Future Internet **2012**, *4*, 322-346; doi:10.3390/fi4010322

OPEN ACCESS

future internet

ISSN 1999-5903

www.mdpi.com/journal/futureinternet

Article

Blueprinting Approach in Support of Cloud Computing

Dinh Khoa Nguyen *, Francesco Lelli, Mike P. Papazoglou and Willem-Jan van den Heuvel

European Research Institute in Service Science (ERISS), Tilburg University, Warandelaan 2, 5037 AB, Tilburg, The Netherlands; E-Mails: F.Lelli@tilburguniversity.edu (F.L.); M.P.Papazoglou@uvt.nl (M.P.P.); W.J.A.M.vdnHeuvel@uvt.nl (W.-J.H.)

***** Author to whom correspondence should be addressed; E-Mail: D.K.Nguyen@tilburguniversity.nl; Tel.: +31-0-13-466-8203.

Received: 23 November 2011; in revised form: 21 February 2012 / Accepted: 19 March 2012 / Published: 21 March 2012

Abstract: Current cloud service offerings, *i.e.*, Software-as-a-service (SaaS), Platform-as-a-service (PaaS) and Infrastructure-as-a-service (IaaS) offerings are often provided as monolithic, one-size-fits-all solutions and give little or no room for customization. This limits the ability of Service-based Application (SBA) developers to configure and syndicate offerings from multiple SaaS, PaaS, and IaaS providers to address their application requirements. Furthermore, combining different independent cloud services necessitates a uniform description format that facilitates the design, customization, and composition. *Cloud Blueprinting* is a novel approach that allows SBA developers to easily design, configure and deploy virtual SBA payloads on virtual machines and resource pools on the cloud. We propose the *Blueprint* concept as a uniform abstract description for cloud service offerings that may cross different cloud computing layers, *i.e.*, SaaS, PaaS and IaaS. To support developers with the SBA design and development in the cloud, this paper introduces a formal *Blueprint Template* for unambiguously describing a blueprint, as well as a *Blueprint Lifecycle* that guides developers through the manipulation, composition and deployment of different blueprints for an SBA. Finally, the empirical evaluation of the blueprinting approach within an EC's FP7 project is reported and an associated blueprint prototype implementation is presented.

Keywords: cloud computing; service-based application; cloud service; blueprint

---

1. Introduction

The cloud abstraction model delivers a shared pool of configurable computing resources (processors, storage, *etc*.) that can be dynamically and automatically provisioned and released. Today's cloud capabilities are defined and provided as three levels of service offerings: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). This allows users of cloud-based services to be able to focus on what the service provides them rather than how the services are implemented or hosted. Currently, there is a substantive emphasis on the low-level technological trajectories of the cloud. For instance cloud scalability is heavily researched with efforts focusing on service horizontal scaling (*i.e.*, adding new server replicas and load balancers to distribute the load) or vertical scaling (on-the-fly changing of the assigned resources to an already running instance) and associated issues such as load balancing [1–3]. However, we observe far less concentration into other aspects of the cloud, such as cloud application development.

There is a clear need for placing emphasis on how to develop enhanced composite service offerings at the application (or SaaS)-level and assign or reassign different virtual and physical resources dynamically and elastically. This leads to forming service syndications 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. This article will focus on such an approach. It will first shed light on some serious shortcomings of the cloud delivery models and management approaches, and then explain how to achieve a holistic cloud solution on the basis of the concept and techniques of cloud blueprinting.

When examining the three cloud delivery models we observe a recurrent theme. They are all constrained by the capabilities that are available by the provider at their delivery level and do not allow for easy extensibility or customization options. Better ways are necessary for cloud service consumers to orchestrate a cohesive cloud computing solution and provision cloud stack services that range across networking, computing, storage resources, and applications from diverse cloud providers. To realize this vision, the key challenges that are summarized below need to be tackled.

At the IaaS level, cross-configurations of the virtual machines comprising a specific cloud service are currently possible only within the IaaS offerings of a single vendor. For instance, the Amazon CloudFormation template [4] enables the Amazon Web Service (AWS) developers to specify a collection of AWS cloud resources and the provisioning of these resources in an orderly and predictable fashion. Nevertheless, this template works only for AWS cloud platform and infrastructure resources and thus lacks interoperability. Moreover, cross-machine configurations need to be inferred at deployment-time instead of remaining statically allocated, as is the norm with IaaS solutions today.

Similar problems also permeate the PaaS solution space. PaaS offerings are constrained by the capabilities that are available by the PaaS provider and do not allow easy extensibility, mash up, or customization options at the PaaS consumer (or developer) level. The underlying PaaS system and the applications built on are not portable across many different public and private clouds and cloud providers. Furthermore, many hosted middleware solutions are not integrated with any IaaS management, thereby lacking elasticity and scaling benefits.

SaaS is also predominantly tethered to proprietary application platforms where the cloud provider runs all elements of the service with the client presented with a complete application. Due to this

vendor lock-in problem, tremendous efforts are required to customize and combine SaaS offerings from multiple providers to offer improved functionality to the client or developer. There have been some initiatives in bringing SaaS offerings from different providers into a joint solution, e.g., the Appirio CloudFactor [5] combines the SalesForce's CRM SaaS offerings with the Google Apps SaaS offerings. However, we observe the lack of a generic approach for integrating SaaS offerings across multiple providers.

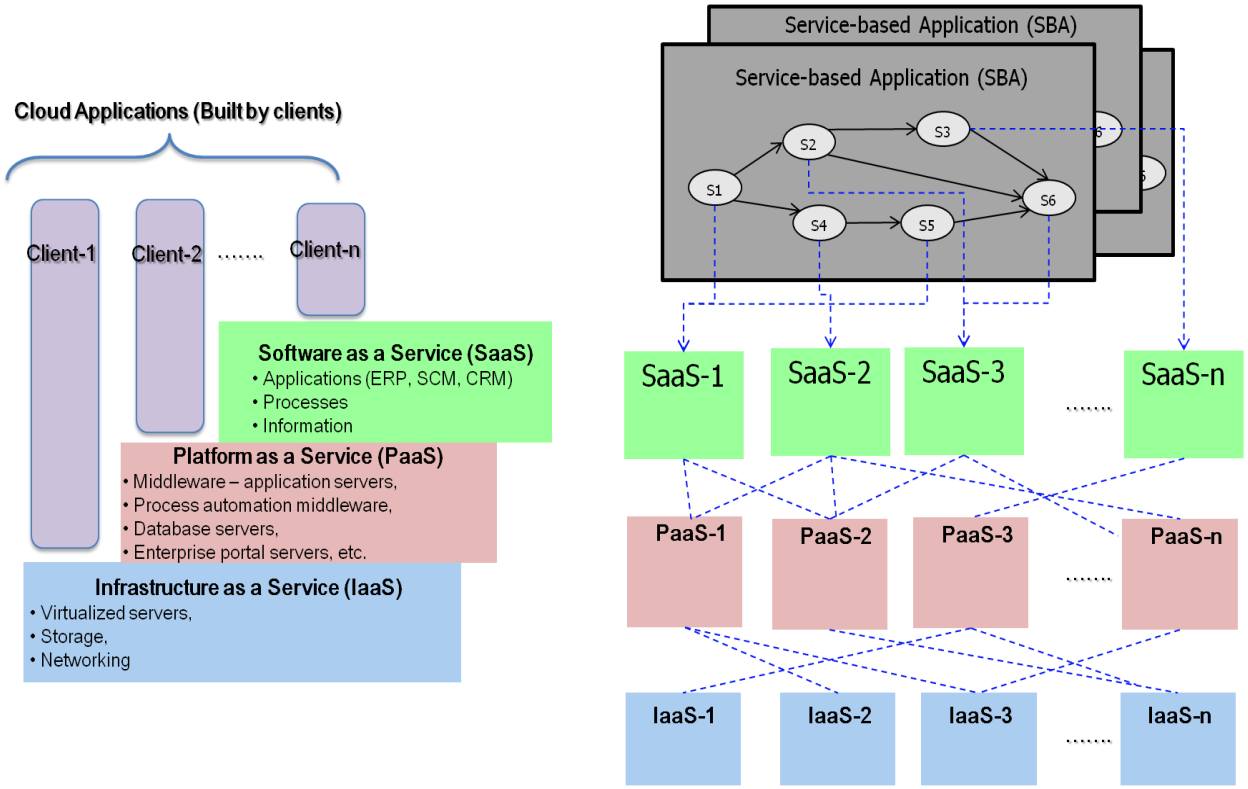
From the preceding discussion it is apparent that the current cloud solutions are fraught with problems:

- They introduce a monolithic SaaS/PaaS/IaaS stack architecture where a one-size-fits-all mentality prevails. They do not allow developers to mix and match functionality and services from multiple application, platform and infrastructure providers and configure it dynamically to address application needs.
- They introduce rigid service orchestration practices tied to a specific resource/infrastructure configuration for the cloud services at the application level.
- The above points hamper the (re)-configuration and customization of cloud applications on demand to reflect evolving inter-organizational collaborations.

There is clearly a need to mash up cloud services from a variety of cloud providers to create a Service-based Application (SBA). This type of integration allows the tailoring of services to specific business needs using a mixture of SaaS, PaaS and IaaS. SBAs in the cloud are normally delivered in terms of end-to-end business processes that are usually syndicated with other external SaaSs (possibly provided by diverse SaaS providers). SBA developers need to couple their applications in whole or part with external SaaS offerings to provide opportunities related to other areas of their clients' business. Hence, a causal connection of SBA-level operations to configurable supporting SaaS, PaaS and IaaSs is required to allow the full automation and optimization of SBA development and maintenance activities.

The above mentioned approach promotes autonomous services (at all levels of the cloud stack) that adhere to the well-established software engineering principle of "separation of concerns" to minimize dependencies. This approach allows any service at any layer, *i.e.*, SaaS, PaaS or IaaS, to be appropriately composed 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. Furthermore, SBA developers can rely on existing work to solve incompatibility issues that may occur during the service composition, e.g., the work in [6] solves the incompatibility issues between structural service interfaces and the work in [7] helps to adapt service protocols. At the same time, the above approach also allows multiple (and possibly composed) PaaS and IaaS options for deploying a given SaaS. To ensure the portability between different cloud providers, standards like the Open Virtualization Format (OVF) [8] may be used as part of an IaaS to describe the virtual machine packaging in a uniform way. Our proposed approach takes all these issues into consideration and thus leads to a *syndicated multi-channel delivery model*, which is illustrated in Figure 1, where it is contrasted with the monolithic cloud stack solutions that permeate the cloud today. Monolithic cloud solutions are shown to enforce one-way vertical deployment "channels".

Figure 1. Monolithic *vs*. Syndicated Multi-channel Cloud Delivery Model.



(a) Monolithic Delivery Models (b) Syndicated Multi-channel Delivery Model

Cloud Blueprinting is a novel powerful solution allowing SBAs to dynamically run on top of federated cloud virtualization solutions. The SaaS components of a SBA are abstracted and described in a series of templates to provide a fast, simplified method for provisioning and automating cloud services. Cloud Blueprinting seeks to simplify deployment by hiding away the complexity of deploying a SBA by helping to manage all configuring of the underlying middleware and integrating with optimal PaaS and IaaS options. It achieves portability across clouds and cloud providers to leverage the benefits of elasticity and scale. It supports a flexible top-down continuous closed-feedback loop service refinement and improvement approach. Application-level decisions regarding the virtualized SBAs are correlated with and used to drive the resource provisioning and adjust the workload and traffic to automate the dynamic configuration and deployment of application instances onto available cloud resources. The blueprinting approach promotes service virtualization by allowing an SOA-inspired approach that supports independent layering within a typical cloud stack. For example, a developer can choose to compose services from multiple SaaS providers into a coherent and integrated SBA, which s/he can then synthesize with platform services from one or more PaaS providers, and deploy on a federated cloud infrastructure of multiple IaaS providers.

Our previous work [9] has defined a *Blueprint* as a uniform abstract description of a cloud service offering that hides all specific technical details and complexities to facilitate the SBA developers with the selection, customization and composition of cross-layered cloud services across various syndication channels. It also proposed a structural schema called the *Blueprint Template* that can be used to describe and customize a blueprint. After releasing the first version of the template within our

industrial partners in the EC's 4caaSt FP7 project [10], we have received useful comments and feedback that helped us work towards an improved version of the template in this paper. Furthermore, this paper introduces a universal *Blueprint Lifecycle* that shows how cloud-based services described by blueprints are designed, composed, and ultimately deployed on a cloud infrastructure, which will results in a running SBA in the cloud.

The rest of this paper is structured as follows: in Section 2 we introduce a cloud computing scenario that has been defined by the 4caaSt community and will be used as the running example throughout the paper; Section 3 reviews related work to show why the blueprinting approach is necessary; in Section 4 we elaborate the *Blueprint* concept, its fundamental structure and different types of blueprint; in Section 5 we propose the *Blueprint Template* to capture cross-layer cloud service offerings; Section 6 proposes a six-step lifecycle for the blueprinting approach that supports the engineering of SBAs in the cloud; in Section 7 we present our empirical evaluation approach and blueprint prototype implementation; finally, Section 8 summarizes the paper and discusses future issues.

2. Motivating Scenario

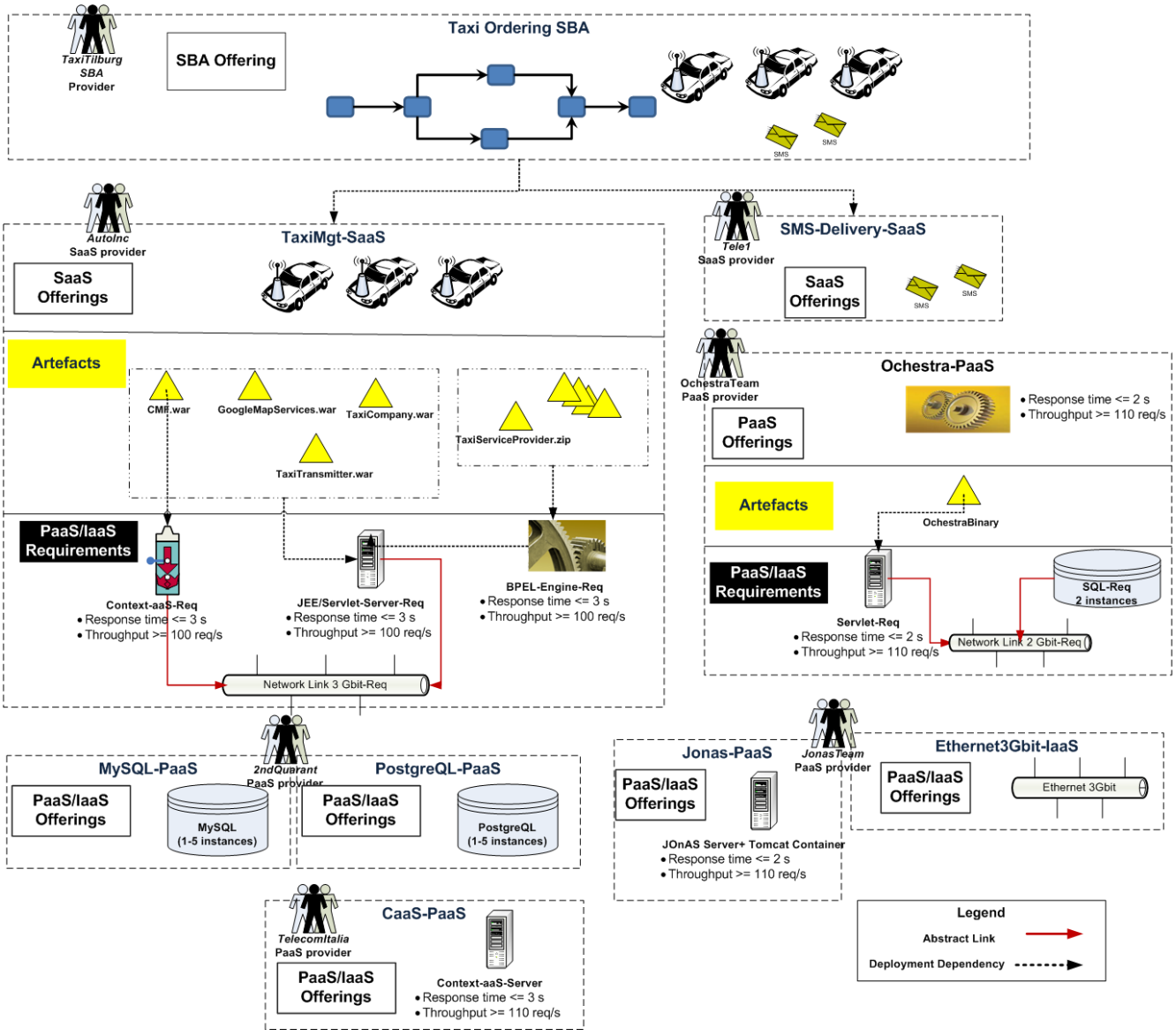
This section presents an enterprise cloud computing scenario developed for the EC's 4CaaSt project. Several industrial IT companies like Telefonica, Telecom Italia, Ericson, 2ndQuadrant and SAP have been involved in the definition of this scenario, which is presented as a simplified description of today industrial reality. The aim is to provide a case study that is sufficiently complex for capturing real problems and sufficiently simple for proposing and validating research solutions. The scenario promotes a uniform description for cloud service offerings to exist so that the design, configuration, and deployment of a cloud-based solution can occur. The scenario in Figure 2 is about developing a process-based application in the cloud that can receive an order for a taxi from the customer by SMS, check the current status of the taxi fleet, allocate a taxi, and finally confirm the order by SMS. It contains seven actors that collaborate in a marketplace where their cloud services can be offered and purchased. In the following, each actor is described.

There are three actors in the scenario that provide complete monolithic PaaS and IaaS offerings hosted on their in-house infrastructure. **2ndQuarant** is a PaaS provider specializing in storage services. It offers the relational PostgreQL Service (*PostgreQL-PaaS*) or the relational MySQL Service (*MsSQL-PaaS*) as two alternative solutions for relational database, where each consumer can request for up to five instances of a solution. Another PaaS provider is **TelecomItalia** that provides the native Context-as-a-Service offering for context information delivery (*CaaS-PaaS*) [11]. **JonasTeam** is another PaaS provider that has two offerings in the marketplace. The first one is a JOnAS server to host Java applications including a Tomcat container to support the execution of servlets (*Jonas-PaaS*). The second one is an IaaS offering for a 3Gbit Ethernet network link (*Ethernet3Gbit-IaaS*).

OrchestraTeam is also a PaaS provider that offers an Orchestra BPEL Engine (*Orchestra-PaaS*). OrchestraTeam differentiates itself from other PaaS providers in a way that their offering *Orchestra-PaaS* is not complete and relies on external cloud platform and infrastructure resources for the deployment. In particular, the binary implementation file of the Orchestra engine (*OrchestraBinary*) needs a Servlet container (*Servlet-Req*) for the deployment. It is also required that the *Servlet-Req* is connected to two relational databases (*SQL-Req*) through a 2Gbit network (*NetworkLink2Gbit-Req*). To maintain the

Quality of Service (QoS) promised in their offering, *i.e.*, response time <= 2s and throughput >= 110 request/s, OrchestraTeam requires the *Servlet-Req* to be constrained by these QoS assertions as well.

Figure 2. The 4caaSt Cloud Computing Scenario.



On the SaaS level, **Tele1** is a telecom company that provides a basic SMS Delivery SaaS (*SMS-Delivery-SaaS*) that is hosted on their in-house platform. In contrast, **AutoInc** is an established Medium Enterprise and has spotted a business opportunity providing fleet vehicle management in the Netherlands. They plan to deploy their business functions as a Taxi Management SaaS (*TaxiMgt-SaaS*), since this provides ubiquitous and common access for their prospective customers, e.g., taxi providers and car-hiring providers. To implement the solution, AutoInc has contracted a software consultancy who wrote the taxi management software in Java and BPEL. The software hence contains a number of war files, bpel files and other binary and configuration files (see Figure 2 for more details). The software hence requires a JEE Server (*JEE/Servlet-Server-Req*) including a Servlet v2.5 for deploying the war files for the web interface, a BPEL Engine (*BPEL-Engine-Req*) for deploying the core BPEL process files of the taxi application, and a context information provider

(*Context-aaS-Req*) that is needed by the *CMF.war* file. The *BPEL-Engine-Req* is required to be deployed on the *JEE/Servlet-Server-Req*, and the *JEE/Servlet-Server-Req* is required to be connected to the *Context-aaS-Req* and to the outside through a network link with 3Gbit bandwidth (*NetworkLink3Gbit-Req*).

TaxiTilburg is a provider of a taxi ordering application that is delivered to the customers in terms of a workflow (*TaxiOrdering-SBA*). In order to implement the steps of their workflow, TaxiTilburg completely relies on the SaaS offerings *TaxiMgt-SaaS* of AutoInc and *SMS-Delivery-SaaS* of Tele1. To maintain the predefined Quality of Service (QoS) level, TaxiTilburg insists AutoInc to specify the QoS requirements for the prospective platform's resources of the *TaxiMgt-SaaS* offering. In particular, all the required platform resources *JEE/Servlet-Server-Req, BPEL-Engine-Req, Context-aaS-Req* should ensure a throughput >=100 req/s and a response time <= 3s.

TaxiTilburg also has policy constraints that cannot be violated by the entire end-to-end system landscape, *i.e.*, all of the application, platform and infrastructure systems involved in the creation of this SBA. These policy constraints prescribe that all the data must be stored only within the Netherlands and all the data communications must be over the Secure Socket Layer (SSL).

In summary, TaxiTilburg has designed their *TaxiOrdering-SBA* offering as an SBA, since the required *SMS-Delivery-SaaS, TaxiMgt-SaaS* and all the other required platform and infrastructure resources are not under their direct control. Deploying the *TaxiOrdering-SBA* on the cloud requires a syndication of the SaaS, PaaS and IaaS offerings of other providers. As we mentioned in the introduction in the previous Section 1, there is still a lack of support in building such an SBA in the cloud. Based on this case study, we will describe in the following sections the cloud blueprinting approach to select and compose different cloud service offerings to develop such an SBA.

3. Related Work

Much of the recent work on cloud-based application development concentrates on the infrastructure level. For example, the ability to manipulate, integrate and customize cloud service descriptions across different cloud providers has been studied in [12] that has IaaS, application and deployment orchestrators but falls short of proposing a solution for the problem at hand. The DMTF has published standards such as the Open Virtualization Format (OVF) [8], to provide an open packaging and distribution format for virtual machines. The work in [13] is grounded on the OVF and proposes a service definition language for deploying complex Internet applications in federated IaaS clouds. These applications consist of a collection of virtual machines (VMs) with several configuration parameters (e.g., hostnames, IP addresses and other application specific details) for software components (e.g., web/application servers, database, operating system) running on the VMs. In [14], this language has been extended into a *service definition manifest* to serve as a contract between a service provider and the infrastructure provider. In this contract, architectural constraints and invariants regarding the infrastructure resource provisioning for an application service are specified and can be used for on-demand cloud infrastructure provisioning at run-time. Using the service definition manifest to specify the structure of a SaaS application, *i.e.*, the SaaS components and their required Virtual Execution Environments (VEE), the Reservoir architecture in [15] can automatically provision the VEE instances that can run simultaneously without conflict on a federated cloud

infrastructure of multiple providers. KPI monitoring mechanisms and elasticity rules in the manifest act as a contract that guarantees the required Service Level Agreement (SLA) between the SaaS provider and the Reservoir architecture.

Model-driven approaches are also employed for the purpose of automating the deployment of complex IaaS services on cloud infrastructure. For instance, the work in [16] proposes a virtual appliance model, which treats virtual images as building blocks for IaaS composite solutions. Virtual appliances are composed into virtual solution model and deployment time requirements are then determined in a cloud-independent manner using a parameterized deployment plan. In a similar way, the approach in [17] describes a solution-based provisioning mechanism using composite appliances to automate the deployment of complex application services on a cloud infrastructure.

In practice, many products have appeared on the market that target the integration issue among different IaaS clouds. The Nimbula product [18] allows users to build their own private IaaS clouds that are interoperable with the Amazon's EC2 IaaS offerings, *i.e.*, users can swap in and out virtual images between their private Nimbula cloud and the public EC2 cloud. Besides, the most appealing attempt for API standardization is the Deltacloud solution [19] that provides a façade API for customers to consume IaaS offerings from different providers.

In conclusion, the above mentioned cloud initiatives in developing cloud-based applications, especially the ones related to the OVF, target the infrastructure level only. *I.e.*, they allow the specification of infrastructure constraints for deploying the applications directly on (federated) data centers but fail to cover the holistic picture of a top-down development of SBAs on the cloud that can guide developers in selecting, resolving and composing SaaS and PaaS offerings.

On the cloud application level, we observe some initial efforts in proposing methodologies for application development on the cloud. The authors in [20] propose a systematic process for developing high-quality cloud SaaSs, taking into considerations the key design criteria for SaaSs and the essential commonality/variability analysis to maximize the reusability. Although this approach claims to develop cloud-based SaaSs, it does not discuss about the cloud support for the deployment environment of the SaaSs. A cloud-agnostic middleware is introduced in [21] that can sit on top of many PaaS/IaaS offerings and enable a platform-agnostic SaaS development. They provide a meta-model for describing SaaS applications and their needed cloud resources, and APIs and middleware services for the deployment. The connection between SOA and cloud computing has been studied by the Service-Oriented Cloud Computing Architecture (SOCCA) proposed in [22]. Using the SOCCA, developers can build SaaS applications following an integrated SOA framework. Cloud platform and infrastructure resources may be discovered by a Cloud Broker Layer and a Cloud Ontology Mapping Layer for deploying the SaaS components. The multi-tenancy feature of cloud computing is also supported by SOCCA where multiple instances of SaaS applications or components can be provided to multiple tenants. Although the SOCCA is a useful reference architecture for developing cloud-based SaaSs following the SOA paradigm, methodology support is lacking here including a concrete definition language for the SaaS applications and components, a composition approach for discovering and composing the cloud resources, and the ability to specify and resolve end-to-end constraints of SaaS applications that might affect the underlying cloud resources.

The Cafe application and component templates [23] are a relevant approach for cloud-based SaaS development. Cafe provides an ad-hoc composition technique for application components and cloud

resources following the Service Component Architecture (SCA). However, this approach requires SaaS developers to possess intricate technical knowledge of the application architecture and the physical cloud deployment environment to select and compose the right application components and cloud resources.

In practice, an attempt to provide a template-based approach for using cloud services is available from Amazon through their AWS CloudFormation offering [4]. This template enables AWS developers to specify a collection of AWS cloud resources and the provisioning of these resources in an orderly and predictable fashion. Another template-based support that enables the orchestration of tasks for managing virtual machines is VMware vCenter Orchestrator [24]. Nevertheless, both the AWS CloudFormation and the VMware vCenter Orchestrator work only on their own IaaS resources, *i.e.*, Amazon's cloud infrastructure resources and VMware vCenter Servers respectively, and thus lacks interoperability across IaaS providers.

In summary, existing work mostly aims to propose standards and techniques for only certain aspects and thus fails to cover the full picture of cloud computing, e.g., providing models and formats for only infrastructure resources, or describing only the functional specification. Furthermore, these standardization efforts do not aim to assist the cloud-based application developers to select, customize and compose various cross-layered cloud services across vendors according to their application requirements. We propose the *Blueprint* concept in the next section as a uniform representation to capture the comprehensive knowledge of a cloud service offering to support cloud application developers during the various development phases.

4. The Blueprint Concept

Taking into account the shortcomings of existing approaches in cloud computing pointed out in Section 3, this section introduces the concept of *Blueprint* in Section 4.1 and continues by introducing different types of blueprints that can be used to capture cloud service offerings in Section 4.2.

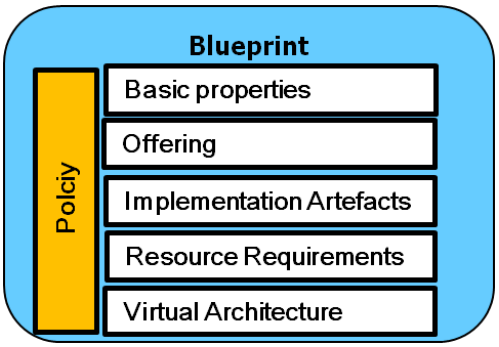
4.1. Fundamental Blueprint Structure

We define Blueprint as a uniform abstract description of a cloud service offering that abstracts away from all specific technical details and complexities to facilitate the cloud application developers with the selection, customization and composition of cross-layered cloud services across various vendors. A blueprint has the following fundamental structure, as illustrated in Figure 3:

- **Basic properties**: Some basic description properties of a blueprint including its unique ID, ownership, release date, version info, *etc*.
- **Offering**: The description of one or many cloud service offerings including their names, functionalities, signature interfaces, interaction protocols, elasticity offerings, and QoS offerings.
- **Implementation Artifacts**: The description of the artifacts that implement the offerings. Implementation artifacts may include the binary files, configuration files or some deployable web packages.
- **Resource Requirements**: The description of the required cloud resources including their QoS requirements. The resource requirements are needed to deploy the implementation artifacts.

- **The Virtual Architecture (VA)**: this part is specified by the application developers to capture the desired virtual architecture in terms of the interdependencies and interconnections between the offerings, implementation artifacts and resource requirements across blueprints.
- **Policy**: describing the policy constraints that may not be violated by any element (the offering, implementation artifacts, resource requirements) of a blueprint.

Figure 3. Fundamental Blueprint Structure.



4.2. Types of Blueprints

A blueprint can be classified based on its status. If a blueprint has already been submitted to a marketplace repository, such as the one provided by SAP [25], and thus can be purchased and reused by other application developers, it is called a *Source Blueprint*. In contrast, a blueprint under development is called a *Target Blueprint*. Application developers can use the blueprint template to describe their offerings in a target blueprint that may reuse other source blueprints and finally be deployed on a cloud infrastructure. For instance in the scenario in Section 2, TaxiTilburg may describe their *TaxiOrdering-SBA* offering in a target blueprint called *TaxiTilburg-Blueprint* and AutoInc may describe its *TaxiMgt-SaaS* offering through the *AutoInc-Blueprint* target blueprint. Other blueprints in the scenario are classified as source blueprints that are available in a marketplace repository and can be selected and reused by the application developers of TaxiTilburg or AutoInc.

A cloud service offering is incomplete if there are still resource requirements on which it relies. To complete this offering, one has to retrieve further offerings of other source blueprints from the marketplace repository to fulfill the resource requirements. Hence, we can also distinguish the two following types of blueprints depending on the completeness of their offerings:

- *Resolved blueprints*: A blueprint is resolved if all of its offerings are complete and ready to be deployed and consumed. For instance in our scenario in Section 2, the blueprints used to describe *SMS-Delivery-SaaS, CaaS-PaaS, PostgreQL-PaaS, Jonas-PaaS, and Ethernet3Gbit-IaaS* offerings fall under this category. According to our assumption, they contain only complete offerings.
- *Unresolved blueprints*: At least one of the offerings captured in this type of blueprint is incomplete, *i.e.*, it is not yet ready to be consumed and still relies on the underlying resources for the deployment. Examples of this type of blueprint are the ones capturing the *TaxiOrdering-SBA, TaxiMgt-SaaS, and Orchestra-PaaS* offerings in the case study. In this case, the service provider needs to specify in the blueprint the resource requirements. He relies on the application developer who is responsible for the "blueprint resolution". This task includes searching for

available blueprints in the repository to fulfill the stated resource requirements. Unresolved blueprints are inherently customizable since the consumers will also be given the ability to customize the underlying configurations of the blueprints after they have been resolved.

A target blueprint is always unresolved, as it requires other source blueprints to fulfill its resource requirements. Source blueprints can be both resolved and unresolved. In case an unresolved source blueprint is retrieved by the application developer from the marketplace for the reuse, it iteratively becomes a new target blueprint that requires the "blueprint resolution" task.

As pointed out in the introduction, our approach is to compartmentalize the monolithic cloud offering in three different layers (SaaS, PaaS, IaaS) and use the blueprints to describe each of these layered resource offerings. The advantage is that blueprints can be flexibly (re-)assembled in different configurations to form a complete SBA. The first step to enable such a flexible syndication of cross-layered blueprints is to support a formal template for describing it in an unambiguous manner. In the next Section 5 we introduce the Blueprint Template as a standard way to describe cross-layered cloud service offerings.

5. Blueprint Template

A blueprint has been defined in the previous Section 4 as a uniform implementation-agnostic description of a cloud service offering that abstracts away from all specific technical details and complexities. Blueprints are used to simplify the SBA development through the syndication of multi-channel composition of cross-layered cloud services across various providers. Our previous work [9] proposed a structural schema called "Blueprint Template" that can be used to describe a blueprint. After releasing the initial version of the template within the 4caast community we have received useful comments and feedback that help us work towards an improved version in this section. Figure 4 illustrates the consolidated version of the Blueprint Template. Using the template, the blueprint provider can describe (instantiate) blueprints that capture their cloud service offerings. In Figure 5, a sample instantiated blueprint is introduced that captures the offering of AutoInc in the motivating scenario from Section 2. The Blueprint Template is divided into template sections; each has a set of proposed blueprint properties. Please note that the template is extensible and will be continuously improved, *i.e.*, if more properties are needed in a particular section, they can be added using the following data structure (property name, property type, [property value range]). In the following, each template section is dissected with a proposed set of properties, alongside with the example provided in Figure 5.

Figure 4. The Blueprint Template.

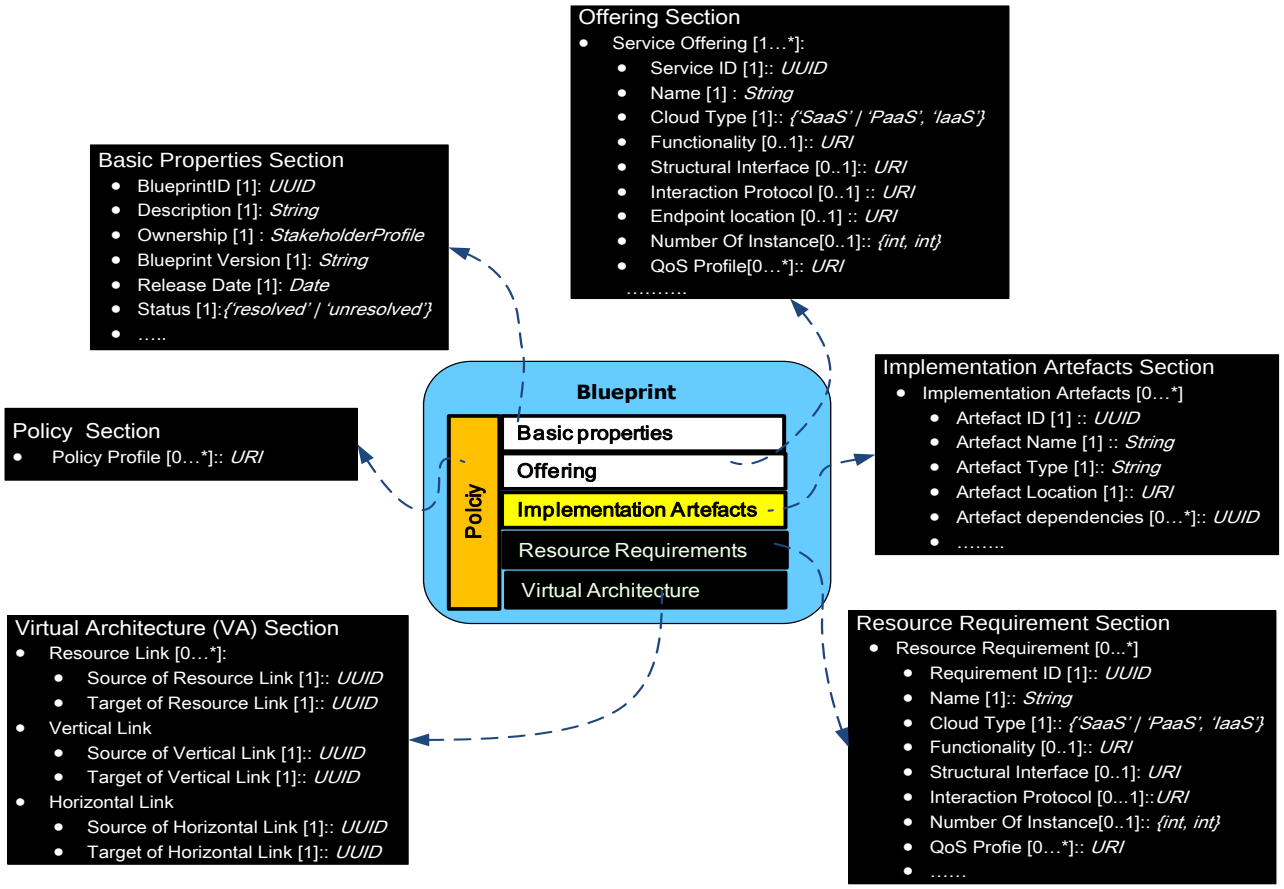
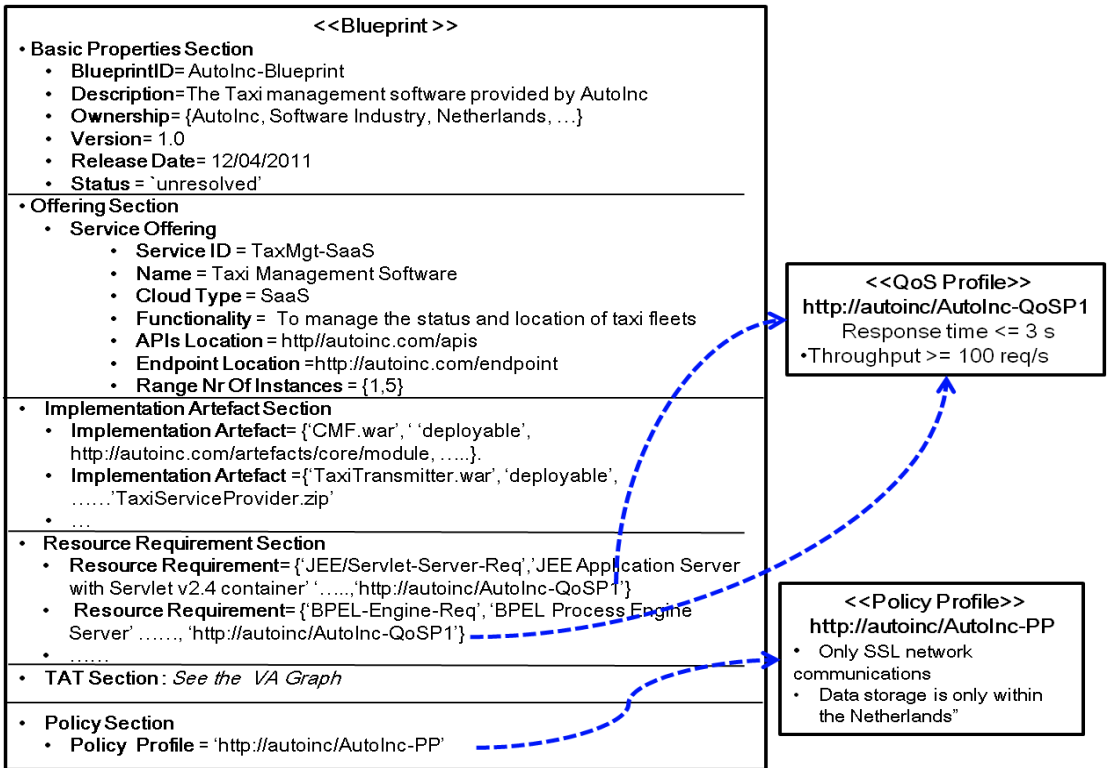


Figure 5. Example of using the Blueprint Template.



5.1. Basic Properties

Most importantly, the Basic Properties section contains an id (BlueprintID) using the UUID type for uniquely identifying a blueprint. This id is used for indexing a blueprint in the blueprint repository as well as referencing the included blueprints in case one would like to offer a blueprint containing a bundle of other included offerings. Apart from the id, the basic properties include a short textual description, the ownership information, the version information, the release date and the status of the blueprint. While other properties can be described using simple types like UUID, String, integer, etc., the ownership may need a more sophisticated data structure, e.g., a StakeholderProfille complex type that contains the name of the blueprint provider, its industry sector, location information, etc. The status property of a blueprint indicates whether that blueprint has been resolved yet. For instance, our AutoInc-Blueprint blueprint in Figure 5 has not been resolved and still has the status "unresolved".

5.2. Offering Section

This section contains one or more service offerings, each with the following nested template properties:

- A unique ID with type of UUID to identify a service offering.
- The name of the cloud resource that is offered as a service to the consumer. The cloud resource could be an entire SBA, single software application, platform, or infrastructure resource. Inspired by the Debian package management approach, naming of a service offering follows a keyword-based approach. This enables an easy searching and matching of offerings that provide the needed cloud resources.
- The type of the cloud resource offered as a service, e.g., SaaS, PaaS or IaaS according to the classification in [26].
- The functionality of the cloud service offering should be described in such a way that augments the consumer to query the blueprint from a blueprint repository. To enable more accurate search and matching of offerings, we suggest using the OVF standard [8] for describing the functionality of PaaS and IaaS offerings, and the service capability description template in [27] for describing the functionality of SaaS offerings.
- The signature-related properties of a cloud service including the URI locations to download the APIs and the endpoint location for programmatic interactions with the cloud service. The APIs of a cloud service comprise of the structural interfaces, e.g., described in WSDLs, and the interaction protocols, e.g., described in abstract BPEL. Using the APIs, a consumer is able to programmatically interact with the cloud service.
- The elasticity offering is specified in terms of the minimum and maximum number of instances of the cloud service (*RangeNrOfInstances*) that can be offered per user session.
- QoS properties of the cloud service can be specified in a number of separate profiles (QoS Profile) using an add-on templates or external languages, e.g., WS-Policy, SLAng, *etc*. The blueprint template allows the specification of URI pointers referencing these separate profiles.

5.3. Implementation Artifacts Section

This section is important for the application developer who is responsible for the deployment and provisioning of a blueprint. It encompasses the following information: an artifact id for uniquely identifying an artifact, an artifact name, an artifact type indicating whether this artifact is a software binary, a composition script, a database startup file or some other kinds of configuration files, an artifact location for downloading, and some artifact dependencies pointing to the other artifacts that have to be executed before executing this one.

Examples of implementation artifacts can be found in Figure 5. *TaxiTransmitter.war* is the ID of a deployable war file that implements parts of the Taxi application SaaS. However, it can work only after another artifact *TaxiServiceProvider.zip*, which is a zip file containing the BPEL scripts for the taxi application, has been deployed properly.

5.4. Resource Requirements Section

A list of cloud resource requirements needed for deploying and provisioning a blueprint is specified in this section. This section supports the application developers in searching for cloud service offerings in a blueprint repository. Each resource requirement is specified with a resource ID, a name, the required functionality, the required Range Number of Instances, the required API (if needed) including the structural interface and interaction protocols and a set of URI references pointing to QoS Profiles that contain the QoS assertions for this resource requirement. Similarly to the offering section, the name of the resource requirement follows a keyword-based naming approach and the required functionality can be described in a more expressive way using the external add-on templates like OVF or the one in [27]. The referenced QoS profiles can be described using an existing external language like WS-Policy or SLang.

As an example of resource requirements in Figure 5, the *AutoInc-Bluerpint* blueprint needs an instance of a JEE Application Server including a Servlet Container. This requirement has a unique id *JEE/Servlet-Server-Req* and is associated with some required QoS properties defined in the *http://autoinc/AutoInc-Req-QoSP1* profile, which prescribes that the required resource should respond faster than 3s and provide the throughput greater than 100 requests per second.

5.5. Virtual Architecture (VA) Section

An element in a blueprint, *i.e.*, a service offering, implementation artifact, or resource requirement, may have an interdependency or interconnection relationship with another blueprint element. The VA Section allows specifying the following relationships between elements within a blueprint or across blueprints, each with a Source and a Target element.

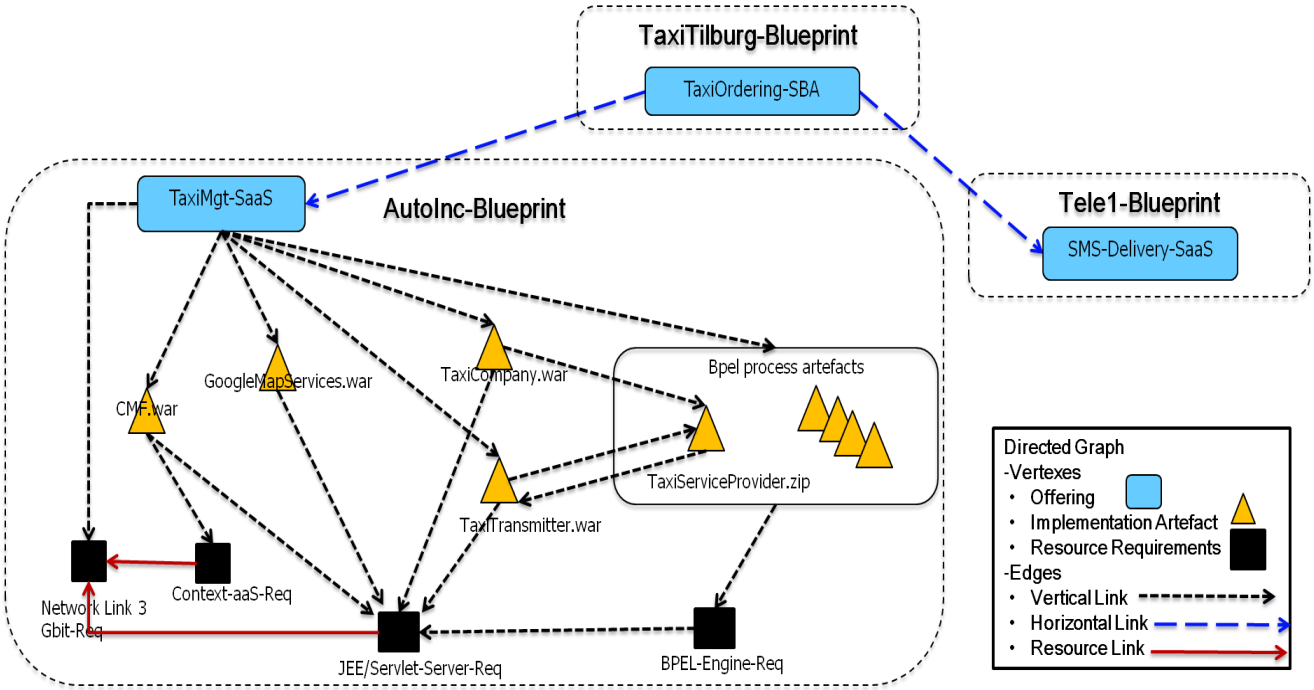
- Vertical Link: This relationship indicates a “direct” deployment dependency between a blueprint element e and another blueprint element e', *i.e.*, e needs to be deployed on e'. For instance, an implementation artifact directly depends on one or more resource requirements for its deployment. Vertical Links are transitive, which results in the so-called “indirect” deployment dependency, or the “indirect” vertical link.

- Horizontal Link: This relationship indicates a "direct" functional dependency between a blueprint element e and another blueprint element e', *i.e.*, e reuses the functionality of e'. For instance, a service offering reuses another service offering and hence, directly depends on its functionality. Horizontal Links are transitive, which results in the so-called "indirect" functional dependency, or the "indirect" horizontal link.
- Resource Link: We consider a virtual network resource as an IaaS service offering or IaaS resource requirement in a blueprint. A resource link is used to allocate blueprint elements to a virtual network resource. Hence, the target of a resource link is always a virtual network resource captured in either an IaaS service offering or an IaaS resource requirement.

Figure 6 illustrates the joint VA graph of the *TaxiTilburg-Blueprint, Tele1-Blueprint, and AutoInc-Blueprint* blueprints in the scenario from Section 2. This graph indicates:

- The *TaxiOrdering-SBA* offering requires the *TaxiMgt-SaaS* offering of AutoInc and the *SMS-Delivery-SaaS* offering of Tele1.
- The *CMF.war, GoogleMapServices.war, TaxiCompany.war, TaxiTransmitter.war* implementation artifacts require a JEE application server that includes a Servlet v2.5 container (*JEE/Servlet-Server-Req*).
- Other implementation artifacts of the *AutoInc-Blueprint* blueprint require a BPEL Engine (*BPEL-Engine-Req*), which is supposed to be deployed on the *JEE/Servlet-Server-Req*.
- The required JEE application server (*JEE/Servlet-Server-Req*) is connected to the required Context-as-a-Service and to the outside by a network link 3Gbit (*NetworkLink3Gbit-Req*).

Figure 6. A sample Virtual Architecture (VA) graph.



In summary, the VA is an essential part of the Blueprint Template that specifies the to-be virtual architecture topology across blueprints. The VA graph in Figure 6 also indicates that the *TaxiMgt-SaaS*

offering of the *AutoInc-Blueprint* blueprint is "unresolved" as it still contains a number of resource requirements. In the further development phases of the blueprint, the resource requirements need to be resolved by searching and selecting appropriate blueprints from the blueprint repository. The *AutoInc-Blueprint* blueprint becomes "resolved" if all the resource requirements can be fulfilled by the offerings of the newly retrieved blueprints.

5.6. Policy Section

Policy constraints are the end-to-end global constraints that may not be violated by the blueprint and its resource requirements. The blueprint provider can specify the policy constraints in a Policy Profile using any existing languages such as WS-Policy, RuleML, *etc*.

For example in Figure 5, the *AutoInc-Blueprint* blueprint of AutoInc is constrained by two policy constraints defined in the policy profile identified by the URI *http://autoinc/AutoInc-PP*. These constraints state that all network communications have to be secured through the Secure Socket Layer (SSL) and all the data have to be stored within the Netherlands. Since they are the global end-to-end constraints specified for the *AutoInc-Blueprint*, all the prospective blueprints that will be selected to fulfill the resource requirements are constrained by them as well.

6. Blueprint Lifecycle Support for Engineering Cloud-Based Applications

To identify how a blueprint is specified, combined and customized with other blueprints and ultimately deployed to the cloud, this section proposes a lifecycle for engineering SBAs within which blueprints are used. Identifying the activities within the lifecycle allows us to analyze each phase the blueprint goes through to understand what it is for, what the goals and benefits are and how each activity can be supported by processes, standards and automation. The blueprint lifecycle we propose is shown in Figure 7. In particular, this figure depicts how the blueprint template is used to initiate a six-step, iterative lifecycle that results in a running cloud application. Currently, our focus is put on the first three phases of the blueprint lifecycle, basically the design, resolution and customization of a new target blueprint. The rest of the lifecycle remains as future issues and will not be discussed in depth in the following subsections.

6.1. Phase 1: Target Blueprint Design

The blueprint lifecycle begins when a developer wishes to design a new SBA in the cloud. To do this, he uses the blueprint template, introduced in Section 5, to specify all the necessary information about the new cloud application. The blueprint template allows the developer to describe not only the functional capabilities of the new application and the qualities it should have (e.g., minimum and maximum values for the end-to-end performance, availability and/or throughput of the complete application), but also the required resources to deploy and provision this application on the cloud. At the end of this process, when the developer has completed their specification, they have created a *Target Blueprint* that contains the application offering, the resource requirements, policy constraints, and a to-be architecture topology expressed through the relationships between the offering and requirements.

As an example of this phase, the sample *AutoInc-Blueprint* blueprint in Section 5 is considered as a *Target Blueprint* that has been designed using our proposed blueprint template.

6.2. Phase 2: Target Blueprint Resolution

In order to transform the *Target Blueprint* into a description that can be deployed on cloud infrastructure, the developer must resolve the resource requirements contained in the target blueprint with concrete SaaS, PaaS or IaaS offerings of other cloud providers. These service offerings are described in *Source Blueprints*, which are specified using the same blueprint template used to develop the target blueprint described in the previous phase. The source blueprints can be found in well-known location(s) (e.g., a marketplace or repository) and are retrieved and matched to the requirements in the target blueprint using a resolution process. Matching of a source blueprint's offering against a target blueprint's requirement comprises of the cloud resource name matching, functionality matching, QoS matching and policy constraint satisfaction. We currently use the keyword-based matching approach for resource name matching. Functionality between a cloud service offering and a cloud resource requirement may rely on external procedure that compares two functionality descriptions, e.g., comparing two OVF documents. Since the QoS and policy properties in our blueprint may be described by the WS-Policy standard, we rely on the existing work in [28] that supports the semantic matching of WS-Policy constructs for the QoS matching and policy constraint satisfaction between the source and target blueprint.

Figure 7. The Blueprint Lifecycle.

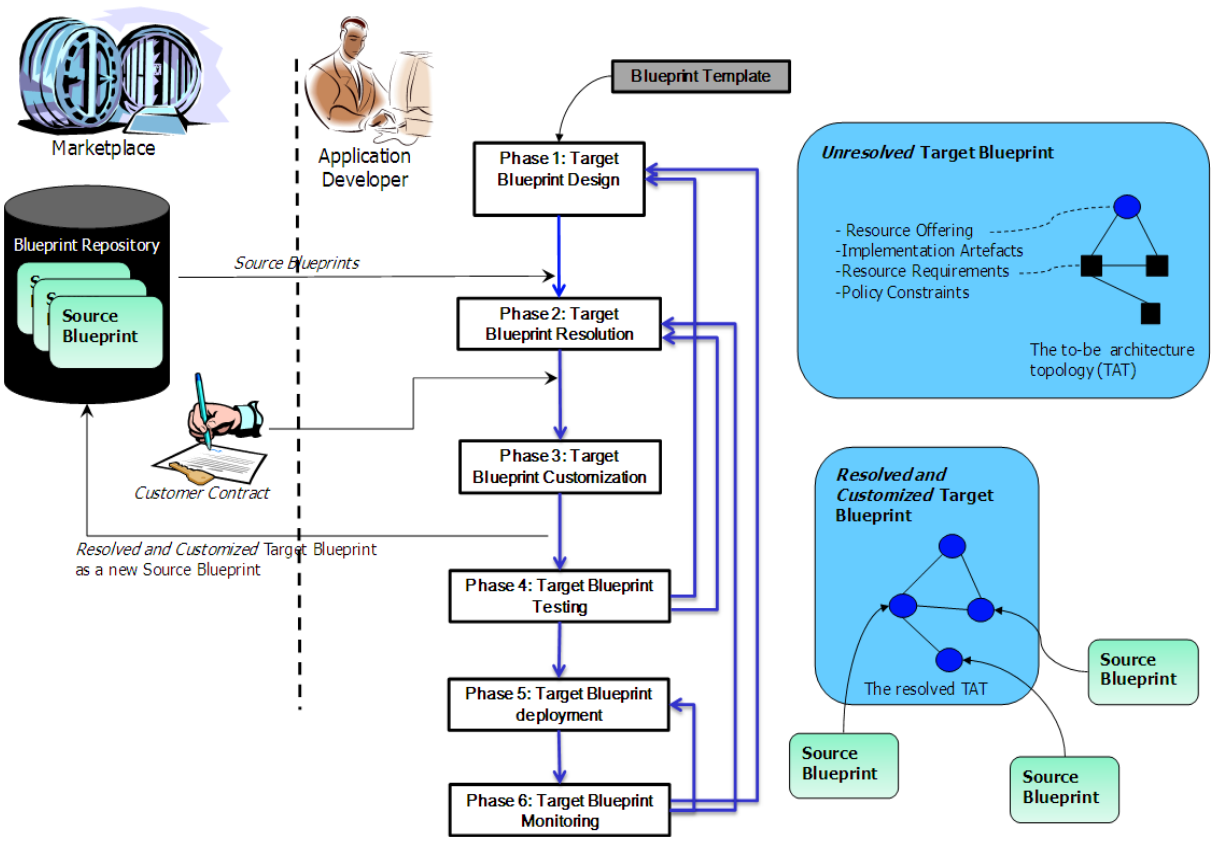
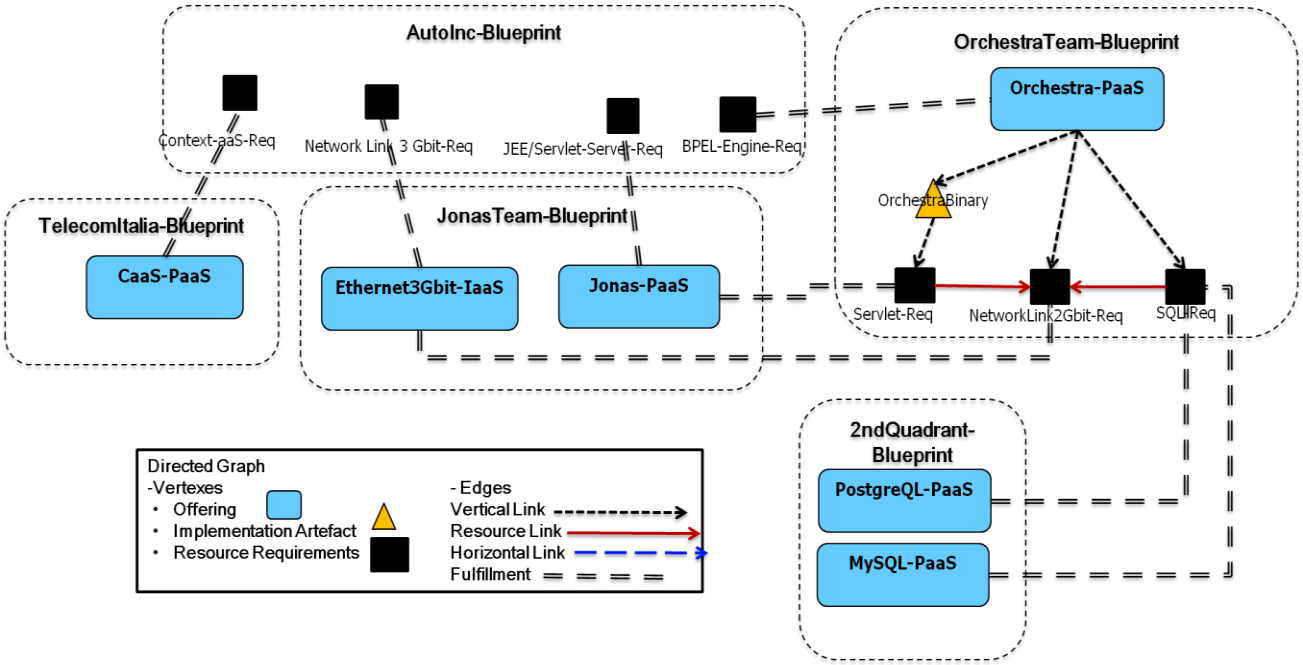


Figure 8 illustrates the resolution of our sample target blueprint AutoInc-Blueprint, i.e., how to fulfill its resource requirements using the offerings of other source blueprints. Note that the target blueprint resolution is an iterative process, since the source blueprints might still contain resource requirements and iteratively becomes a new "unresolved" target blueprint. For instance in Figure 8, the offering Orchestra-PaaS of the blueprint OrchestraTeam-Blueprint is reused to fulfill the BPEL-Engine-Req requirement, yet it still contains three further resource requirements that have to be fulfilled for the deployment.

Figure 8. Resolving the sample Target Blueprint *AutoInc-Blueprint*.



The result of the resolution process is a set of alternative "resolved" target blueprints that provides options/alternatives to the developer as to how the target blueprint's requirements can be satisfied by source blueprints (*i.e.*, specifications of actual cloud services). For instance from Figure 8, resolving the requirement *SQL-Req* of the *OrchestraTeam-Blueprint* blueprint will result into two alternative "resolution results", one with the *PostgreQL-PaaS* offering and the other with the *MySQL-PaaS* offering.

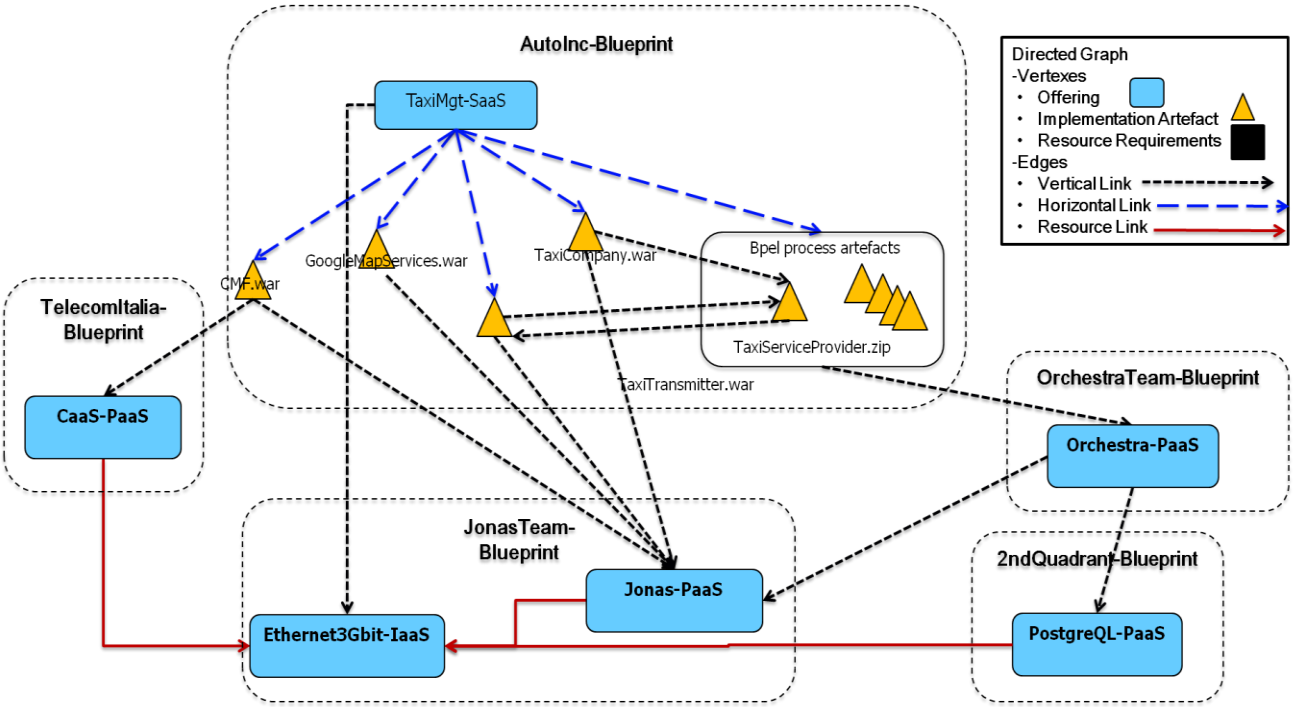
6.3. Phase 3: Target Blueprint Customization

The next step allows the developer to customize the target blueprint through selecting from the alternative configurations provided by the resolution process and complete the target blueprint. We assume in our case study that after resolving the *AutoInc-Blueprint* blueprint, the application developers decide to use the PostgreQL database instead of the MySQL database. Figure 9 shows the VA graph of the selected "resolved" *AutoInc-Blueprint* blueprint, in which all the resource requirements have been fulfilled ("resolved") by the offerings of the source blueprints.

6.4. Phase 4: Target Blueprint Checking & Testing

Once a configuration has been chosen in Phase 3, the target blueprint containing only fully-resolved blueprints is checked and tested. The purpose of this phase is to provide stakeholders with information about the behavior, properties and quality of the cloud application that has been designed, resolved and customized by the application developer from the target blueprint. If, during the testing process, the resolved and customized target blueprint fails to meet the criteria specified for the final cloud application, the developer/builder may choose one of two options: they may (1) return to Phase 1 of the lifecycle and re-design the target blueprint to create a new specification for the application, or (2) return to Phase 2 of the lifecycle to select different source blueprints and re-configure the existing target blueprint using different offerings, perhaps from different providers.

Figure 9. The VA graph of the selected "resolved" *AutoInc-Blueprint*.



When the application developer is satisfied that the resolved and customized target blueprint operates and behaves according to the design, meets the requirements that guided its design and development and that it can be implemented with the same characteristics the developer can choose to (1) store the resolved and customized target blueprint in the marketplace/repository so others may re-use it (possibly within a new applications) and/or (2) deploy the application using the services resolved and selected in the previous phases. However, if the application is not satisfactory and does not meet the requirements of the application developer/builder then they have two further options: (1) if the issues found during the testing process are significant, then the developer can choose to return to Phase 1 to redesign the target blueprint, possibly modifying its structure or behavior to take advantage of a particular capability; or, (2) if the problems are of a lesser magnitude, then the developer can choose to return to Phase 3 to re-select source blueprints and re-configure the original target blueprint with different cloud service offerings.

6.5. Phase 5: Target Blueprint Deployment

After the testing phase, the application is deployed onto the cloud services selected in the customization phase. The deployment is driven by a deployment plan that ensures SLAs for the correct QoS, specified and tested by the developer earlier, are agreed in advance before any of the application components are deployed. If any of the QoS cannot be agreed, the developer is notified and the customization/testing process is repeated until all QoS are satisfied. Once this step has been completed successfully, the developer, e.g., a virtual service operator or provider, deploys or reuses the SaaS, PaaS or IaaS resources required to support the application and the application is available for use.

6.6. Phase 6: Target Blueprint Monitoring

When the application has been deployed, the operational performance of the entire end-to-end process fulfilled by the application must be monitored to ensure it is performing as designed and expected, *i.e.*, is operating within the constraints set out in the agreed SLA for the component services.

In case an SLA constraint is violated, e.g., performance has descended, a notification event should be triggered to inform the application developers to take new correcting actions. These actions could be a redeployment (which leads back to Phase 5), a new target blueprint resolution (back to Phase 2) or a completely new design of the cloud application (back to Phase 1).

In this section we have presented the lifecycle for the SBA development that incorporates iterative phases for design, testing and deployment. The benefit of documenting this process is that it demonstrates the roles, responsibilities and interactions between the developer, cloud service providers and third-parties (e.g., the marketplace or source blueprint repository) and illustrates how the blueprint template is used to specify and ultimately to deploy a running application.

7. Empirical Evaluation and Prototype Implementation

The empirical evaluation for the blueprinting approach aims to understand how well the proposed blueprint template can support the industry companies in describing their cloud service offerings. It started with a collaborative case study design with our industry partners in the 4caaSt project such as Telefonica, TelecomItalia, 2ndQuadrant and SAP, which resulted in the *Taxi Order Process Scenario* introduced in Section 2. Then, we implemented the proposed Blueprint Template in XSD and distributed it to the 4caaSt community to get feedback. We organized a "blueprint training" virtual workshop to explain the concept of blueprint and train our industry partners how to use the XSD blueprint template. As the cloud service offerings identified in the case study are the real industrial offerings of our partners, we requested them to use the provided XSD blueprint template to describe their blueprints in XML and submit them to our blueprint repository. Figure 10 presents a snippet of the instantiated XML document of the "unresolved" target blueprint *AutoInc-Blueprint* from our case study. The VA section in the *AutoInc-Blueprint* blueprint can be visualized by the "unresolved" VA graph on the left side of the Figure. As mentioned in Section 6.2, resolving a target blueprint means to replace the requirement IDs in its VA template section with the IDs of the offerings of the newly retrieved source blueprints. We used XQuery [29] for accessing and modifying data inside the XML document of the *AutoInc-Blueprint* blueprint.

Our evaluation has shown that the proposed blueprint template is capable to capture all the necessary aspects of an industrial cloud service offering and simple enough to be used by our industry partners. Following our guidelines during the blueprint training workshop, a TelecomItalia representative was able to design a blueprint "on-the-fly" for his offering. We experienced just a little amount of emails exchanged for the blueprint template support, which confirms its simplicity of use.

Apart from the evaluation of the blueprint template, we have worked towards a comprehensive blueprint tool for supporting the application developers during the entire SBA development lifecycle introduced in the previous Section 6. Figure 11 depicts the prototype architecture of the blueprint tool. The Blueprint Headquarter (BlueprintHQ) server provides the SOAP API for the following three operations: *getBlueprint(BlueprintID)* to retrieve a blueprint from the repository based on its unique ID, *getBlueprints([property, value]*)* to retrieve several blueprints based on their property values, and *resolveBlueprint(BlueprintID)* to resolve a blueprint by invoking the Blueprint Resolution Engine [30]. This API is consumed by the marketplace component provided by our 4caaSt partners Telefonica and SAP to productize a blueprint and advertise it in a marketplace. The API is also consumed by the BlueprintEXE server that supports a web-based user interface for visualization purpose. Figure 12 illustrates the VA graph visualization of the sample "resolved" *AutoInc-Blueprint* blueprint on the web interface of the BlueprintEXE. All the components of the blueprint prototype, *i.e.*, the BlueprintEXE, BlueprintHQ, blueprint repository and blueprint resolution engine, are currently hosted on a Flexiscale virtual machine [31] (supported by our 4caaSt partner Flexiant).

Figure 10. XML implementation of the sample *AutoInc-Blueprint*.

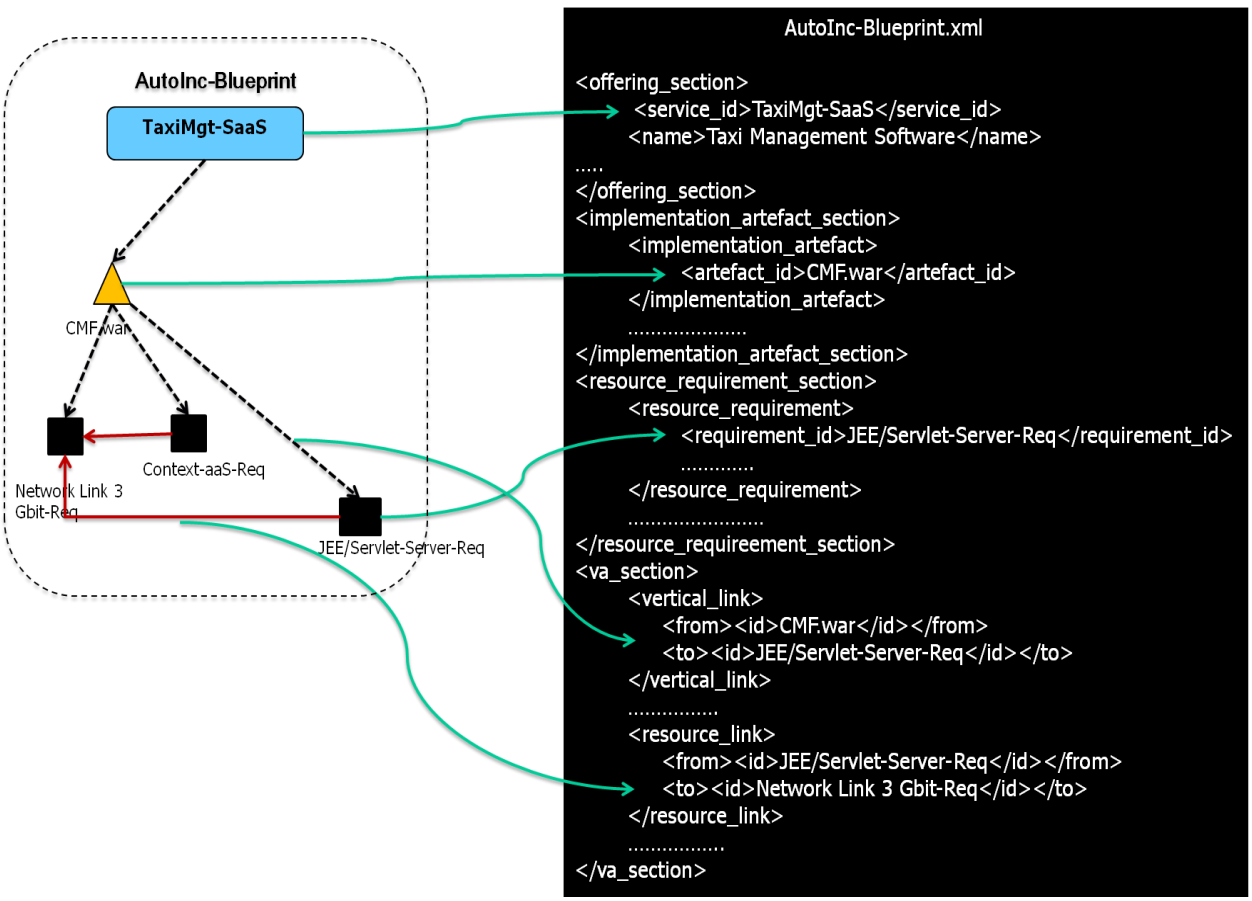


Figure 11. Blueprint Prototype Architecture.

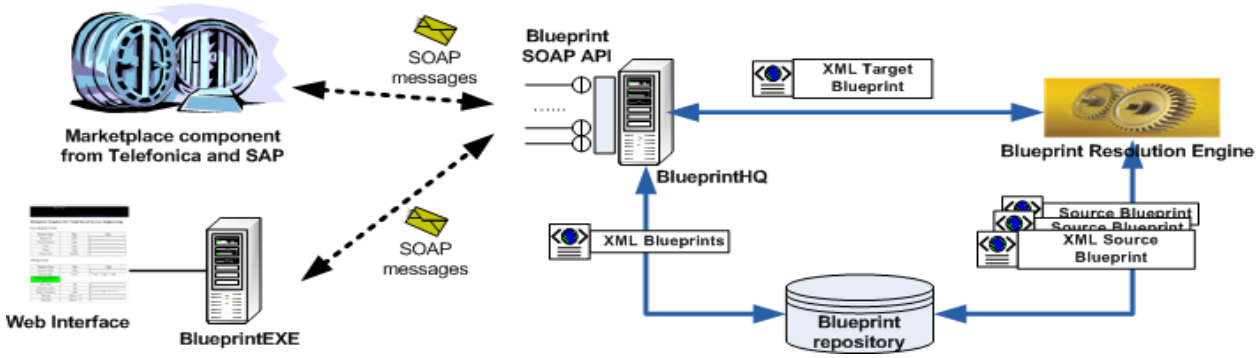
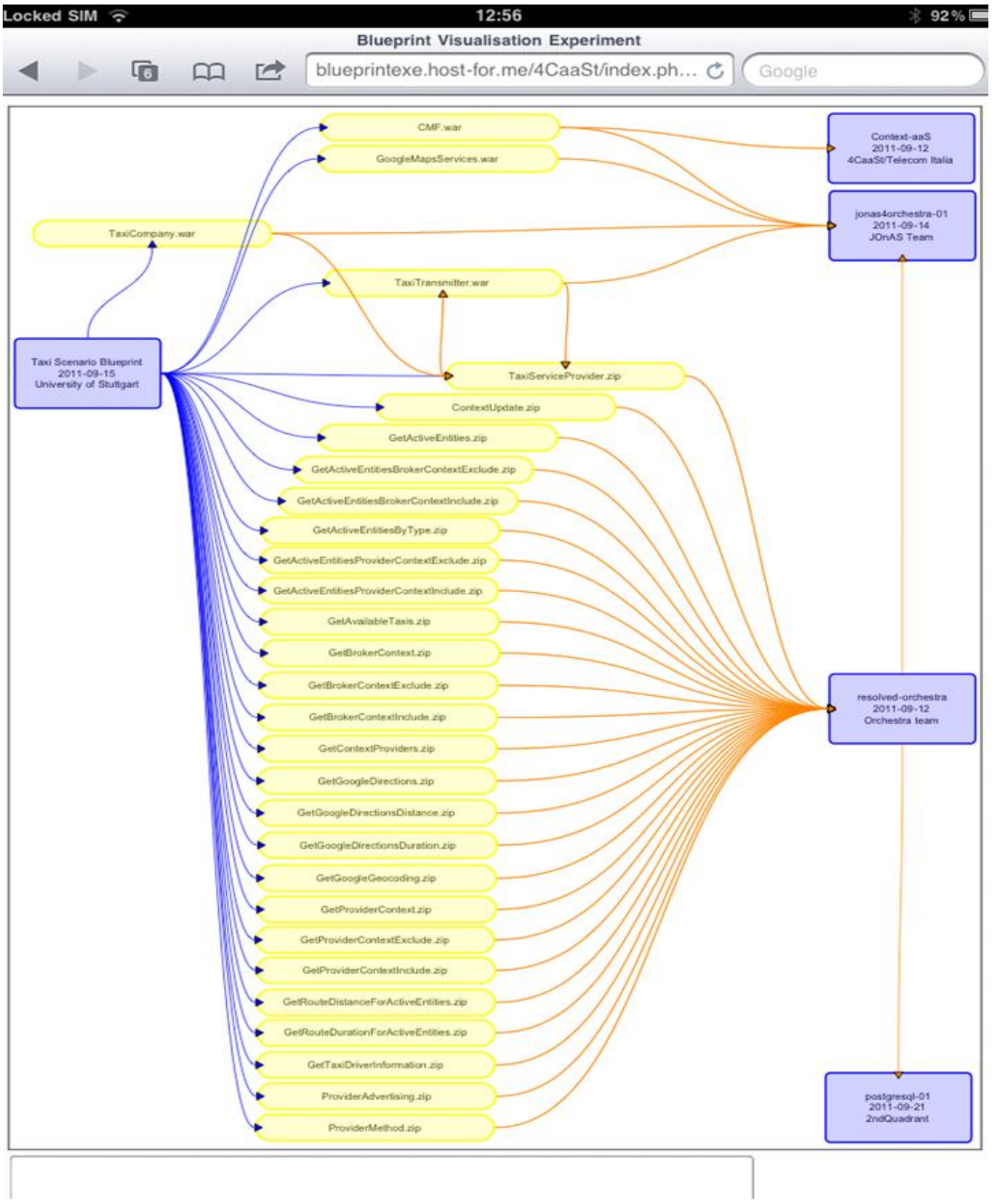


Figure 12. The web interface of our blueprint prototype showing a resolved blueprint.



The blueprint prototype has shown the feasibility of implementing a tool for the blueprinting approach. In particular, we have merely focused on the blueprint support for the first three phases of the SBA development lifecycle described in Section 6, namely the design and reuse of blueprints for SBA development. Additional components can be integrated to the prototype including the deployment controller, monitoring component, adaptation engine, *etc*.

8. Conclusion and Future Work

Cloud blueprinting is a novel approach for engineering Service-based Applications (SBAs). Following this approach, developers can create sophisticated SBAs from applications, platforms and infrastructures offered by different providers in the cloud to achieve end-to-end business requirements. This paper has proposed the *Blueprint* concept as a uniform, abstract description of cross-layer cloud service offerings, a *Blueprint Template* for describing the blueprints, and a *Blueprint Lifecycle* that explains how blueprints are used during all the engineering phases of an SBA.

Blueprint has been adopted as one of the main innovative concepts within the EC's 4caaSt FP7 project. An industry cloud computing case study has been jointly defined by the 4caaSt community and has been used as a running example in this paper to demonstrate our blueprinting approach. The current blueprint XSD template and web-based blueprint prototype is integrated in a joint 4caaSt demonstration. In the future, our vision is to continuously improve the structure of our blueprint template to capture new requirements of SBA development in the cloud. In particular, we will look at other cross-cutting concerns in cloud computing such as security, reliability, pricing, licensing, *etc*., thanks to the extensible design of the blueprint template that allows adding more blueprint properties. More functionality will also be developed for our blueprint prototype towards a comprehensive blueprint tool that supports the entire SBA development lifecycle and targets the effect of changes in the blueprints and their composition.

Acknowledgement

The research leading to this result has received funding from the Dutch Jacquard program on Software Engineering Research via contract 638.001.206 SAPIENSA; and the European Union's Seventh Framework Programme FP7/2007-2013 (4CaaSt) under grant agreement no 258862. The authors would like to thank Mathijs van der Paauw for his work on implementing the prototype.

References

1. Olivier, S.; Prins, J. Scalable Dynamic Load Balancing Using UPC. In *Proceedings of the 37th International Conference on Parallel Processing*, Washington, DC, USA, 8–10 October 2008.
2. Rodero-Merino, L.; Vaquero, L.M.; Gil, V.; Galán, F.; Fontán, J.; Montero, R.S.; Llorente, I.M. From infrastructure delivery to service management in clouds. *Future Gener. Comput. Syst.* **2010**, *26*, 1226–1240.
3. Wu, H.; Kemme, B. A Unified Framework for Load Distribution and Fault-Tolerance of Application Servers. In *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*; Springer-Verlag: Berlin, Germany, 2009; Volume 5704, pp. 178–190.

4. Amazon Web Services. AWS CloudFormation. Available online: http://aws.amazon.com/de/cloudformation (accessed on 19 March 2012).
5. Appirio CloudFactor. Available online: http://www.cloudfactorapp.com/ (accessed on 19 March 2012).
6. Andrikopoulos, V.; Benbernou, S.; Papazoglou, M.P. Managing the Evolution of Service Specifications. In *Proceedings of CAiSE '08 Proceedings of the 20th international conference on Advanced Information Systems Engineering*, Montpellier, France, 18–20 June 2008; pp. 359–374.
7. Dumas, M.; Benatallah, B.; Nezhad, H.R.M. Web service protocols: Compatibility and adaptation. *IEEE Data Eng. Bull.* **2008**, *31*, 40–44.
8. DMTF: Open Virtualization Format (OVF). Available online: http://www.dmtf.org/standards/ovf (accessed on 19 March 2012).
9. Nguyen, D.K.; Lelli, F.; Taher, Y.; Parkin, M.; Papazoglou, M.P.; van den Heuvel, W.J. Blueprint Template Support for Cloud-Based Service Engineering. In *Proceedings of the 4th European Conference ServiceWave'11*, Poznan, Poland, 26–28 October 2011.
10. EC's 7th Framework project 4caaSt. Available online: http://4caast.morfeo-project.org/ (accessed on 19 March 2012)
11. Moltchanov, B.; Fra, C.; Valla, M.; Licciardi, C.A. Context Management Framework and Context Representation for MNO. In *Proceedings of Twenty-Fifth AAAI Conference on Artificial Intelligence*, San Francisco, CA, USA, 7–8 August 2011.
12. Keahey, K.; Tsugawa, M.; Matsunaga, A.; Fortes, J. Sky computing. *IEEE Internet Comput.* **2009**, *13*, 43–51.
13. 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; ACM: New York, NY, USA, 2009; pp. 19:1–19:12.
14. 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*; ACM: New York, NY, USA, 2010; pp. 61–72.
15. Rochwerger, B.; Breitgand, D.; Levy, E.; Galis, A.; Nagin, K.; Llorente, I.M.; Montero, R.; Wolfsthal, Y.; Elmroth, E.; Caceres, J.; Ben-Yehuda, M.; Emmerich, W.; Galan, F. The reservoir model and architecture for open federated cloud computing. *IBM J. Res. Dev.* **2009**, *53*, 4:1–4:11.
16. Konstantinou, A.V.; Eilam, T.; Kalantar, M.; Totok, A.A.; Arnold, W.; Snible, E. An Architecture for Virtual Solution Composition and Deployment in Infrastructure Clouds. In *Proceedings of the 3rd International Workshop on Virtualization Technologies in Distributed Computing*; ACM: New York, NY, USA 2009; pp. 9–18.
17. Chieu, T.; Mohindra, A.; Karve, A.; Segal, A. Solution based deployment of complex application services on a cloud. In *Proceedings of the IEEE International Conference on Service Operations and Logistics and Informatics (SOLI)*, Qingdao, China, 15–17 July 2010.
18. Nimbula. Available online: http://nimbula.com/ (accessed on 19 March 2012).
19. Apache Deltacloud. Available online: http://deltacloud.apache.org/ (accessed on 19 March 2012).

20. La, H.J.; Kim, S.D. A Systematic Process for Developing High Quality saas Cloud Services. In *Proceedings of the 1st International Conference on Cloud Computing*, Beijing, China, 1–4 December 2009; Springer-Verlag: Berlin, Germany, 2009; pp. 278–289.
21. Maximilien, E.M.; Ranabahu, A.; Engehausen, R.; Anderson, L.C. Toward cloud-Agnostic Middlewares. In *Proceeding of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, Orlando, FL, USA, 25–29 October 2009; ACM: New York, NY, USA, 2009; pp. 619–626.
22. Tsai, W.T.; Sun, X.; Balasooriya, J. Service-Oriented Cloud Computing Architecture. In *Proceedings of the Seventh International Conference on Information Technology New Generations*, Las Vegas, NV, USA, 11–13 April 2011; pp. 684–689.
23. Mietzner, R. A method and implementation to define and provision variable composite applications, and its usage in cloud computing. Ph.D. Thesis, Universitaet Stuttgart, Stuttgart, Germany, August 2010.
24. VMware vCenter Orchestrator. Available online: http://www.vmware.com/products/vcenter-orchestrator/overview.html (accessed on 19 March 2012).
25. SAP service marketplace. Available online: http://www.sap.com/services/more/servsuptech/service-marketplace.epx (accessed on 19 March 2012).
26. Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I.; Zaharia, M. A view of cloud computing. *Commun. ACM* **2010**, *53*, 50–58.
27. Oaks, P.; Edmond, D.; ter Hofstede, A. Capabilities: Describing What Services Can Do. In *Proceedings of the First International Conference on Service-Oriented Computing, ICSOC 2003*, Trento, Italy, 15–18 December 2003; pp. 1–16.
28. Verma, K.; Akkiraju, R.; Goodwin, R. Semantic Matching of Web Service Policies. In *Proceedings of the Second Workshop on SDWP*, Orlando, FI, USA, July 2005.
29. Xquery W3C Recommendation. Available online: http://www.w3.org/TR/xquery/ (accessed on 19 March 2012).
30. The Blueprint Headquarter (BlueprintHQ) prototype server. Available online: http://blueprintexe.host-for.me/4CaaSt/blueprintHQ.htm# (accessed on 19 March 2012).
31. Flexiscale Virtual Machines. Available online: http://www.flexiscale.com/ (accessed on 19 March 2012).