Scalable Computing: Practice and Experience
Volume 13, Number 3, pp. 201–213. http://www.scpe.org
ISSN 1895-1767

CHALLENGES FOR THE COMPREHENSIVE MANAGEMENT OF CLOUD SERVICES IN A PAAS FRAMEWORK

SERGIO GARCÍA-GÓMEZ [*] MIGUEL JIMÉNEZ-GAÑÁN [†] YEHIA TAHER [‡] CHRISTOF MOMM [§] FREDERIC JUNKER [¶] JÓZSEF BIRO [‖] ANDREAS MENYCHTAS [**] VASILIOS ANDRIKOPOULOS [††] AND STEVE STRAUCH [‡‡]

Abstract.
The 4CaaSt project aims at developing a PaaS framework that enables flexible definition, marketing, deployment and management of Cloud-based services and applications. This paper describes the major challenges tackled by 4CaaSt for the comprehensive management of applications and services in a PaaS. These challenges involve the blueprint language to describe applications in the cloud and its lifecycle management, as well as a one stop shop for Cloud services and a PaaS level resource management featuring elasticity and advanced Network as a Service capabilities. 4CaaSt also provides a portfolio of ready to use Cloud native services and Cloud enabled immigrant technologies. The evaluation process followed to assess 4CaaSt progress is also described.

Key words: Cloud, Platform as a Service, Service Composition

AMS subject classifications. 68M14, 68U35

1. Introduction. Cloud computing is transforming the way applications and services are created, provided and consumed. The virtualization of infrastructures has lowered the barriers of entry such as cost and provisioning time for many providers, especially SMEs [1] [2]. Through the virtualization of platforms, SMEs can compete in an almost equal basis with the established players [2].

The European Union-funded 4CaaSt project[1] aims to create an advanced PaaS implementation, which supports the optimized and elastic hosting of Internet-scale, multi-tier applications and enabling the creation of a true business ecosystem [7]. Applications coming from different providers can be tailored to different users, integrated, mashed up and traded together.

In [3], the 4CaaSt value proposition is highlighted as:

- A higher level of abstraction regarding applications and services deployment, hiding the operational complexity while providing a resource efficient solution.
- A broad set of built-in programming libraries, building blocks and specific functionalities, as well as common facilities beyond what is offered and fostered by State-of-the-Art PaaS Clouds, easing development of killer applications showing the value of the 4CaaSt platform.
- An attractive business ecosystem supporting facilities to promote and monetize applications and services, as well as create an active community of users, providers and developers.
- The necessary tools to monitor the execution and manage the lifecycle of applications.

This paper focuses on the description of the research challenges that are being tackled in the 4CaaSt project, highlighting the main benefits for service developers and providers and explaining how this benefits are going to be assessed. The 4CaaSt platform revolves around the innovative concept of blueprint, an abstract description of an application or service that decouples what they offer from the resources required from the various layers of the Cloud stack. The blueprint leverages a great flexibility for the creation, deployment and marketing of applications and services in the Cloud.

The paper is structured as follows. The different usage models supported by 4CaaSt are presented in Section 2. Section 3 describes the most important innovations of the project. Section 4 explains the 4CaaSt validation scenarios and a simplified example to illustrate 4CaaSt platform process, and finally Section 5 summarizes the conclusions and the most important benefits to be obtained from the platform.

[*]Telefónica Digital, Spain (sergg@tid.es).
[†]Universidad Politécnica de Madrid, Spain (mjimenez@fi.upm.es)
[‡]ERISS, University of Tilburg, The Netherlands (Y.Taher@TilburgUniversity.edu)
[§]SAP Research Center Karlsruhe, Germany (christof.momm@sap.com,)
[¶]University of St. Gallen, Switzerland (frederic.junker@unisg.ch)
[‖]Nokia Siemens Networks, Budapest, Hungary (jozsef.biro@nsn.com)
[**]National Technical University of Athens, Greece (ameny@mail.ntua.gr)
[††]University of Stuttgart, Germany (steve.strauch@iaas.uni-stuttgart.de)
[‡‡]University of Stuttgart, Germany (vasilios.andrikopoulos@iaas.uni-stuttgart.de)
[1]http://www.4caast.eu/

2. Usage Models. According to the most cited architectural concepts for Cloud computing, Platform as a Service is an important part of Cloud computing architecture. PaaS represents the middle layer connecting the IaaS and the SaaS layer, see for example [5] [4] . However, this reflects a very simplified view on Cloud architecture. While 4CaaSt concentrates on the Platform as a Service layer of the Cloud stack, the way in which the project deals with the combination of services from the different layers, benefits from an analysis of how different roles can use them. First of all, the following roles are identified:

Service Provider , who markets deployed services based on existing software, e.g. for SaaS, PaaS, IaaS, and any other XaaS layer.

Software Provider , who provides new application software or platform (middleware) ready to be deployed on the IaaS/PaaS layer.

Customer , who contracts a software to be deployed on Cloud resources, or any service in general (SaaS, PaaS, IaaS).

Beyond those external users, it must be taken into account that there is also a Cloud platform manager. The different cloud stack layers can be used and combined in different ways by the roles specified above, leading to an abundance of deployment scenarios for applications and services over Cloud Computing resources.

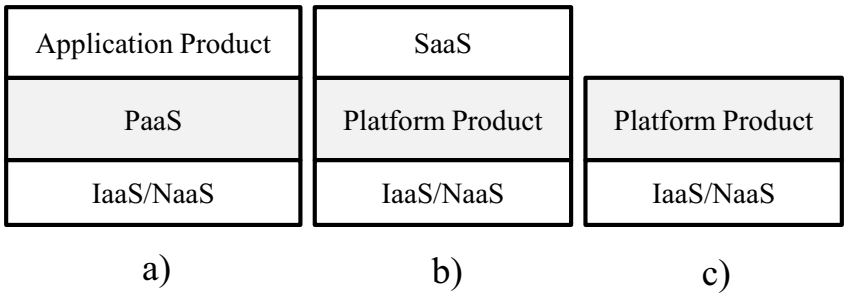

Fig. 2.1: Examples of cloud usage models.

Figure 2.1 shows three examples of scenarios of such deployments. *Application over a platform (a)*: A software provider can develop an application (e.g., a content management system) and publish it so that it can be deployed using some platform resources (a web container, a RDBMS) offered as a service (PaaS). *SaaS over a platform (b)*: A service provider can develop and/or deploy an application (e.g.: a billing application) on top of several platform products (deployed over IaaS resources) to offer a service to any external customer that contracts it (SaaS model). *Pure platform over IaaS (c)*: A development company can contract several platforms (e.g. a web container, a RDBMS, etc.) to be deployed over an IaaS and use them for their own developments. They can also contract PaaS services (e.g., a Non-SQL Data Store) as part of the development/deployment environment of their applications.

4CaaSt concentrates on the usage models that make use of the Platform Software and the PaaS, providing mechanisms to support the application software and SaaS lifecycles when deployed over the 4CaaSt platform. That includes the description of applications and their dependencies on the platform and infrastructure layers, the decisions about which specific resources to use, the deployment and configuration of platforms and application components and their management and lifecycle, the monitoring of the different layers the application consist of, and on top of that, the services required to trade with all of these usage models through appropriate business models.

Since the automatic provisioning and management of resources is one of the key features of Cloud Computing, managing the end-to-end lifecycle of any of those scenarios is a challenge. The innovative concept of the 4CaaSt blueprint, described in the following section, in combination with the marketplace functionality and the integrated management of software, services and resources, enable the reification of many usage models and business models as described above. This flexibility constitutes a key differentiator of the 4CaaSt platform in relation to major competitors both from the market and academia.

3. The 4CaaSt Innovations.

3.1. Blueprints. In 4CaaSt, every application, service or component is described by a blueprint, a description that specifies the various aspects that are linked to such resource and are required to manage its lifecycle, set up the runtime environment, and support the business transactions. Cloud Blueprinting is a powerful solution that aims at providing next-generation software developers with significant methods and tools that enable them to easily aggregate, configure and deploy virtual service-based application payloads on virtual machine and resource pools on the Cloud [8] .

The long-term benefits of Cloud Blueprinting will address concerns at the heart of the Enterprise of the Future and global service marketplaces by:

- Enabling novel geography spanning, end-to-end service applications to be built.
- Encouraging innovation through novel integrative service/Cloud development.
- Empowering service developers to better meet changing application requirements and develop customized service applications.
- Allowing new, innovative business models to be developed through the use of on-demand service platforms, infrastructure and supporting services.

To achieve its aim, Cloud Blueprinting promotes autonomous services at all levels of the Cloud stack that adhere to the same principles of separation of concerns to minimize dependencies. This solution allows any service at any layer to be appropriately combined with a service at the same level of the Cloud stack or swapped in or out without having to stop and modify other components elsewhere. At the same time Cloud Blueprinting allows multiple (and possibly composed) resource/infrastructure or implementation options for a given service at the application-level. This enables forming service aggregations on demand at any level of the Cloud stack that may potentially involve various SaaS/PaaS/IaaS providers by breaking up the current SaaS/PaaS/IaaS monolithic approach.

After having studied relevant literature [16], a proliferation of solutions for Cloud service development has been observed [10] [11] [12] [13] [14]. But, such methods have clearly shown considerable shortcomings to provide an efficient solution to deal with important aspects related to Cloud service-based applications. Some of these aspects are the elasticity and multi-tenancy of SaaS applications used to compose service-based applications. Current Cloud service offerings are often provided as a monolithic one-size-fits-all solution and give little or no opportunity for further customization. As a result, these stand-alone Cloud service offerings are more likely to show failure in meeting the business requirements of several consumers due to a lack of flexibility and interoperability.

The Cloud blueprinting approach introduces a series of Blueprint templates used to abstract and describe the components of Cloud Blueprinting-based applications. The use of templates provides a fast and simplified method for provisioning and automating Cloud services. It can be seen as a way for providing an understanding of the features used to deliver reliable and scalable Cloud deployments, and achieving better interconnection between physical and virtual infrastructures. To better manage Blueprint templates, the Blueprint framework interlaces several inter-related components [9]: (1) a declarative Blueprint Definition Language (BDL) that provides the necessary abstraction, constructs to describe the operational, performance and capacity requirements of Cloud services; (2) a Blueprint Constraint Language (BCL) that specifies any explicitly stated rule or regulation that prescribes any aspect of Cloud service; (3) a Blueprint Manipulation Language (BML) which provides a set of operators for manipulating, comparing and achieving mappings between blueprints, and (4) a simple Blueprint Query Language (BQL) for querying collections of blueprints.

The Blueprint model helps managing services when they transit through lifecycle stages: design, deployment, testing and monitoring. As illustrated by 3.1, after a provider has created the components of a service, the software provider begins the process of making it available to Cloud consumers by creating a source Blueprint model that defines the content of, and the interface to the service. Initially, during design each provider describes all relevant aspects of an offered service in a structure called a source blueprint. A service provider (might be distinct from the software provider) customizes the source blueprint templates to create a service offering for consumption by one or more consumers.

During design, an interim *target blueprint* model is created by combining a set of source blueprint models that a developer has selected. Combining source blueprints to satisfy the functional and non-functional requirements of the target blueprint relies on the blueprint resolution technique. A Cloud service developer normally starts designing a new and unresolved target blueprint that captures his to-be services. During the resolution process, in order to fulfill all resource requirements in the target blueprint, he relies on the offerings of other available third-party source blueprints that can be queried and purchased from the marketplace. The

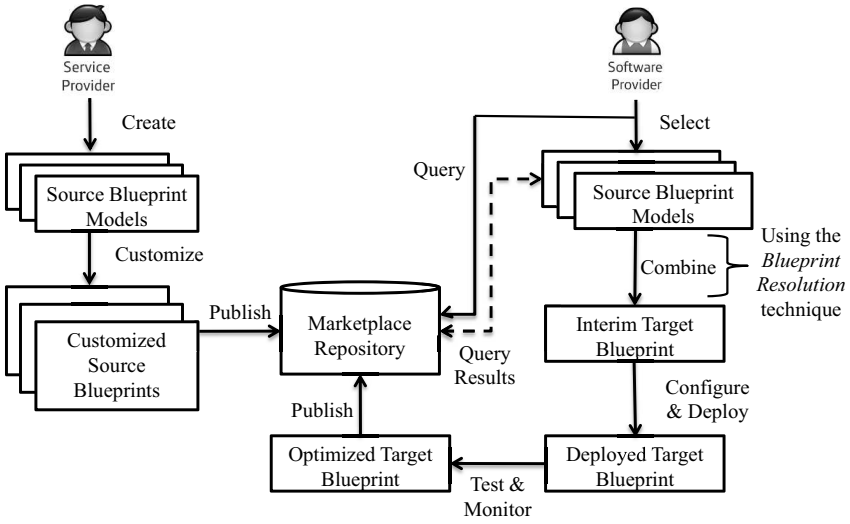

Fig. 3.1: Blueprint support for the Cloud service lifecycle.

blueprint resolution is an iterative process that ends up with a set of blueprints that fulfill all the requirements and constraints needed for actual deployment; the whole solution is referred to as *abstract resolved blueprint*. Subsequently, a deployment plan with configurability points is generated. This plan drives platform resources and virtual machine placement and network configuration.

3.2. Cloud eMarketplace. 4CaaSt provides a cloud *One Stop Shop* marketplace that supports the trading of all types of XaaS (SaaS, PaaS, etc.) services in a unified way, via a single point of access both for consumers and providers of services. This uniform specification of commercial offerings for any type of service is enabled by the marketplace's tight relationship with the 4CaaSt service engineering layer (by means of blueprints). Based on the model depicted in fig. 3.2, the 4CaaSt marketplace manages all phases of publishing and purchasing a service: information of products and stakeholders; negotiation and resolution of products; contracting and settlement of services; money flows; and analytics. Furthermore, the marketplace is tightly integrated with the 4CaaSt platform, so information sources can be leveraged by the marketplace. For instance, monitoring data from the service execution can be used for market analysis purposes.

The 4CaaSt marketplace may support different usage models, as discussed in Section 2: service providers can contract platform resources and even third-party software in order to provide and provision their own services, either by themselves or through the 4CaaSt marketplace.

The 4CaaSt infrastructure supports multiple contracting models: on one hand, customers can contract access to a public instance of a service, which is used by multiple customers and runs already before customers start using it; on the other hand, software developers can enable applications to be deployed on demand. Customers can contract a private instance of an application that will be deployed once the contract has been established.

The concept of the blueprint and its lifecycle allows 4CaaSt to provide fully-automated support for these different types of trading software, services and resources. This variety of deployment and contracting models makes software and service providers more flexible in implementing and applying diverse business models in 4CaaSt.

The 4CaaSt platform providers themselves can thereby develop dynamic business ecosystems. As a result, an intensified emergence of innovative applications for end- and corporate users can be expected on the 4CaaSt marketplace.

Service providers can define the price models for their services, which are then stored in machine-readable documents for automated processing (based on the Universal Service Description Language, USDL). When customers start using a service, their quantitative consumption is monitored by the 4CaaSt infrastructure, so

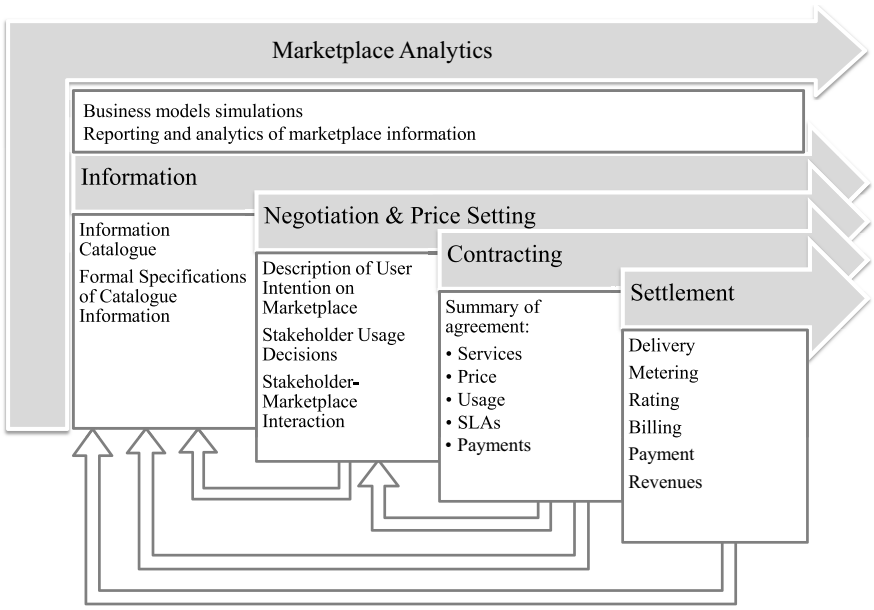

Fig. 3.2: Marketplace Processes.

the marketplace can automatically compute billing amounts and issue bills. Despite their machine-readable design, price models are very versatile and allow service providers to define the most appropriate price models, including subscription, pay per use, revenue sharing and advertisement-based models. The price model of a service can be defined either in advance or dynamically after the *blueprint resolution* by considering the aggregated cost the provider incurs by using third-party services.

The 4CaaSt marketplace also includes a feature called *business resolution*, which allows customers to select the most appropriate service for their needs. In addition to the technical requirements resolved by the blueprint resolution, the customer can specify business requirements such as business model, cost or workload. The solutions fulfilling the customer's technical requirements are customized and evaluated by the individual customer's business requirements. In this process, advanced methodologies are applied that combine the customer request and the available service offerings with various information sources such as data on market analysis customer behaviour. The quality of each solution is examined with the marketplace's analytics services and the outcomes are exploited to fine-grain future resolutions and selections.

In addition, the agents operating on the 4CaaSt marketplace, i.e. buyer and seller agents, can communicate via social tools. The data originating from this interaction is called *social data*, which describes the relationship between the agents on the marketplace. Social data is recorded and analysed to enable *socially enhanced market analysis (SEMA)*. The goal of SEMA is to employ social data to more accurately *(a)* forecast demand for given products and functionalities, and *(b)* predict the behavior of other market participants in the eMP. Market analysis tools without social enhancement are typically inaccurate, e.g. due to a lack of high-precision data input for analysis. Therefore, social enhancement can support or even replace the statistical estimation approaches currently employed by market analysis tools used by electronic marketplaces. To enable SEMA in the first place, we identify the characteristics of social data emerging in business social environments and define procedures for its acquisition, quality assurance, preparation and analysis. Analysis methods include:

- measuring the importance of individual nodes in the social graph of agents operating on the 4CaaSt marketplace.
- approaches from artificial intelligence to perform *plan recognition*, i.e. predicting the actions of agents based on partial knowledge of their environment.

Beyond SEMA, further *social enhancements* can be explored and implemented in the future, which are used to *(a)* improve features already existing in other cloud marketplaces, and *(b)* enable entirely new features, which could not exist before. In particular, these social enhancements include:

1. Socially enhanced search
2. Socially enhanced ratings
3. Socially enhanced user adaptation

These social enhancements are employed within the B2B and B2C areas. The C2C area is not covered, particularly with respect to *ratings* and *consumer adaptation*. Social data and social enhancements in C2C are out of the scope of this seminar paper as per the research objectives of the 4CaaSt project.

After analysing the most prominent Cloud services marketplaces (Windows Azure Marketplace, SuiteApp, Zoho Marketplace, Google Apps Marketplace, Google Play (formerly known as Android Market), and Force.com AppExchange), it has been realised than no other platform supports the full suite of functionalities envisioned by the 4CaaSt integrated marketplace outlined above [15]. The most comprehensive offering is provided by Force.com, although it only allows the trading of applications based on their exposed development APIs. Furthermore, it has been detected a trend for IaaS providers to include PaaS capabilities in their offerings, or SaaS providers to enable the application development over their platform [17] . However, no other platform supports in the same way a combined offering of services in the different levels of the Cloud stack. This can therefore be considered as a unique feature of 4CaaSt as compared to other Cloud marketplaces.

3.3. PaaS Deployments and Elasticity. Current PaaS offerings, like the Google App Engine, Amazon Beanstalk, Force.com and Windows Azure rely on a dedicated, homogenous set of infrastructure resources and middleware components. They are able to automatically provision resources but are limited to their choice of technology. 4CaaSt adds support for automatically generating and executing different elastic PaaS Deployments using different middleware and infrastructure components/services. This approach works as follows:

1. **Blueprint creation + resolution.** 4CaaSt component or service providers have to create a blueprint that includes all information to perform the deployment design generation and execution. In addition to the basic blueprint content this includes: (a) Definition of elasticity constraints for each artifact, in particular whether horizontal and/or vertical scalability are supported in principle. (b) Optional definition of elasticity rules for the blueprint based on monitoring KPIs + adaptation actions defined for blueprint itself or required blueprints. To define these rules we use the open Rules Interchange Format (RIF) [31]. (c) "Hints" defining quality/capacity requirements of required blueprints for achieving certain qualities, e.g. 1000 4CaaStAppServerPowerPoints required for serving 100 users in parallel with good responsiveness. (d) References to Chef-based Installation + configuration scripts for the included artifacts to enable an automated installation of the software stack. Using the 4CaaSt marketplace the customer selects a blueprint and defines his non-functional requirements (based on the hints). The blueprint resolution described before then resolves all functional dependencies by creating the abstract resolved blueprint (ARB).
2. **Deployment Design Generation.** An ARB can be considered as structured bill of material defining the required components and services for a solution. However, an ARB still leaves many degrees of freedom how these sets of components are concretely deployed, e.g. put all components on one single virtual machine (VM) or create a distributed, elastic deployment. The deployment design generation step therefore accounts for creating the best deployment design for a given ARB based on the specified customer requirements, i.e. the design that fulfills the customer requirements with a minimum set of resources. If a solution should support a certain workload range (e.g. 100-1000 users) these designs may be elastic, i.e. supporting a vertical or horizontal scaling of certain nodes. However, a prerequisite for this is that elasticity rules are available and the given elasticity constraints permit such a scaling. Since it is possible to define a set of (consistent) elasticity rules for each blueprint, it may happen that for the ARB more than one set of rules is available. In this case we use the rules defined for the "highest" blueprint in the ARB tree, e.g. the application component.
3. **Resource Provisioning.** The final deployment designs are captured using extended OVF [30] specifications (OVF++), which contains all information for "executing" the deployment, in particular the VM specifications, the list of "products" that have to be installed on the VM including deployment scripts and the selected elasticity rules. Using this information, first all required VMs are created, then for each VM the specified software stack is installed and configured (including monitoring!) and finally the machines are started.
4. **Service Operation + Adaptation.** Having completed the provisioning process, all VMs are running within the specified infrastructure cloud, e.g. Flexiscale. In addition to this, the services are registered

in the 4CaaSt runtime infrastructure, which supports monitoring, accounting as well as elastic scalability for enforcing certain quality constrained using the predefined elasticity rules.

Fig. 3.3 provides an overview to the architecture implementing this approach.

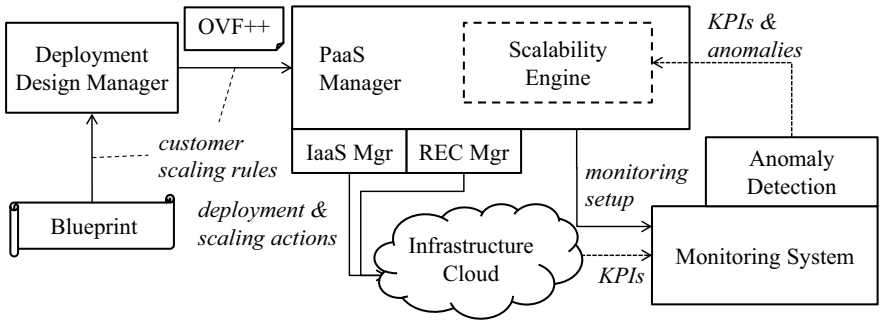

Fig. 3.3: Architecture of the PaaS deployment and elasticity mechanism.

The whole deployment design generation step is performed by the Deployment Design Manager Component, while the PaaS Manager accounts for both the resource provisioning and the elastic scaling during service operation. For the resource provisioning the PaaS Manager first interacts with a IaaS Manager for creating all required VM. After that, the so called Runtime Execution Container (REC) Manager is used to configure the individual software stacks within the machines. To this end, each machine per default includes an agent that can be remote controlled by the REC Manager. The implementation of this infrastructure is based on Chef [29]. To perform the automated elasticity the PaaS Manager includes a scalability engine, which is configured with the rules and actions defined in the OVF++. The necessary KPIs are thereby delivered through the 4CaaSt Monitoring System. This requires the configuration of monitoring probes within the different VMs, which is handled by the REC Manager during the resource providing phase. In contrast to existing solutions, the PaaS Manager supports scaling in the PaaS layer, using KPIs not only from the IaaS layer (CPU usage, free memory, etc.) but also from the PaaS layer (number of transactions per second, number of tenants, etc.). In this way, it is possible to provide PaaS architectural vertical and horizontal scaling, by decoupling middleware and components from the virtual machines in which they are hosted and deploying scalable architectures (load balancers, shared/not-shared execution containers, stateless components replication, etc.).

3.3.1. Example. In the following we explain how the approach works based on a simplified sample scenario, namely a web app that requires a Tomcat servlet engine and a MySQL database. Figure 3.4 illustrates how the blueprinting and the deployment design work for this scenario.

The "hints" used for the sizing are relations defined on a discrete set of qualities or attributes. What needs to be underlined is that 4CaaSt defines a strictly layered requirements translation approach based on blueprints. This means that every component that is being described by a blueprint, allows only requirements of the same layer or directly depending ones to be defined as hints. In the example mappings are defined between each of the offered workload values (100, 1000, 3000) and the required DB and app server (AS) capacity defined in terms of 4CaaStDBPoints / 4CaaSTASPoints, e.g. 100 Users require 100 DB and 100 AS points. In addition to this the hints defines the effect of adding more instances of a blueprint, e.g. linear behavior: Two 100 point DB / AS instances would serve 200 Users. Whether it is possible to create multiple instances depends on the elasticity constraints. In the example, the database is not scalable. Thus, as soon as the database is part of a node, there can only be 1 instance of it. For this reason, the minimal deployment design comprising only one node is not sufficient for satisfying the customer requirements. Instead, the deployment design manager proposes the maximum elastic design with arbitrary many Tomcat instances and one DB instance (due to the elasticity constraint). The initial sizing thereby is determined based on the available hints, which in this case would be 2 small Tomcat instances and one big database instance. If the workload changes at runtime, the elasticity engine executes the available rules. For the example, only rules for the Tomcat blueprint are defined as shown on Figure 3.4. So there is no need for choosing between several alternative rule sets. This Tomcat

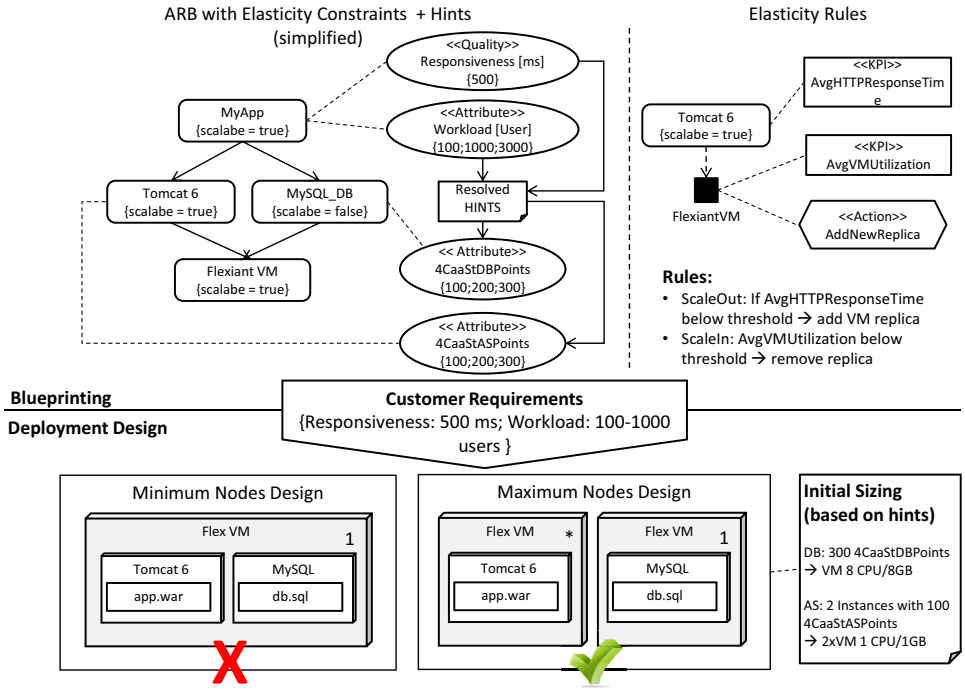

Fig. 3.4: Blueprinting and Deployment Design by Example.

rule set is included in the OVF++ and can be executed by the PaaS Manager. To install the corresponding monitoring probes, we assume a Chef-based installation/configuration script is referenced in the blueprint for every KPI.

3.4. Network as a Service. NaaS (Network as a Service) is a very popular emerging topic in the field of cloud computing. Consequently, the term is extremely overloaded, covering a wide range of networking related services and a variety of underlying technologies. The meaning of the term applied in 4CaaSt context is derived from the classic NIST definition of Infrastructure as a Service, according to which infrastructure is understood as the trio of computing, storage and networking resources. Yet, typical state-of-the-art IaaS services still focus more on computing and storage than on networking. Therefore, the major 4CaaSt objective concerning NaaS is to emancipate networking resources in a typical IaaS portfolio and give more control to clients over networking options.

Early IaaS offerings provided very simple L3 networking options: single connection to the internet with automatically preallocated IP addresses and no possibility of separated networks. These were sufficient for simple applications but not for more complex enterprise or telco application suites, formed by a set of closely or loosely cooperating VMs, possibly in multiple tiers, possibly at different data centers. Admittedly, the cloud networking scene is evolving rapidly today with various more advanced offers both at L3 and L2 (support for multiple virtual networks, secure connections towards customer premises, some control over addressing, etc.), but further work is needed to create a mature and sufficiently feature-rich networking environment.

4CaaSt tries to contribute to the ongoing networking evolution by addressing the following important aspects. The first requirement is support for network quality, i.e. the ability (of the customer) to request quality attributes of the required networking services. This requires fine-grained control over the network elements forwarding the traffic. The second requirement is the integration of the network control subsystem with the cloud control subsystem and the presentation of the networking options in the IaaS APIs. This requires a definition of a well-thought, modular architecture with interfaces providing well-thought future proof networking abstractions, allowing the continuous seamless adaptation of new networking technologies. Finally 4CaaSt also aims to integrate the NaaS offer vertically into its PaaS concept. Network resources can be requested/offered in blueprints and the resource management layer will deploy/control networking resources in an orchestrated

way alongside with other IaaS and PaaS resources.

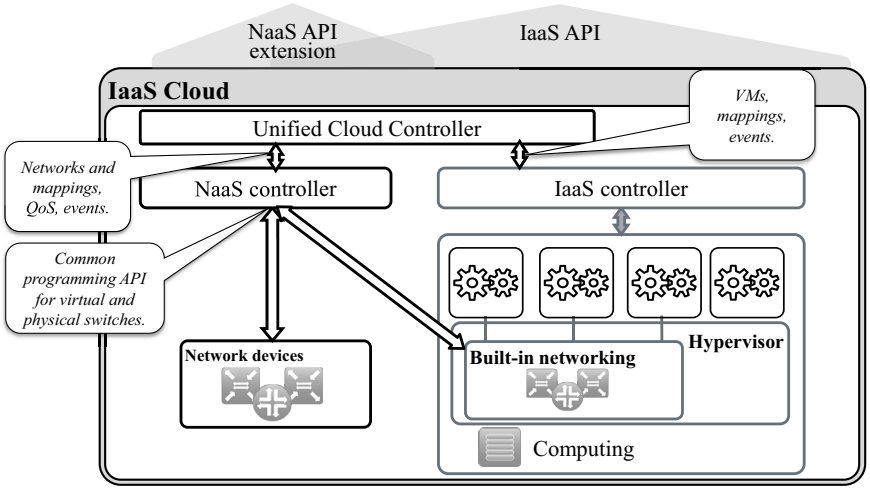

Fig. 3.5: Network as a Service Architecture.

In order to address the aforesaid requirements 4CaaSt proposes a new NaaS Manager component. This component aims to take over (fully or partially) the network management responsibility from other existing entities such as Data Center network operators or IaaS. NaaS Manager will support programmable control over the network and provide better information about the network state. Furthermore it will provide new features like traffic Engineering, QoS, multicast, anycast, etc.

NaaS Manager will provide a northbound interface towards the 4CaaSt resource management layer for the management of networking resources belonging to the deployed/managed 4CaaSt applications/entities. Its southbound interface will directly access the network elements (both physical and virtual) forming the local network under control. The architecture will enable flexible deifinition of networks boundaries and consider the requirement of close but modular integration of the NaaS Manager with different existing cloud manager platforms and possibly with peer network managers. Thus, The 4CaaSt NaaS Manager will implement the Software Defined Network paradigm for Data Center networks and integrate this concept with the 4CaaSt Paas paradigm.

3.5. Service Enablers. Service Enablers represent a common view of the integrated services inside the platform, allowing the Cloud platform to manage such services in a homogeneous way. Service Enablers have been defined so as to provide in a homogeneous way heterogeneous services, and are used to provide both Immigrant and Native PaaS technologies. Management of the services spans different actions, namely deploying the components and provisioning the services after a purchase, monitoring the correct working of the services, and accounting the usage of resources, so as to charge for them in the marketplace. Since each of these actions poses different requirements, different mechanisms and technologies have been used.

Service provisioning and deployment are built using Chef recipes, which are scripts in a domain specific language based on Ruby. Recipes are created by technology owners so as to enable the component deployment, configuration, and provisioning. Pre-defined sets of recipes, called cookbooks, are created by technology owners and invoked by REC Manager so as to provision new tenants in the services and deploy technology stacks.

Service monitoring is based on two different monitoring technologies, currently JMX and collectd, for which collectors have been created. Each technology component implements monitoring probes providing several Key Performance Indicators (KPIs) as declared in its Blueprint, and the same deployment technology installs the probe providing the needed KPIs.

Service accounting is done in an active way, providing the Marketplace with periodic reports about the resources used by a specific application. These reports, called Service Detailed Records (SDRs), provide information about the concepts declared in the price model of the service, so the Marketplace can charge the customer.

There is also a Software as a Service provisioning API used to notify an application offered as SaaS in the marketplace, the addition of removal of a client allowed to invoke that service. Those applications must provide a simple RESTfull interface so as to notify the users allowed to use the service. Basic authentication processes occurs behind the scenes so that the user purchasing a service can then log on the purchased service and start using it wit her marketplace credentials.

Native Service Technologies include technologies developed specifically for the Cloud or that happen to be very suitable for it; therefore they are likely to be involved in Cloud-oriented applications. 4CaaSt will offer several native Cloud technologies as services, such as Context aaS, Network Enablers aaS or Cloud Data Store Capabilities that can be used and deployed in an on-demand fashion according to a client's workload and requirements. These technologies are packaged as services offered by 4CaaSt in a uniform fashion, following the concept of technology enabler described above, and thus offering a common interface used by the platform in different phases of applications lifecycle, namely deployment, configuration, monitoring, management and billing.

3.6. Immigrant PaaS technologies. Immigrant PaaS technologies refer to a set of tried and proven technologies that were available before the advent of Cloud computing and now need to be adapted for the new era. These technologies form a series of building blocks for the 4CaaSt platform and can be used on demand, either in conjunction or independently, when building applications and services based on the 4CaaSt platform. 4CaaSt focuses on the provisioning of infrastructure components for composite applications and services, and in particular on providing Cloud-ready databases, application servers and service composition engines, working when required in tandem through an integration layer that handles the communication with services external to the 4CaaSt platform.

Our investigation of the State of the Art showed that existing Cloud solutions for data storage, application hosting, service composition and integration are either immature or focused exclusively on one feature like availability [18]. Furthermore, merely deploying existing non Cloud-native technologies on the Cloud as part of VM images fails to utilize the full potential of the Cloud computing paradigm in terms of on-demand elasticity and scalability on PaaS level [27]. For these reasons, in 4CaaSt we look into how these technologies can be instead immigrated to the Cloud, adding to them mechanisms and capabilities in order to make them Cloud-aware, offering out-of-the-box capabilities like multi-tenancy, scalability and PaaS-oriented management and configuration. These technologies can then be provided as installable software, or as services, similar to the Native Service Technologies.

Cloud-enabling existing technologies creates a number of research and implementation challenges to be overcome. An example of such challenges is the case of immigrating a critical middleware component like the Enterprise Service Bus (ESB) [20] to the Cloud. The concept of ESB as the messaging hub between applications addresses the fundamental need for application integration and in the last years it has become ubiquitous in enterprise computing environments. ESBs control the message handling during service invocations and are at the core of each Service Oriented Architecture (SOA) [20]. Enabling, for example, multi-tenancy at the ESB level allows multiple service consumers to use the same application offered as a service by a provider in a fully costumizable per consumer manner, both on the level of tenants, and that of users belonging to a particular tenant. Apart therefore from allowing organizations to outsource the development, deployment, operation and management of such applications to a service provider, this solution also maximizes the benefits on the provider side.

Making an ESB solution multi-tenant however requires fulfilling the following requirements [28], as identified among others by [21], [25], [24]:

- providing for *tenancy awareness*, i.e. tenant-based identification and hierarchical access control for tenants and their users,
- offering *tenant-based facilities* like management interfaces providing tenant- and user-specific deployment and configuration options, and
- ensuring *tenant isolation* and *security*, making sure that no tenant data and computational resources can be accessed by other tenants or unauthorized third parties.

In order to satisfy these requirements, in 4CaaSt we extend the open source ESB solution Apache ServiceMix [19] which is based on the OSGi Framework [26] implementing the JBI specification [22] as discussed in [28]. Beyond extending the ServiceMix components responsible for processing and transforming messages to support multi-tenant aware message exchanges, we also implemented an OSGi-based management service and the respective

Web GUI and Web Services API offering tenant-based deployment, configuration and administration of service endpoints. As the management components of the underlying resources are implemented as EJB components, we use container-managed transaction demarcation, which allows the definition of transaction attributes for whole business methods, including all resource changes [23]. A performance evaluation of our Apache ServiceMix extension is currently being performed, with encouraging results.

Given the disparity of technologies involved (databases, application servers, orchestration engines and integration technologies), the extracted requirements for enabling multi-tenancy and scalability demand a significant amount of work in order to fulfill the goals of the project.

4. Validation. 4CaaSt requirements and business view are being validated through three different scenarios. These three scenarios are implementing application prototypes which make use of some of the native cloud services and immigrant technologies provided in 4CaaSt, and are being integrated with the overall 4CaaSt platform in order to evaluate the different features provided. Below, a description of the three scenarios, together with the validation proposal, is shown. Additionally, a simplified generic scenario is described in order to illustrate the most important steps in 4CaaSt lifecycle.

4.1. A marketplace for SME's applications. The first scenario aims at developing a web application (a taxi fleet management application) that is delivered on demand on cloud resources provisioned for the customer, i.e., the resources are booked, deployed, and configured so that the software stack and the web application can be properly deployed. This means that the application is not offered as a service (it is not a multitenant SaaS), but a COTS software that is automatically installed on the cloud.

This scenario is used in the current platform release to validate a number of important features:

- Marketplace products and price models definitions.
- Contracting and business resolution.
- Blueprint resolution.
- Flexible deployment on cloud resources.
- Immigrant technologies.

4.2. A marketplace for mass market applications. This scenario is based on the creation, deployment and offering of SaaS solutions over PaaS/IaaS resources, and how to sell them as services for the mass market. In this case, a multitenant tourism application is deployed and offered as a service to final users (tourists). In the current release, it is used for the validation of the following features:

- Blueprints definition.
- Price models definition based on accounting capabilities.
- Provisioning of services.
- Accounting, charging and settlement.
- Native services.

4.3. A hybrid cloud. This scenario implies the cloud bursting of resources at PaaS level of an application that is running on a private cloud. The evaluated features are:

- Marketplace offering and contracting.
- SLAs definition and deployment design and sizing.
- Flexible cloud deployment and elasticity management.
- Network as a Service.
- Advanced IaaS features.

4.4. A simplified scenario. In order to illustrate the value proposition of 4CaaSt platform and validate the proposed approach of the project, this section describes a simple scenario where the main actors are involved, and their interactions with the system depicted:

1. The *PaaS (4CaaSt) provider* wants to have multiple technologies (software or services) in a 4CaaSt platform so that developers have a wider range of technological options to develop their applications. For instance, Context as a Service or a PostgreSQL-based service. For each of them, he must provide a blueprint with the offering and the requirements, and publish it in the marketplace according to a price model.
   External *Service Providers* could follow the same approach in order to provide services in a 4CaaSt compliant Cloud Provider, enabling them to do business in the 4CaaSt ecosystem.

2. The *Software provider* creates the application using his preferred tools (e.g., Eclipse). When he wants to integrate an existing service, he can browse through the 4CaaSt One Stop Shop in order to get information and download client libraries or other artifacts. He could also contract virtual machines or platform services (like a Tomcat deployment) to deploy and test his own application. Once the application is ready, he can create and publish the application blueprints and artifacts.
3. The application is now available and the 4CaaSt platform has all the information required to deploy and manage it. A *Service provider* can create a commercial offer of the application in the marketplace. He selects the application blueprint and defines the price model and the application is ready to be contracted. Any *Customer* will be able to obtain his own instance of the application.
4. When the *Customer* logs into 4CaaSt marketplace, he can select one out of many existing application and services and customize it if such action is allowed for the service.
5. Given the functions, price, performance, availability, or even the customer profile, the *4CaaSt Blueprint Resolution process* decides the best combination of services (blueprints) to deploy or provision for this particular *Customer*. *4CaaSt platform* decides also about which Cloud resources to use, including computing, networking, platforms and services. The application is deployed and the *4CaaSt platform* starts monitoring and accounting its usage.
6. The monitoring information is used by the 4CaaSt platform to take runtime elasticity decisions according to the constraints defined by the *Customer*.
7. Finally, the *4CaaSt marketplace* charges the *Customer* and distributes incomes among the various *Service providers* involved, including revenue sharing policies when they are defined.

5. Conclusion. The 4CaaSt project aims at creating an innovative framework for creating, marketing, deploying and managing applications on the Cloud, both over platform products and platform as a service. 4CaaSt introduces the concept of blueprint, a technical description of an application or a service that decouples the various dependencies it has along the Cloud layers.

Thanks to the blueprint innovation and how applications and services are traded and provisioned/deployed, 4CaaSt may support multiple usage and business models, giving to software and service providers the flexibility to use the resources and services as they prefer.

As a summary, 4CaaSt allows software and service providers to focus on their business (both the software and the service monetization), leaving the underlying complexity of infrastructure and platforms out of their concerns. Among the benefits that 4CaaSt provides for practitioners or IT managers, the following ones can be highlighted:

- 4CaaSt provides a large portfolio of Cloud-aware, ready to use, mainstream technologies (ESBs, RDBMS, BPM engines, etc.)
- 4CaaSt offers an easy mechanism to integrate off the self-services (context, telco, mashups, etc.).
- 4CaaSt allows developers to decouple the development and specification of applications from their actual deployment.
- 4CaaSt enables the trading of applications and services based on multiple usage models and business models, providing integrated settlement of revenue flows among providers.
- 4CaaSt is capable of on-demand deploy, configure and provision software and services into virtual machines to offer applications ready to use, including complex configuration of the networking infrastructure.
- 4CaaSt scalability is not only based on adding more infrastructure resources, but also to vary the application architecture for a more efficient usage of resources.

To date, a first release of the platform has been delivered, implementing the main features for executing an end-to-end process, and the scenarios are being integrated and validated. Moreover, the focus of the future work will be in some of the most advanced and challenging features described in this paper, especially those related to PaaS elasticity and blueprint resolution, which will represent 4CaaSt unique value proposition.

Acknowledgments. The research leading to these results has partially received funding from the 4CaaSt project (http://www.4caast.eu/) from the European Unions Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 258862. This paper expresses the opinions of the authors and not necessarily those of the European Commission. The European Commission is not liable for any use that may be made of the information contained in this paper.

REFERENCES

[1] Yefin V. Natis, Benoit J. Lheureux, Massimo Pezzini, David W. Cearly, Eric Knipp, and Daryl C. Plummer. *PaaS Road Map: A Continent Emerging*, Gartner (2011).

[2] IDC. *Market Analysis - Worldwide Public Application Development and Deployment as a Service: 2010 2014 Forecast.* (pp. 1 - 31), Framingham, MA, USA. 2010.

[3] Oliveros Díaz, E., García Gómez, S. *4CaaSt Value Proposition - A 4CaaSt Whitepaper.* (pp. 1-17) October 2011. Brussels. Available at http://www.4caast.eu.

[4] L.M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Linder. A Break in the Clouds: Towards a Cloud Definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2009.

[5] Peter Mell and Tim Grance, *The NIST Definition of Cloud Computing*, Tech. report, July 2009.

[6] Mitrabarun Sarkar, Brian Butler, and Charles Steinfield, *Cybermediaries in electronic marketspace: Toward theory building*, Journal of Business Research **41** (1998), no. 3, 215–221.

[7] James F. Moore, *Predators and prey: a new ecology of competition*, Harvard Business Review **71** (1993), no. 3, 75–86.

[8] Michael P. Papazoglou, Willem-Jan van den Heuvel. *Blueprinting the Cloud.* IEEE Internet Computing 15(6): 74-79 , 2011.

[9] Amal Elgammal, Oktay Tretken, Willem-Jan van den Heuvel, Mike P. Papazoglou. *Root-Cause Analysis of Design-Time Compliance Violations on the Basis of Property Patterns.* ICSOC 2010: 17-31

[10] Mietzner, R. *A method and implementation to define and provision variable composite applications, and its usage in Cloud computing.* Dissertation, Universitaet Stuttgart, Fakultaet Informatik, Elektrotechnik und Informationstechnik, Germany, August 2010.

[11] Galan. F, Sampaio. A, Rodero-Merino. L, Loy. I, Gil. V, Vaquero. L.M. *Service specification in Cloud environments based on extensions to open standards.* In: Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE. COMSWARE '09, New York, NY, USA, ACM, 2009 19:1-19:12

[12] Chapman, C., Emmerich, W., Marquez, F.G., Clayman, S., Galis, A. *Software architecture definition for on-demand Cloud provisioning.* In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. HPDC '10, New York, NY, USA, ACM, 2010, 61-72

[13] Wei-Tek Tsai; Xin Sun; Balasooriya, J. *Service-Oriented Cloud Computing Architecture.* Seventh International Conference on Information Technology: New Generations (ITNG), vol., no., pp.684-689, 12-14 April 2010.

[14] Hyun Jung La and Soo Dong Kim. *A Systematic Process for Developing High Quality SaaS Cloud Services.* In Proceedings of the 1st International Conference on Cloud Computing (CloudCom '09). Springer-Verlag, Berlin, Heidelberg, 278-289, 2009.

[15] García-Gómez. S. (editor). *D3.1.1, Marketplace Functions: Scientific and Technical Report.* 4CaaSt Project Deliverable D3.1.1, 2011. Available at http://www.4caast.eu.

[16] Lelli. F. (editor). *D2.1.1, Blueprinting the Cloud: Scientific and Technical Report.* 4CaaSt Project Deliverable D2.1.1, 2011. Available at http://www.4caast.eu.

[17] Giessmann, A. (editor). *D9.1.2, Market and Competition Analysis.* 4CaaSt Project Deliverable D9.1.2, 2012. Available at http://www.4caast.eu.

[18] 4CaaSt Consortium, *Immigrant PaaS Technologies: Scientific and Technical Report D7.1.1.* Deliverable, July 2011.

[19] Apache Software Foundation, *Apache ServiceMix.*

[20] David A. Chappell, *Enterprise Service Bus*, O'Reilly Media, Inc., 2004.

[21] C.J. Guo, W. Sun, Y. Huang, Z.H. Wang, and B. Gao, *A framework for native multi-tenancy application development and management*, Proceedings of the 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (CEC/EEE'07), IEEE, 2007.

[22] Java Community Process, *Java Business Integration (JBI) 1.0, Final Release*, 2005, JSR-208.

[23] *Enterprise JavaBeans (EJB) 3.0, Final Release*, JSR-220, 2006.

[24] Rouven Krebs, Christof Momm, and Samuel Konev, *Architectural Concerns in Multi-Tenant SaaS Applications*, Proceedings of the 2nd International Conference on Cloud Computing and Service Science (CLOSER'12), SciTePress, April 2012 (Englisch).

[25] Ralph Mietzner, Tobias Unger, Robert Titze, and Frank Leymann, *Combining Different Multi-Tenancy Patterns in Service-Oriented Applications*, Proceedings of the 13th IEEE Enterprise Distributed Object Conference (EDOC 2009), IEEE, September 2009 (Englisch).

[26] OSGi Alliance, *OSGi Service Platform: Core Specification Version 4.3*, 2011.

[27] L.M. Vaquero, L. Rodero-Merino, and R. Buyya, *Dynamically scaling applications in the cloud*, ACM SIGCOMM Computer Communication Review **41** (2011), no. 1, 45–52.

[28] S. Strauch, V. Andrikopoulos, F. Leymann, and D. Muhler, *ESBMT: Enabling Multi-Tenancy in Enterprise Service Buses*, in 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2012) - to appear, IEEE Computer Society, December 2012.

[29] Opscode Chef, http://www.opscode.com/chef/, 2011.

[30] Distributed Management Task Force *DMTF. Open virtualization format specification. Specification DSP0243 v1.0.0d. Technical report*, https://www.coinor.org/OS/publications/optimizationServicesFramework2008.pdf, 2008.

[31] W3C *Rule Interchange Format (RIF) Working Group* http://www.w3.org/2005/rules/wiki/RIF_Working_Group, 2010.

Edited by: José Luis Vásquez-Poletti, Dana Petcu and Francesco Lelli
Received: Sep 20, 2012
Accepted: Oct. 15, 2012