaaS source code and seat it on any infrastructure service at their disposal, whether public or private. With this multi-cloud and open source paradigm, Cloud Foundry has received extensive recognition. Behind this success, there is a strong community of users and organizations as well as PaaS solutions based on Cloud Foundry like AppFog, Stackato and the .NET extension Iron Foundry.

Iron Foundry Iron Foundry[8] is a PaaS platform totally based on Cloud Foundry which differs from the former by supporting .NET deployment environments and Microsoft SQL databases.

2.2. PaaS Ecosystem Characterization

This section briefly overviews some technical characteristics of the aforementioned PaaS solutions. Within this context, each platform is analyzed regarding the following features: supported programming languages, development frameworks, database support, user interfaces, underlying IaaS platform and associated monitoring capabilities.

[4] http://www.cloudbees.com/
[5] http://www.heroku.com/
[6] http://git-scm.com/
[7] http://cloudfoundry.com/
[8] http://ironfoundry.org/

Table 1 summarizes and compares the mentioned features on each of the PaaS platforms currently supported by the devised PaaS Manager framework.

Table 1. Description of the selected PaaS platforms

Provider	Prog. Languages	Frameworks	Databases	User Interface	IaaS	Monitoring
CloudBees	Java JRuby Scala	Grails Java Web Play! Spring	MySQL	Web GUI SDK, API *Plugin* Eclipse	AWS HP OVH.com priv. IaaS	Web GUI New Relic Logs
Cloud Foundry	Java Node.js Ruby	Grails Java Web Spring Rails Node Sinatra	MongoDB MySQL Postgres Redis	CLI, API Eclipse	*Multi-Cloud*	API Logs
Iron Foundry	Java Node.js Ruby PHP .NET	Grails Java Web Spring Rails Node Sinatra ASP .NET	MongoDB MySQL Postgres Redis SQL Server	CLI, API Eclipse	*Multi-Cloud*	API Logs
Heroku	Java Node.js Ruby Python	Grails Django Spring Rails Sinatra	Postgres MySQL Redis, etc.	Web GUI CLI, Git API Eclipse	AWS	New Relic Logs

3. PaaS Manager

The PaaS Manager framework is a layer which supports the fundamental operations implemented by the providers' native APIs from our ecosystem, thus abstracting the platforms' differences for developers. This common management approach is intended to fit the many developer's requirements in a PaaS environment, namely: **create and manage** applications and databases, **acquire information** concerning applications and databases, **monitor** applications in real-time and **migrate** applications between vendors, if feasible.

Within the context of this work, four providers were selected (CloudBees, Cloud Foundry, Iron Foundry and Heroku) to belong to an interoperable PaaS ecosystem. All such vendors have different APIs, monitoring and deployments tools except Cloud Foundry and Iron Foundry which share the same API implementation but use distinct technologies.

3.1. PaaS Manager Functionalities Overview

In order to accomplish the objectives of this work the selected PaaS platforms' APIs and inherent management processes were broadly studied. As result, nineteen key operations were specified based on the encountered similarities. In cases where the similarity factor was reduced, certain developments needed to be conducted in order to sustain a complete transparency for developers. In other cases, when some exposed method was not shared

with the remaining providers from the ecosystem, an aggregation could not be made by the PaaS Manager. Besides the similarity factor, the selection took into account the provision of fundamental application lifecycle processes. Figure 1 summarizes some of the major actions and states associated with the traditional application lifecycle.

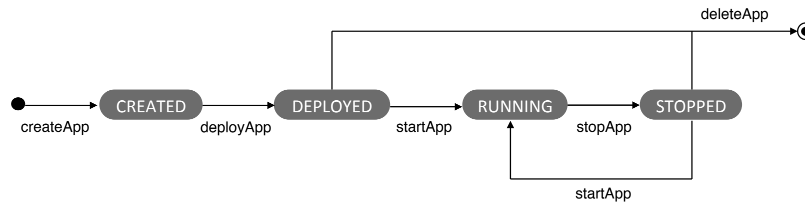

Fig. 1. Application Lifecycle Diagram

The PaaS Manager framework allows that developers interact with any supported vendor using for that purpose the defined API methods. The methods include simple tasks such as create, deploy, start, stop, delete and monitor applications. Furthermore, other advanced functionalities are also available, allowing to restart, scale manually, update or migrate applications, along with create and delete databases in any supported vendor. In certain transactions, transformations were necessary including Git support for source code deployment due to Heroku, as equally merging various methods from the same API to achieve a unified operation. The scalability and semantics were also unified now being possible to scale an application horizontally across more instances. Regarding the monitoring process each vendor supports different paradigms: CloudBees and Heroku only provide the collection of monitoring metrics through NewRelic API, which is an Application Performance Management widely used in cloud environments; Cloud Foundry and Iron Foundry simply expose a monitoring operation with basic metrics through the native API. To accomplish a seamless portability of applications, an algorithm was implemented to enable the migration to a new platform if it supports the dependencies needed to the application to run properly. From the wide range of supported operations, some of the fundamental processes will be detailed over the Section 3.3.

3.2. PaaS Manager Logical Architecture

Figure 2 presents the PaaS Manager logical architecture and the integrating modules which support the defined operational processes. A detailed description of each module will be given at Section 3.3. The PaaS Manager architecture has a modular design that allows the entire system to remains fully operational even if some vendor or monitoring API is not operating correctly. Consequently, each API has been implemented by distinct modules and managed by single entities. Finally, a REST interface exposes the specified operations to be invoked by any HTTP client application.

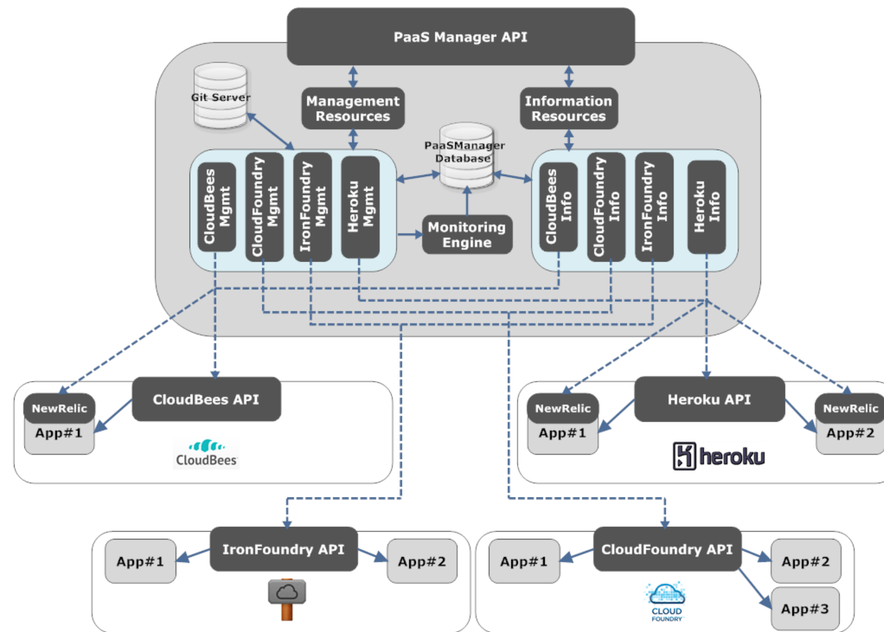

Fig. 2. PaaS Manager Architecture

3.3. PaaS Manager Modules

This section describes in detail the design and the implementation strategy of the main modules supporting the PaaS Manager shown in Figure 2. The next sections overview the following components: the PaaS Manager API, the Management Resources module, the Information Resources module and the Monitoring Engine.

PaaS Manager API The selected PaaS vendors from the defined ecosystem provide a RESTFul API for developers. In this perspective, it made perfect sense that the PaaS Manager API was also REST compliant. This API is the element that exposes the supported features and abstracts occurring transformations in background. The interface can be easily implemented by any HTTP client leveraging a lightweight and web oriented approach with JSON representations. Each specified method has a respective URI which gives the possibility to manipulate resources through GET, POST, PUT and DELETE HTTP verbs. Within the addresses design, REST good practices defined by Roy T.Fielding [8] were considered to keep the user interaction simple and intuitive as possible. As part of the authentication and authorization all the requests to the PaaS Manager API are secured via an *api-key* that identifies the logged user. On the other hand, the authentication and authorization with vendors are done through an unique account enabling the PaaS Manager to act as a mediator between users and platforms providers. Consequently the end user does not need to register in any vendor before using the supported cloud services.

As previously mentioned, with the PaaS Manager developers have the ability to manage their applications through the entire life-cycle. The operations presented in Figure 1

are part of a wide range of services which were divided into 2 groups: *i)* **Management Services** and *ii)* **Information Services**. Each group combines a set of operations isolating the features related with the management of applications and databases, with the operations related with acquisition of information. Therefore this fragmentation encourages the development of third-party modules that focus only on the desired features. Table 2 depicts the list of methods supported by the PaaS Manager and the mapping to the respective operations invoked on each PaaS API provider.

Table 2. PaaS Manager API methods (Management & Information Services)

Management Services	*Description*	*CloudBees*	*Cloud Foundry Iron Foundry*	*Heroku*
createApp	create an application in a specific PaaS	-	createApplication	createApp
deployApp	deploy the application source code	application DeployArchive	uploadApplication	Git
migrateApp	migrate an application to another PaaS	application DeployArchive	uploadApplication	Git
startApp	start an application	application Start	startApplication Application	set Maintenance Mode = 0
stopApp	stop an application	application Stop	stopApplication	set Maintenance Mode = 1
restartApp	restart an application	application Restart	restartApplication	restart
deleteApp	delete an application	application Delete	deleteApplication	destroy Application
scaleApp	manual scaling	application Scale	updateApplication Instances	scaleProcesses
updateApp	update the application source code	application DeployArchive	uploadApplication	Git
createService	bind a database to an application	databaseCreate	createService bindService	addAddon
deleteDatabase	remove a database	databaseDelete	deleteService unbindService	removeAddon
Information Services	*Description*	*CloudBees*	*Cloud Foundry Iron Foundry*	*Heroku*
getAppStatus	application health	applicationInfo getCrashes	getApplication	listProcesses
getAppStatistics	application real-time statistics	New Relic API	getApplicationStats	New Relic API
getAppInfo	application basic information	applicationInfo	getApplication getCrashes	listApps listProcesses
getAppListInfo	list of applications	applicationInfo	getApplication getCrashes	listApps listProcesses
getServiceInfo	database basic information	databaseInfo	getService	listAppsAddons
getServiceApp ListInfo	list of databases	databaseInfo	getService	listAppsAddons
getAppLogs	application logs	tailLog	-	getLogs
getPaaSOffering	PaaS supported technologies	-	-	-

Management Resources The Management Resources which integrates Figure 2 is a decision module responsible for forwarding the message to the desired method implemented by a specific PaaS adapter. In terms of management services, four adapters are defined:

CloudBees Mgmt, Cloud Foundry Mgmt, Iron Foundry Mgmt and Heroku Mgmt. These adapters have all the logic that implements the methods related with management tasks (create, deploy, start, stop, etc.) and exposed by each PaaS API. Besides interacting directly with the vendors APIs and returning unified JSON responses, these adapters are integrated with other key elements, for instance, the PaaS Manager database and a Git Server. The central database keeps state of created applications, storing the application framework identifier and the vendor name where the application is hosted.The Git server maintains repositories for each hosted application enabling users to keep multiple versions of source code. Git repositories are essential for some of the crucial operations supported by PaaS Manager in particular, create, deploy, update and migrate applications. As illustrative examples, the deployment (*Deploy App*) and migration (*Migrate App*) methods supported by the PaaS Manager are now described in detail.

A) Deploy App: In order to unify the deployment process the PaaS Manager only supports Git. Thus, developers do not need to worry about the different tools offered by each vendors before deploying their applications. Figure 3 illustrates the process performed during the deployment operation.

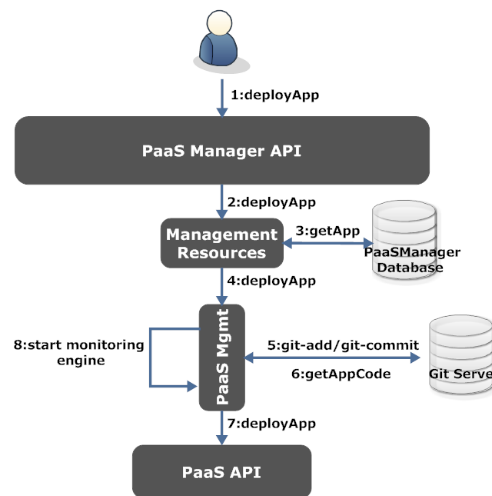

Fig. 3. PaaS Manager - Deployment Process

The request received by the API is instantly sent to the Management Resources, which performs a search in the central database in order to acquire the platform identification where the application was created with a previous operation (*createApplication*). Upon the outcome, the Management Resources invokes the specific PaaS adapter which in turn runs *git-add* and *git-commit* commands in the application repository. After this process is accomplished, the deployment is carried based on the PaaS supported paradigm. With CloudBees, the repository is examined until the adapter finds the application *web archive* (.war). For Cloud Foundry and Iron Foundry the process is similar for Java-based applications, however, for other environments only a *.zip* file containing all the source code

is sent to the platform API. Finally, with Heroku the deployment is accomplished running the *git-push* command towards the remote repository previously created with the *createApplication* operation. In case of success, a monitoring worker starts in order to extract real-time statistics to the central database. The monitoring engine adopted strategy will be analyzed later.

B) Migrate App: The portability of applications is becoming crucial in the developers' point of view. The decision of migrating may be based on seeking a PaaS environment that would give a better performance or a more advantageous business model for the planned activity. This work does not intend to define any standard model that would be shared by heterogeneous vendors, but rather aims to adapt a solution to what is already exposed by each platform API. Therefore, it requires a prior analysis if the platform to where the application will be migrated supports the required technologies for the application run properly. From the defined ecosystem, only Heroku allows to retrieve the application source code via Git commands, particularly *git-pull*. However, the remaining PaaS don't support any form of access after the deployment process is accomplished. Hence it was essential to maintain all the applications source codes and respective versions in a central Git server. This approach may question the architecture scalability, although, the deletion of applications performed by the developers will enforce the removal of unnecessary repositories keeping the system clean and efficient.

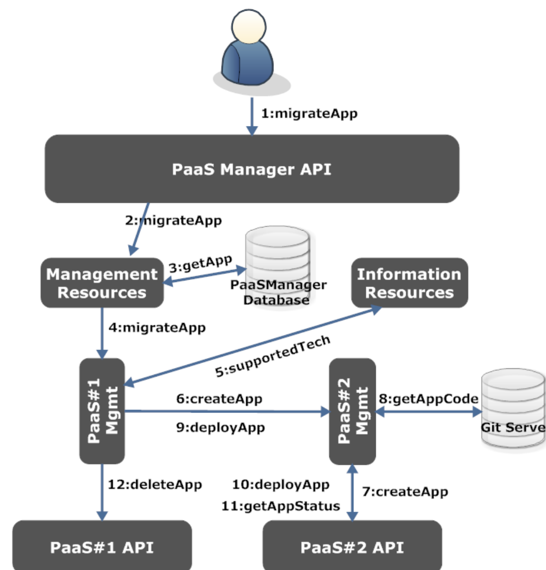

Fig. 4. PaaS Manager - Migration Process

Figure 4 summarizes the main steps that are performed during an application migration process. The request done towards the PaaS Manager API is forwarded to the Management Resources module. The central database is queried in order to return the PaaS identification where the application is hosted, as well as the application's framework (e.g,

Ruby on Rails, Spring, Zend). After the specific adapter has access to this information, it analyzes whether the new vendor to where the application will be migrated supports the required technologies. In the case of correlation, the application code kept in the Git server is deployed on the new platform by invoking the needed operations. The application status in the new PaaS is queried, which in case of success triggers the removal of the same application in the previous provider. To complete the migration process, the application's information in the central database is updated and the monitoring engine associated to the new platform is started. However, if the application is not running correctly, the adapter activates the removal process keeping the initial application running in the first provider. At the moment, the PaaS Manager does not support the migration of database data and does not maintain application state when a migration process occurs. To overcome these limitations, the PaaS Manager should synchronize the entire database data to a central location and auto-configure the application source code files in order to connect to the newly created database. Nevertheless, the PaaS Manager supports an operation that returns the databases access credentials for import and export data operations.

Information Resources The Information Resources module of Figure 2 is a decision module that implements methods related to acquisition of information concerning applications and databases. Likewise the Management Resources module, four PaaS adapters were defined: CloudBees Info, Cloud Foundry Info, Iron Foundry Info and Heroku Info. These adapters have all the logic that implements the methods related to acquisition of information (get application info, get database info, monitoring, logs, etc.) and exposed by each PaaS API. In brief, the specified operations give all the essential information about the behavior of developers' applications in specific hosting environments. As an illustrative example, the *Get App Status* process is described in Figure 5. The *Get App Status* operation gives the possibility to check the status of an application. In order to set simple and unified semantics, four possible states were specified: *running*, *stopped*, *crashed* and *unknown*. Firstly, the request made to the PaaS Manager API is instantly routed to the Information Resources module, which directs the message to the specific PaaS adapter. Finally, the adapter's logic invokes the platform API operation, mapping the result to one of the four states aforementioned.

Monitoring Engine Monitoring processes could give important feedback about how a piece of software reacts and behaves in a specific environment with hundred or thousands of users' requests. Within such purposes, several studies conducted in this area tried to define monitoring frameworks and metrics models which could be standardized and consequently shared among cloud providers [17]. The metrics list is quite extensive, including parameters such as availability, response time, RAM, CPU usage, database requests, web transactions, threads, user sessions, among others [5]. The definition of a common monitoring standard would unify SLAs even in heterogeneous environments. However, the PaaS service model is in its infancy and currently each vendor still provides different metrics and tools for monitoring software.

As discussed in section 3.1, CloudBees and Heroku have partnered with NewRelic, on the other hand, Cloud Foundry and Iron Foundry provide a native monitoring support. In this perspective, unlike the remaining methods supported by the PaaS Manager,

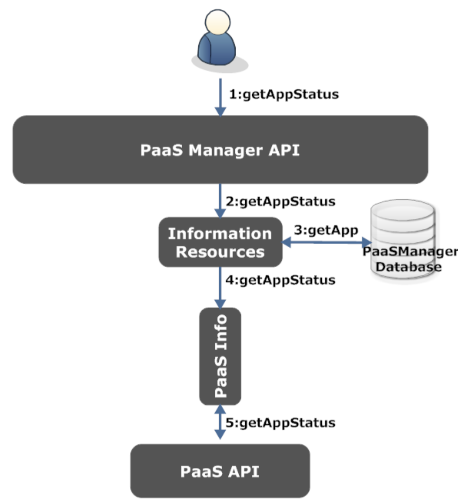

Fig. 5. PaaS Manager - Get App Status Process

the defined monitoring process is vendor-oriented and collects real-time metrics exhibited by each platform. After the application has been deployed in one of the platforms, a background job is launched and kept alive until the application is stopped or until it is removed. This process is defined by a synchronous sampling towards the provider's monitoring tool (New Relic API or native API). The samples are then stored in the central database and can be queried through the PaaS Manager API. The module responsible for carrying this process is the Monitoring Engine, which implements the above described monitoring strategy (as depicted in Figure 6).

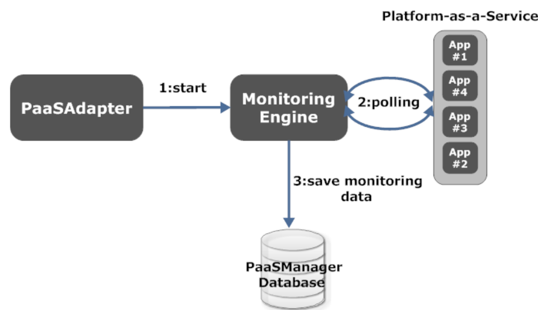

Fig. 6. PaaS Manager - Monitoring Process

4. Illustrative Use-case Scenario: Cloud Service Broker

There is a wide range of services that could take advantage of the devised PaaS Manager platform. In fact, new modules may be built to interact with the PaaS Manager API in order to use the returned information for specific purposes. This section only describes a simple illustrative use-case resorting to the implemented PaaS Manager framework. This illustrative use-case, deployed with the objective of validating the PaaS Manager functionalities, also highlights the usefulness of the PaaS Manager in both users and corporate point-of-views.

This service architecture, depicted in Figure 7, integrates a Cloud Service Broker, the PaaS Manager, and distinct client interfaces: web, command-line interface and Git for deploying the application source code. The Cloud Service Broker assists the user in the application lifecycle making the process more comfortable and attractive through a single interface. Furthermore, the Could Service Broker also acts as a recommender engine which can be based on distinct and configurable rules. As example, based on the technical profile of a given application, the recommendation system can recommend which PaaS is more suitable for the objectives of such application. Through the web or command-line interface the user has access to the management and information operations as well as the recommendation process. For that purpose, a given user can fill a form with the technical profile of the application (e.g. runtimes, frameworks and databases required). Consequently the Cloud Service Broker invokes a specific PaaS Manager method (the *getPaaSOffering* operation) returning the technologies supported by each platform from the ecosystem. The recommender engine of the Cloud Service Broker analyses the returned information comparing it with information entered in the technical profile of the application. As result, it is possible to exhibit an ordered list with the recommended PaaS platforms for such application.

The use case described in Figure 7 is currently available for use providing an easy and intuitive interface with the selected PaaS vendors. Thus, from the user point of view, interacting with this simple recommendation architecture does not add significant complexity, but provides end-users with an attractive and valuable service able to deal with multi-PaaS platforms.

In the above mentioned example, the recommendation engine has considered the technical requirements of an application as the main input to recommend a given PaaS. Nevertheless, it is important to highlight that the recommendation engine can also be extended to consider other recommendation rules. As example, business constraints can be added to the recommendation process in order to find the most cost-effective PaaS platform. Other advanced approaches can also be considered, such as include notifications when certain thresholds in the monitoring values of the PaaS platforms are exceeded (e.g. response times, application performance metrics, etc.). This will allow, for instance, to inform the users about which PaaS platforms are effectively appropriated to comply with specific service level agreements that could have been previously defined.

5. Performance Analysis

This section presents preliminary performance analysis tests conducted in order to evaluate some fundamental operations from the PaaS Manager framework. As illustrative ex-

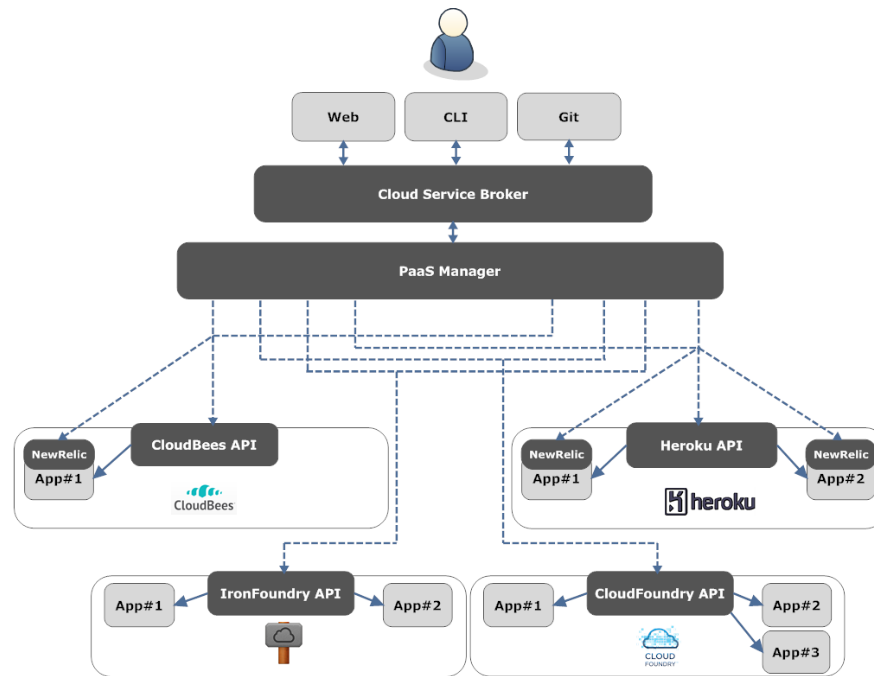

Fig. 7. Illustrative Use-case: Cloud Service Broker

amples, from the several operations supported by the PaaS Manager results from four specific illustrative methods are presented, namely: *i) createApp*, *ii) deployApp*, *iii) getAppStatus* and *iv) migrateApp*.

The Apache JMeter tool[9] was used for performing load tests simulating the simultaneous access of several users to the PaaS Manager framework. The main metric analyzed was the response time. Obviously this metric is highly variable and depends on several factors such as network conditions, the application server, computational resources, intrinsic PaaS characteristics, etc. In this perspective, it is important to emphasize that the purpose of this analysis is not to directly compare the efficiency of each provider, but only to trace a preliminary picture of the PaaS Manager architecture's performance. Thus, the acquired values can be seen as preliminary references of the system behavior. The machine used for hosting the software and database was a Intel® DualCore™2GHz with 3GB of RAM and Ubuntu 12.04 as operating system.

In the results presentation, each test was divided into two series: *PaaS Manager* and *PaaS Manager+PaaS API*, which respectively isolate the time consumed only by the PaaS Manager internal processing modules, from the time consumed by the internal processing plus the request to the specific PaaS API. The obtained values were used as reference for understanding the overhead added by PaaS Manager in each operation, and the scalability of the solution with different numbers of users in critical tasks. For some of the selected

[9] http://jmeter.apache.org/

methods tests were performed with 10 and 30[10] simultaneous users. On each figure, the plotted numbers are the average values obtained among all the users, complemented by the corresponding standard deviation interval.

5.1. Create App

The *createApp* process initializes a Git repository for the application and prepares the execution environment in the specific PaaS provider.

The results with 10 simultaneous users are presented in Figure 8 a). For the *PaaS Manager* series, Heroku and Cloudbees have response times under 60 ms, and Cloud Foundry and Iron Foundry have average response times around a value of 500 ms. This discrepancy exists because the internal logic of the operation *createApp* is different for each platform. Obviously, the series *PaaS Manager+PaaS API* depends fundamentally on the response of the native API. In this case, Heroku showed the highest value obtained on the ecosystem, about 2000 ms to complete the request. Cloud Foundry Iron Foundry have average values in the order of 1200 ms, while CloudBees, the only PaaS that does not support this operation through a unique API operation, does not have any assigned result. The same operation was also tested with 30 concurrent users, in Figure 8 b), and no significant differences were observed, with the response time values having the same order of magnitude as in the previous experience. Such results suggest that the PaaS Manager architecture does not significantly degrades its performance even in the presence of an higher level of simultaneous users.

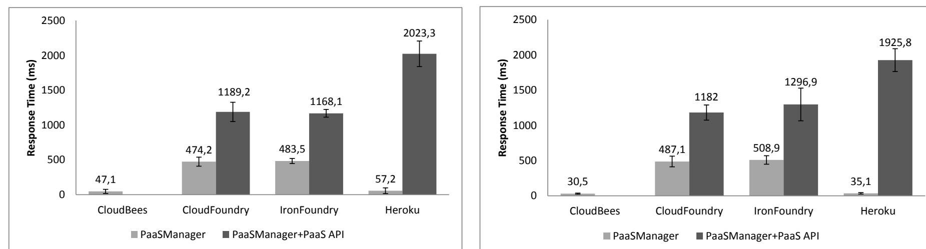

Fig. 8. Create App Process Response Time (ms) for a) 10 users, b) 30 users.

5.2. Deploy App

The *deployApp* process, detailed in Figure 3, covers various steps including the uploading of the application' source code from the Git repository to the selected PaaS provider, and the starting of the monitoring engine for collecting metrics in real-time. In the experiments, the used application was a simple Java web service (8 MB) that could be deployed in any of the supported providers.

---

[10] This specific maximum number of users was chosen from preliminary experiences which showed that possible throttling effects may occur when higher levels of requests are made to the PaaS vendors APIs.

The results with 10 simultaneous users are presented in Figure 9 a). The *PaaS Manager* series shows response time values roughly between 600 and 1200 ms. The *PaaS Manager+PaaS API* series reveals samples with response times greater than 90 seconds for CloudBees case, from which more than 90% of the response time was used by the provider's API for processing the deployment request. The remaining providers showed lower values, with Heroku having values around 63 seconds and Cloud Foundry/Iron Foundry around 6 seconds. In both cases the *PaaS Manager* series showed substantially lower values, meaning that the PaaS Manager does not induce a significant overhead in the traditional deployment process. The same operation was also tested with 30 concurrent users, in Figure 9 b), and no significant differences from the test with 10 users were observed, thus giving preliminary indications corroborating the scalability of the devised architecture.

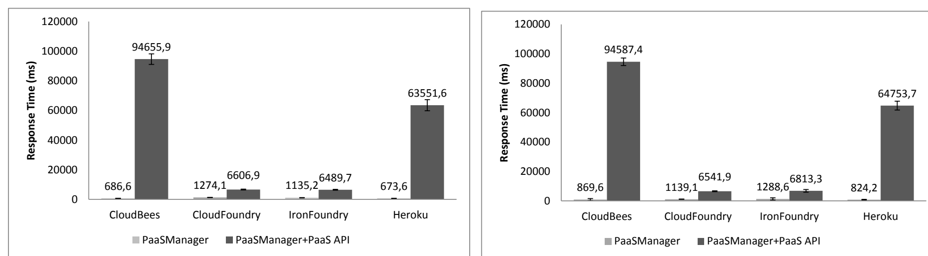

Fig. 9. Deployment Process Response Time (ms) for a) 10 users, b) 30 users.

5.3. Get App Status

The *getAppStatus* process, detailed earlier in Figure 5, acquires state information about an application, thus providing essential information for developers.

The results with 10 simultaneous users are presented in Figure 10 a). The *PaaS Manager* series shows response time values which roughly vary between 60 and 600ms, depending on the used platform. Such series values includes the acquisition of the platform identifier and the message forwarding to the respective adapter. In the *PaaS Manager+PaaS API* series, that also includes the request to the provider's native API, Heroku, Cloud Foundry and Iron Foundry reveal the highest values around 1200 and 1300ms. In the two former cases, the adapters perform a log-check for each instance where the application executes in order to detect operating errors. This procedure justifies the increasing of the response time comparatively to the getting status process performed explicitly through the provider's API.

The same operation was also tested with 30 concurrent users, in Figure 10 b), where it is visible some slightly higher response times in the CloudFoundry and IronFoundry *PaaS Manager* series. However, considering the increase made in the number of concurrent users, it is reasonable to assume that the architecture still presents an acceptable behavior as regards the scalability perspective.

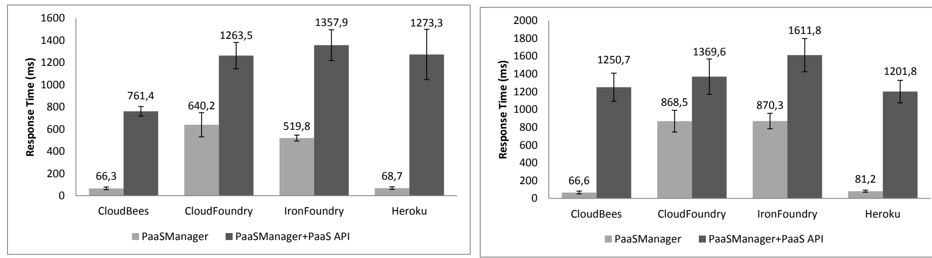

Fig. 10. Get App Status Process Response Time (ms) for a) 10 users, b) 30 users.

5.4. Migrate App

The *migrateApp* operation, previously detailed in Figure 4, includes several procedures for enabling the portability of applications between different providers. Initially, it is confirmed wether the platform to where the application will be migrated supports the required technologies for the application run properly. In case of success, the application is deployed in the new platform whereas the existing application in the previous PaaS is eliminated.

In this specific test, the focus was only given to evaluate the *migrateApp* response time part that is due to the PaaS Manager framework, i.e. the steps from 1 to 5 of Figure 4 depicting the global migration process. For that purpose, several tests were performed testing all possible cases of migration between all the PaaS providers pairs of the supported ecosystem (i.e. twelve migration cases). For each individual migration case, thirty individual tests were performed being the corresponding average values analyzed. Table 3 summarizes the obtained results, were it is possible to observe the PaaS Manager average response times approximately varying between 1200 and 1300 ms for all PaaS migration cases. Such response time values introduced by the PaaS Manager layer can be also considered has having lower impact in the global migration process. In fact, the global migration process integrates much more time consuming operations, as the case of the *deployApp* method previously overviewed.

Table 3. *migrateApp* process - PaaS Manager Response Time part (ms)

PaaS Manager [migrateApp]	*CloudBees*	*Cloud Foundry*	*Iron Foundry*	*Heroku*
CloudBees	—	1271,5	1191,4	1221,6
Cloud Foundry	1260,7	—	1195,5	1262,2
Iron Foundry	1287,7	1195,3	—	1319,0
Heroku	1213,2	1203,7	1230,4	—

6. Conclusions and Future Work

The main contribution of this work was the definition and development of an abstraction layer (PaaS Manager) that unifies the acquisition of information and the management of applications created in PaaS environments. This subject has been very discussed in the research area of interoperability and portability between cloud environments. However, despite the existence of some European projects and initiatives from large companies, this work is one of the first implementations with practical results. Furthermore, the developed architecture was integrated with a recommender system and respective web interface for recommending the best platform provider for the user's application.

The design of the solution involved an analysis of the APIs from the selected platforms as well as the definition of the key operations that the abstraction layer should bear. This activity allowed the specification of several modules: Management Resources, mainly for creating and deploying applications; Information Resources, for acquiring information about applications and databases; and the Monitoring Engine, an indispensable element for the collection of statistics in real-time. All these features are then exposed through a RESTful API that abstracts the differences between the supported PaaS providers. In order to evaluate the PaaS Manager in real scenarios several load tests were conducted. The results showed that the architecture does not introduce a significant overhead in most of the supported operations and that it behaves well with concurrent access from several users. In short, the topic discussed throughout this work has been receiving great attention by the community, waiting for new initiatives and projects that intend to give users the opportunity to control multiple platforms providers in a unified way.

As future work, there are several topics that could be addressed to enrich the PaaS Manager functionalities and to overcome some of its current limitations. As example, the security aspects of the PaaS Manager could be matter of improvements. For that purpose, secure mechanisms and protocols can be integrated to rule the interactions between the end users and the devised platform. Another interesting topic is related with fault-tolerance aspects of the proposed PaaS aggregation framework. In this perspective, we envisage scenarios resorting to load balancing mechanisms and to the replication of the PaaS Manager entity in order to improve both the performance and fault-tolerance levels of the proposed framework. Furthermore, it is also intended to wide the currently supported PaaS ecosystem to other PaaS offerings.

In a distinct perspective, there are also additional possibilities to enrich the methods supported by the PaaS Manager. In particular, the focus will be the development of operations for importing and exporting data from databases, as well as the study of techniques for migrating databases between PaaS providers. Moreover, there are other essential operations for any software development project that are not currently exposed in the APIs offered by the selected PaaS platforms (e.g. testing and debugging operations). However, such methods could be also integrated in the PaaS Manager whenever they are made available on the APIs offered by the selected PaaS vendors.

Acknowledgments. This work has been supported by FCT - Fundação para a Ciência e Tecnologia within the Project Scope: PEst-OE/EEI/UI0319/2014.

References

1. Armbrust et al., M.: A view of cloud computing. Commun. ACM 53(4), 50–58 (Apr 2010)
2. Buyya et al., R.: Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst. 25(6), 599–616 (Jun 2009)
3. Beimborn, D., Miletzki, T., Wenzel, S.: Platform as a service. Business & Information Systems Engineering 3(6), 381–384 (Oct 2011)
4. Bitzer, J.: Commercial versus open source software: the role of product heterogeneity in competition. Economic Systems 28(4), 369–381 (December 2004)
5. Cheng, X., Yuliang, S., Qingzhong, L.: A multi-tenant oriented performance monitoring, detecting and scheduling architecture based on SLA. In: Joint Conferences on Pervasive Computing. pp. 599–604. JCPC 2009, IEEE Internet Computing, Tamsui, Taipei (2009)
6. Cunha, D., Neves, P., Sousa, P.: A platform-as-a-service api aggregator. In: World Conference on Information Systems and Technologies. pp. 807–818. WorldCIST'13, Springer, Algarve, Portugal (2013)
7. Durkee, D.: Why cloud computing will never be free. Commun. ACM 53(5), 62–69 (May 2010)
8. Fielding, R.T.: REST: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
9. Harsh, P., Jégou, Y., Cascella, R.G., Morin, C.: Contrail virtual execution platform challenges in being part of a cloud federation - (invited paper). In: Abramowicz, W., Llorente, I.M., Surridge, M., Zisman, A., Vayssière, J. (eds.) ServiceWave. Lecture Notes in Computer Science, vol. 6994, pp. 50–61. Springer (2011)
10. Jiménez-Peris, R., Patiño-Martínez, M., Magoutis, K., Bilas, A., Brondino, I.: Cumulonimbo: A highly-scalable transaction processing platform as a service. ERCIM News 2012(89) (2012)
11. Kolb, S., Wirtz, G.: Towards Application Portability in Platform as a Service. In: Proceedings of the 8th IEEE International Symposium on Service-Oriented System Engineering (SOSE). IEEE, Oxford, United Kingdom (April 7–10 2014)
12. Loutas, N., Kamateri, E., Bosi, F., Tarabanis, K.: Cloud computing interoperability: The state of play. In: Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science. pp. 752–757. CLOUDCOM '11, IEEE Computer Society, Washington, DC, USA (2011)
13. Martino, B.D., Petcu, D., Cossu, R., Goncalves, P., Máhr, T., Loichate, M.: Building a mosaic of clouds. In: Guarracino, M.R., Vivien, F., Träff, J.L., Cannataro, M., Danelutto, M., Hast, A., Perla, F., Knüpfer, A., Martino, B.D., Alexander, M. (eds.) Euro-Par Workshops. Lecture Notes in Computer Science, vol. 6586, pp. 571–578. Springer (2010)
14. Mell, P., Grance, T.: The nist definition of cloud computing. Tech. Rep. 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD (September 2011)
15. Oliveros, E., Garcia, S.: 4caast value proposition, white paper (October 2011)
16. Petcu, D.: Portability and interoperability between clouds: challenges and case study. In: Proceedings of the 4th European conference on Towards a service-based internet. pp. 62–74. ServiceWave'11, Springer-Verlag, Berlin, Heidelberg (2011)
17. Shao, J., Wang, Q.: A performance guarantee approach for cloud applications based on monitoring. In: Proceedings of the 2011 IEEE 35th Annual Computer Software and Applications Conference Workshops. pp. 25–30. COMPSACW '11, IEEE Computer Society, Washington, DC, USA (2011)
18. Zeginis, D., D'Andria, F., Bocconi, S., Cruz, J.G., Martin, O.C., Gouvas, P., Ledakis, G., Tarabanis, K.A.: A user-centric multi-paas application management solution for hybrid multi-cloud scenarios. Scalable Computing: Practice and Experience 14(1) (2013)
19. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications 1(1), 7–18 (May 2010)

David Cunha received the MSc degree in Computer Networks Engineering and Services at the University of Minho in 2012. He worked at Portugal Inovacao in Cloud Computing Services area at the Platforms and Multi-Service Networks department.

Pedro Neves received his M.S. and PhD degrees in Electronics and Telecommunications Engineering from the University of Aveiro, Portugal, in 2006 and 2012 respectively. From 2003 to 2006 he joined the Telecommunications Institute (IT), Portugal, and participated in the DAIDALOS-I and DAIDALOS-II European funded projects. In 2006 he joined Portugal Telecom Inovacao, Portugal, and participated in several European funded projects (e.g. WEIRD, HURRICANE, FUTON, MEDIEVAL, Cloud4SOA, Mobile Cloud Networking and CoherentPaaS). He has been involved in six book chapters, as well as more than 30 scientific papers in major journals and international conferences. His research interests are focused on cloud and network services management, including the infrastructure and platform layers.

Pedro Sousa graduated in Systems and Informatics Engineering at the University of Minho, Portugal, in 1995. He obtained a MSc Degree (1997) and a PhD Degree (2005), both in Computer Science, at the same University. In 1996, he joined the Computer Communications Group of the Department of Informatics at University of Minho, where he is an Assistant Professor and performs his research activities within the Algoritmi R&D Center. His main research interests include Networking Technologies and Protocols, Quality of Service, Traffic Engineering, P2P, Mobility in IP networks, Cloud Services and Software, Network and Services Optimization. The researcher is also currently involved in some multidisciplinary projects in the field of Intelligent Systems for Network Optimization. He is member of the IEEE professional association and IEEE Communications Society.

Received: August 29, 2013; Accepted: April 17, 2014.