No. 2011-067

FOUNDATIONS OF BLUEPRINT FOR CLOUD-BASED SERVICE ENGINEERING

By Dinh Khoa Nguyen

June 2011

ISSN 0924-7815

Foundations of Blueprint for Cloud-based Service Engineering*

Dinh Khoa Nguyen

ERISS Technical Report- CentER Discussion Paper
v0.1
JEL-Code : M15 - IT Management

European Research Institute in Service Science (ERISS)
Tilburg University, The Netherlands
D.K.Nguyen@tilburguniversity.edu

Abstract. Current cloud-based service offerings are often provided as one-size-fits-all solution and give little or no room for customization. This limits the ability for application developers to pick and choose offerings from multiple software, platform and infrastructure service providers and configure them dynamically and in an optimal fashion to address their application requirements. Furthermore, combining different independent cloud-based services necessitates a uniform description format that facilitates their design, customization, and composition. Hence, there is a need to break down the monolithic offerings into loosely-coupled cloud services offered by multiple providers that can be flexibly customized and (re-)composed in different settings. We propose in this paper the *Blueprint* concept - a uniform abstract description for cloud service offerings that may cross different cloud computing layers, i.e. software, platform and infrastructure. Using the proposed Blueprint Template for engineering cloud service offerings will solve these shortcomings and subsequently lower the barrier to entry for cloud computing.

Keywords. Cloud Computing, Cloud-based Service Engineering, Blueprint Template, Service-oriented Computing, Service-oriented Architecture.

1 Introduction

Recently, the field of *cloud computing*, where computational, infrastructure and data resources are available on-demand from a remote source, has become hugely

* The research leading to this result has received funding from the Dutch Joint Academic and Commercial Quality Research and Development (Jacquard) program on Software Engineering Research via contract 638.001.206 SAPIENSA: ServiceenAbling PreexIsting ENterpriSe Assets; and the European Union's Seventh Framework Programme FP7/2007-2013 (4CaaSt) under grant agreement n 258862.

popular. One of the reasons for its popularity is because cloud computing gives the option to outsource the operation and maintenance of IT tasks, allowing organizations and their employees to concentrate on their core competencies. This, together with pay-as-you-go billing that reduces the need for capital expenditure on equipment, means that with cloud computing software-services can be designed and tailored to a business's individual requirements.

The US National Institute of Standards and Technology (NIST) defines three delivery models for services in the cloud, or cloud services [1]: 1) Software-as-a-Service (SaaS), or the provision of opaque software applications over-the-network, running on a cloud infrastructure; 2) Platform-as-a-Service (PaaS), which provides an environment to deploy consumer or third-party applications to the cloud; and 3) Infrastructure-as-a-Service (IaaS), which allows a consumer access to fundamental computing resources, such as computational processing, storage and network, and from which they can deploy and run arbitrary operating systems, middleware and applications. We use the term *XaaS* throughout this report to encompass these three delivery models for cloud services.

1.1 Research Problems in Service-based Applications (SBA)

By combining different, independent networked XaaS offerings from one or more providers we can compose Service-Based Applications (SBAs) to perform a desired end-to-end function. Note that SBAs have a profound difference with respect to component-based applications; whilst the owner of the component-based application also owns and controls its components, the owner of a SBA does not, in general, own the XaaS components nor it can control their execution [2]. XaaSs and SBAs are ideally matched since the flexibility of cloud computing provides the fabric through which SBAs can be constructed and deployed.

However, despite these advantages, there are a two main issues that need to be considered carefully when migrating (parts of) an SBA to 'the cloud'. The first problem concerns the issue of multi-tenancy of the XaaSs used to compose a SBA; current XaaSs are often provided as a one-size-fits-all solution and give little or no room for further customization. As an example, the customization of the cloud platform and infrastructure through which SaaS applications are provided is not possible, and monolithic SaaS offerings are likely to be ineffective in meeting the business requirements of several consumers. For instance, if the platform performance of a given SaaS application runs down, the only possible solution to deal with this matter is to replace the entire application by another one with a better platform performance. This is because the application is a monolithic block which does not allow dealing with only the disturbed platform as an independent element of the service stack. Therefore, there is clearly a need to break down the monolithic SaaS stack and provide a more effective and flexible method for SBA designers to select, customize and aggregate XaaS services, i.e. software, platform and infrastructure services, offered by several XaaS providers, as illustrated in Figure 1. Secondly, creating SBAs in the cloud and integrating them with other XaaS offerings presents further problems. For example, when designing, deploying and operating an SBA across several XaaS

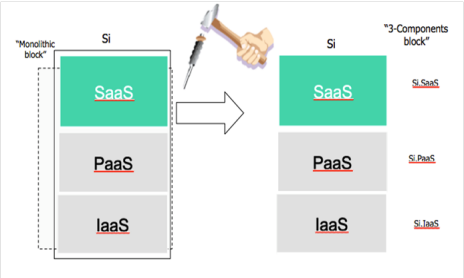

Fig. 1. Breaking the monolithic SaaS Block

providers, difficulties can arise due to the inconsistency of cloud interfaces and the fact that proprietary technologies are an entry barrier, especially to SMEs, due to the lack of IT staff that can be dedicated to cloud computing development and operations. Consequently, cloud computing remains largely within the domain of established players.

1.2 Requirements

To allow the flexible design and deployment of customized SBAs in a timely manner, we envision a common foundation to address the way XaaSs are designed, engineered and provided though the cloud. Our work revolves around the concept Blueprint - an abstract description of XaaS offerings that may cross different cloud computing abstraction layers, i.e. SaaS, PaaS and IaaS. We believe that the barrier to entry for cloud computing can be lowered and SME-empowered through a common standardized *Blueprint Language* for describing and manipulating XaaSs that facilitate the composition of SBAs. The blueprint language should include the following components

- *C1:* A well-defined Blueprint Template for cloud computing that provides a means for XaaS providers to abstractly (i.e., independent of implementation) and unambiguously describe their XaaS offerings on multiple abstraction layers, i.e., SaaS, PaaS and IaaS offerings.
- *C2:* A Blueprint Algebra to allow the XaaS integrator to flexibly customize, reconfigure or aggregate the selected abstract XaaS offerings on multiple abstraction layers; e.g., in case of migrating an existing SaaS offering to another PaaS offering available in the market due to performance or cost reduction issue.
- *C3:* A Blueprint Query Language to allow the entity composing XaaS offerings into an SBA to search and select abstract XaaS offerings from a repository, such as a service marketplace in [3].

The motivation for developing a blueprint template, blueprint query language and blueprint algebra for XaaSs comes from our participation in the European Commission's (EC's) 4CaaSt project [4], which has the goal of creating an advanced cloud platform that supports the optimized and flexible hosting

of Internet-scale multi-tier applications. The 4CaaSt platform will contain the features necessary to facilitate the programming of rich applications and enable the creation of a true business ecosystem where applications, platforms and infrastructure from different providers can be traded, customized and combined.

1.3 Contributions

This work targets the first requirement C1 of the Blueprint Language and defines the formal template structure required for our vision to allow the abstract descriptions of XaaS offerings. This template is called the XaaS Blueprint Template and provides a common structure, syntax and semantics for describing an XaaS offering that may cross cloud-computing abstraction layers, i.e. SaaS, PaaS and IaaS. Such a template allows for XaaSs to be designed in a uniform way and hence to be composed across several XaaS providers. As an example, a SaaS application could be provided by a provider A, whilst its PaaS layer is provided by another provider B and its IaaS layer by yet another provider C. This will give more flexibility to deal with each layer of a given SaaS application as an independent element of the cloud stack. That means if the platform performance provided by the provider B doesn't meet the user requirement, it will be possible for provider A to search for another PaaS which better fits the user's needs. Migrating this SaaS to a new PaaS will not affect the SaaS application functionality, yet the SaaS is now running on top of another PaaS.[1]

1.4 Structure of the report

The rest of this paper is structured as follows: Section 2 presents a scenario used as an example for deriving a set of requirements for the Blueprint as a uniform representation for XaaS offerings. In Section 2.3, we revise the existing related works and show that they fail to meet the blueprint requirements. Then, Section 3 leads us to the formal definition and structure of the Blueprint concept, as well as a lifecycle for blueprint providers to create, modify, and compose blueprints. Our proposed template in the next Section 4 presents a significant data structure for describing the blueprint. Finally, Section 5 draws some conclusions and future issues.

2 Motivating Scenario

This section presents an enterprise computing scenario developed for the EC's 4CaaSt project [4]. As the scenario in Section 2.1 shows, it requires a uniform description for XaaS offerings to exist so that the design and configuration, deployment of a cloud-based SBA can occur. We point out these requirements afterwards in Section 2.2

[1] Contributions will be continuously updated.

2.1 Scenario

The scenario contains a number of information entities and we refer to a particular information entity using a Universally Unique IDentifier (UUID). As can be seen later in Section 4, we propose the use of blueprint as the uniform description for XaaS Offering and the UUIDs will be used for representing particular information entities in our sample blueprints. The motivating scenario in Figure 2 contains four actors, each of which is now described in the following.

CE1 and **CE2** are two providers of an open source *composition engine*. The offering of CE1 (uuid=CE1-PaaS) is already a complete PaaS, i.e. the engine is hosted on an in-house JEE platform and Linux machine, and connected to the outside through a 3Gbit Ethernet link. The offering of CE2 (uuid=CE2-PaaS) has similar configuration, except they use Windows machines and their Ethernet network is only 2Gbit. However, both these offerings are monolithic, since they are already preconfigured and no customization of the underlying resources is allowed for the consumers.

The telecom service provider, **Tele1**, provides a basic *SMS Delivery SaaS* (uuid=SMS-Delivery-SaaS) through two alternative configurations. They provides a *binary artefact* written in Java (uuid=art-sms-delivery) and requires a *composition engine* for the deployment. Tele1 has set up business with both CE1 and CE2 by signing a contract to use both the *CE1-PaaS* and *CE2-PaaS*. Their system integrators are responsible for deploying the *art-sms-delivery* on one of these two external PaaS, depending on the customer selection.

AutoInc is an established SME and has spotted a business opportunity providing fleet vehicle management in the Netherlands. They plan to deploy their business functions as a *Vehicle Management (VM) SaaS* (uuid=VM-SaaS), since this provides ubiquitous and common access for their prospective customers, e.g., logistics companies, car-hiring providers and businesses with mobile engineers, such as utility providers. AutoInc's idea is to reduce their customer's transport bills through intelligent fleet management which optimizes the allocation of work to vehicles, crews and depots via GPS tracking and 'just-in-time' route planning. They need a solution to capture these business ideas as SaaS functionalities offered to the prospective customers. Furthermore, in order to implement the ideas, AutoInc has contracted a software consultancy who is writing the vehicle management software in the Java programming language. The software requires a *JEE application server* (uuid=AutoInc-Req01) including a *Servlet v2.5 container* (uuid=AutoInc-Req02) for the web interface, and 2 instances of *MySQL database* (uuid= AutoInc-Req03) for recording the vehicle's location together with its characteristics, such as to which company it belongs to, its type and capacity. The software hence contains 3 main artefacts: the *application core binary* (uuid= AutoInc-Art01) as a jar file and a *configuration file* (uuid=AutoInc-Art02) to be deployed on the prospective JEE application server, and a *configuration and setup file* (uuid=AutoInc-Art03) for the prospective MySQL database.

In order to reach some predefined KPIs, AutoInc has also some extra non-functional requirements for the prospective platform resource requirements. The JEE application server should ensure the *throughput* $\geq$ *100 req/s* and the *re-*

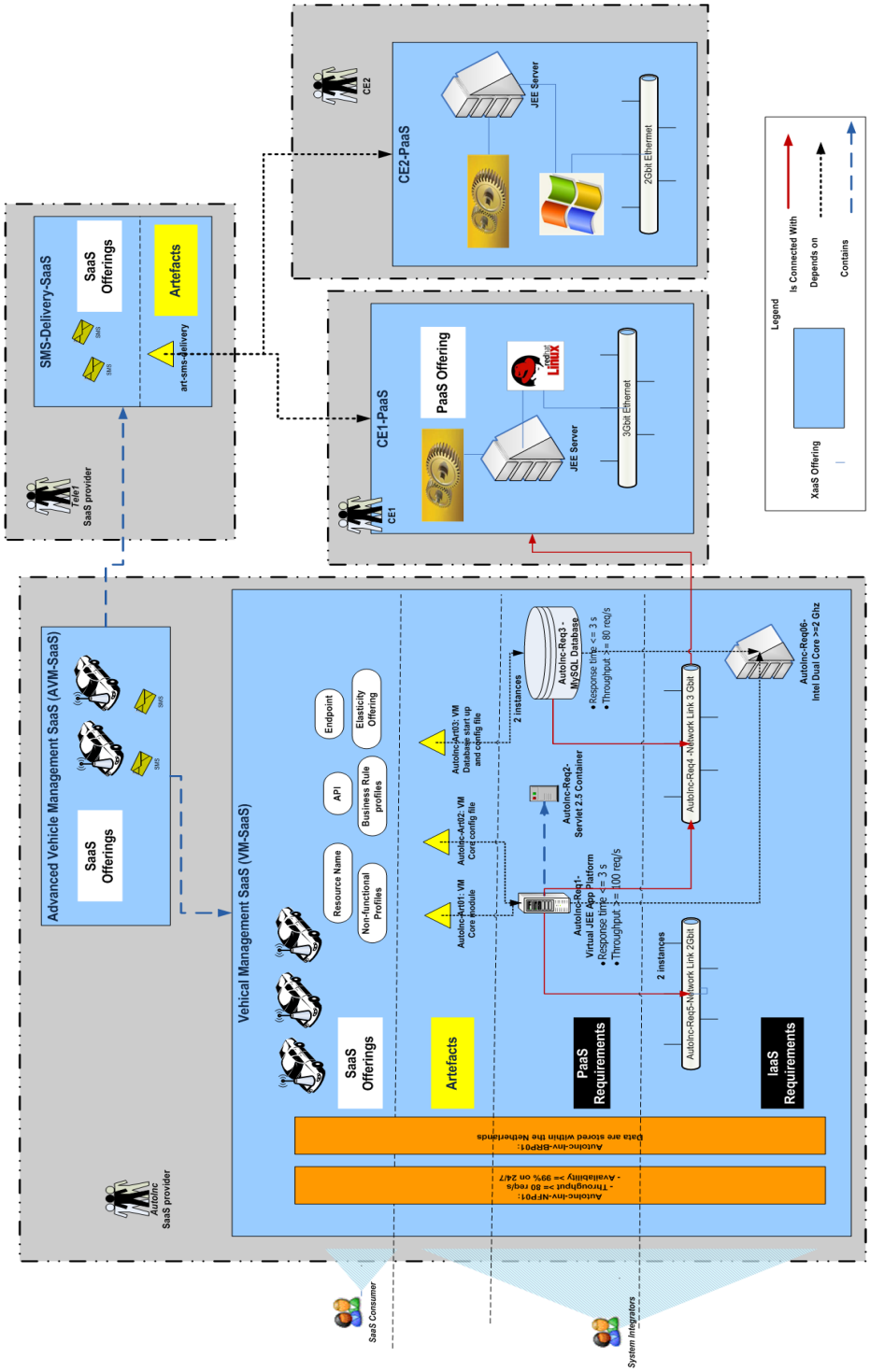

Fig. 2. XaaS Offerings Scenario

sponse time ≤ 3s. For the prospective MySQL DB, throughput requirement is relaxed to ≥ 80req/s while the response time requirement remains the same.

Based on foreseen performance requirements, AutoInc has derived several infrastructure requirements for the machine hosting the platform resources and the connecting network links. It requires the prospective JEE Application server to be connected to the outside through 2 lines of *2Gbit network link* (uuid=AutoInc-Req05), and with the MySQL database through a *3Gbit network link* (uuid=AutoInc-Req04). Both the JEE Application Server and the MySQL DB must be deployed on *Intel Dual core machine with the processor ≥ 2Ghz* (uuid=AutoInc-Req06).

AutoInc also has some invariant constraints for the whole system prescribing that the whole system must have *throughput ≥ 80 req/s and availability ≥ 99% on 24/7* (specified in the Non-functional Profile for Invariants, uuid=AutoInc-Inv-NFP01), and *all the data must be stored only within the Netherlands* (specified in the Business Rules Profile for Invariants, uuid=AutoInc-Inv-BRP01).

Furthermore, in the future AutoInc foresees some of their prospective customers will prefer an SMS service for dispatching new work to vehicles, and decide to include this feature in their new *Advanced Vehicle Management (AVM) SaaS* offering (uuid=AVM-SaaS). They cooperate with the telecom company Tele1 for acquiring its *SMS-Delivery-SaaS*. Due to some performance and economic reason, AutoInc chooses the first configuration offered by Tele1, in which the underlying resources, i.e. composition engine, JEE Server, Linux machine and 3Gbit Ethernet, are actually provided by another provider, the CE1 PaaS provider. As AutoInc also wants their *VM-SaaS* to interact with the *SMS-Delivery-SaaS*, a link is required between the *AutoInc-Req04* and the *CE1-PaaS*.

Requiring a number of platform and infrastructure resources, and including the external SMS Delivery SaaS of Tele1 means that AutoInc has designed their VM-SaaS and AVM-SaaS offerings as Service-Based Applications (SBA), since the required platform and infrastructure resources and the SMS delivery service are not under their direct control. This is the distinguishing characteristic of an SBA from a component-based application. However, in order to enable further developments and deployment of this SBA, AutoInc also needs to specify their architectural requirements through a **Virtual Architecture Topology (VAT)**. This VAT specifies which artefacts should run on which resources and how the resources are connected to or deployed on each other. Figure 2 visualizes both the resource requirements and the VATs.

2.2 Derived Requirements for the Blueprint

Taking into account the scenario presented in the previous Section **??**, we derive first the functional requirements for the Blueprint as an abstract uniform representation for XaaS offerings. On the one hand, the Blueprint should enable AutoInc, Tele1, CE1, and CE2 to describe their XaaS offerings, and publish them in a marketplace so that the consumers can discover and use them. On the other hand, it also needs to provide the system integrators with information about the required XaaS resources for deploying the XaaS offerings.

FR1: The Blueprint should be able to describe all types of XaaS offerings. A blueprint provider can use blueprints to specify his offerings of all three

types SaaS, PaaS and IaaSs. As an example, AutoInc is a SaaS blueprint provider whilst CE1 and CE2 are two PaaS blueprint providers.

FR2: The Blueprint should contain the description of the functional capabilities of a XaaS offering, e.g. a (Advanced) Vehicle Management Software, a SMS delivery software, or a composition engine. This will enable the searching and discovery of suitable XaaS offerings in the marketplace.

FR3: The Blueprint should contain the technical specification of a XaaS offering. This includes the technical interfaces for interacting with the XaaS, the endpoint of the XaaS, and the elasticity offering, i.e. how many instances of the XaaSs can be offered to a single consumer.

FR4: The Blueprint should contain also the descriptions of non-functional properties of the XaaS offering, as well as all the business rules applied on it. Some examples of the non-functional properties are the QoS properties, security levels, pricing schemes, etc. For instance in our scenario, the VM-SaaS offering of AutoInc is ensured to response $\leq$ 5s and costs 10000 euros for an offer with less than 500 vehicles. Examples of the business rules are regulatory compliance rules or some legal issues, e.g. in our scenario the VM-SaaS offering is delivered within only the Netherlands and all the Dutch tax rules and legal issues must be applied.

FR5: The Blueprint should allow to specify the implementation artefacts of an XaaS offerings, in particular which artefacts are provided and where to get them. Examples of artefacts of AutoInc are the software binary *AutoInc-Art01* and configuration and setup files *AutoInc-Art02* and *AutoInc-Art03*.

FR6: The Blueprint should allow to specify the underlying platform and infrastructure resource requirements for deploying the artefacts, including

- The functional resource requirements, e.g.*AutoInc-Req01* (a JEE Application Server), *AutoInc-Req02* (a Servlet 2.5 Container), *AutoInc-Req06* (an Intel Dual Core machine $\geq$ 2Ghz) etc.
- Non-functional requirements of the required resources, if any, e.g. *AutoInc-Req01 with throughput $\geq$ 100req/s and response time $\leq$ 3s.*
- Rule-based requirement of the required resources, if any.

FR7: The Blueprint should allow to specify the architectural requirements through a Virtual Architecture Topology (VAT), which contains information about how the needed resources are organized and which artefacts are supposed to be deployed on which prospective resources. For instance in our scenario, *AutoInc-Req01* should contain *AutoInc-Req02* and run on *AutoInc-Req06*, or the *AutoInc-Req04* should be linked to the external *CompositionEngine-PaaS* offering of CE3.

FR7: Alongside the above information, the Blueprint should also allow all the providers to prescribe some invariant constraints for both their service consumers and system integrators. These invariant constraints include the functional, non-functional and rule-based constraints that are applied on all the offerings, artefacts, resource requirements, and the VAT, e.g. AutoInc has prescibed some following invariants: *Key value based storage must be used, throughput of the system must be $\geq$ 80 req/s, Availability must be $\geq$ 99% on 24/7, and all data storage must be within the Netherlands.*

FR7: The BT should avoid introducing 'accidental complexity' and provide a solution that contains as little non-essential information as possible.

XaaSs are subject to change, since evolutions according with customer requests are part of everyday life. In addition, current economic and technical conditions may as well influence the evolution of a service. The description of a service contained in a blueprint should reflect this need. What follows are a set of non-functional requirements that should guide the design of a blueprint.

NFR1: Configurability and customizability: The Blueprint should allow the description of an XaaS offering that is inherently configurable and customizable.
NFR2: Modularity: The Blueprint should be modular and extensible as an XaaS offering can be the result of a composition of several existing XaaS offerings.
NFR3: Evolvability: XaaSs are subject to changes of infrastructure, market and service offering. Therefore the Blueprint should be extensible in order to capture new arising properties or characteristics of an XaaS offering.

Following the derived requirements for the Blueprint, we will demonstrate in the next Section 2.3 how existing approaches lack the ability to fulfill these requirements and subsequently in Section 3, how these gaps can be filled by our Blueprint concept.

2.3 Evaluation of Related Work

The motivating scenario described above indicates the need for a uniform representation that can accurately describe the capabilities and resource requirements of cloud computing infrastructure, platforms and software services. The benefit of such a representation is that it not only provides a uniform description for XaaS offerings across vendors, but also serves as a resource requirement specification necessary for deploying and provisioning those offerings. Furthermore, it can be used with operators and constraints to provide a mechanism for querying, customizing and aggregating those offerings.

However, cloud computing is a relatively new research area and only a few existing works exist that partly target our research objective. We first review some related works addressing the lack of standardization for describing XaaSs that results in the well-known vendor lock-in problem. The work in [5] addresses the vendor lock-in problem that prevents the interchangeability and interoperability between cloud services and presented a state-of-the-art in both standardization efforts and on-going projects. Similarly, the ability to manipulate, integrate and customize XaaS descriptions across different cloud providers has also been recognized in [6], which has IaaS, application and deployment orchestrators, but falls short of proposing a solution for the problem at hand.

Much recent work on standardizing the description of XaaSs concentrates mostly on the infrastructure level, and does not cover the complete cloud computing picture. As an example, the DMTF has exhibited proven standards, such as the Open Virtualization Format (OVF) [7] that provides open packaging and distribution formats for virtual machines, and the Virtualization Management

(VMAN) [8] specifications that address the management lifecycle of a virtual environment to help promote interoperable cloud computing services. The work in [9] uses the OVF to define a service definition language for deploying complex Internet applications in federated IaaS clouds. These applications consist of a collection of virtual machines (VM) with several configuration parameters (e.g., hostnames, IP and other application specific parameters) for software components (e.g., web server, application server, database, operating system) included in the VMs. Similarly, the approach in [10] targets the interoperability between the federated clouds by providing a collection of proposals for 'Intercloud' protocols and formats. However, this work is still in the early stage and targets only interoperability between data centers, i.e. only on the infrastructure level.

Approaching the standardization for XaaSs from the different perspective, the Model Driven Engineering (MDE) research community has realized the benefit of combining MDE techniques with SaaS development and suggested combining MDE with cloud computing [11]. As the article describes, there is no consensus on the models, languages, model transformations and software processes for the model-driven development of cloud-based SaaSs. Following the MDE vision, [12] proposes a meta-model that allows cloud users to design applications independent of any platform and build inexpensive elastic applications. From their point of view, a cloud application is a software provided as a service that should avoid the vendor lock-in problem concerning the underlying platforms. This meta-model however describes only the capabilities and technical interfaces of the cloud application service. Similarly, [13] presents a different customer-centric cloud service model. This model concentrates on aspects such as the customer subscription, capability, billing, etc., yet does not cover other technical aspects of the cloud services including the technical interfaces of the cloud services, the elasticity, the required deployment environment, etc. Other existing models, e.g. [14], also lack a formal structure and definitions (reducing their usability and reusability) or are not explicit and assume tacit knowledge.

Apart from the need to have a uniform description for XaaS offerings to avoid vendor lock-in, another requirement is to specify in the description the resource requirements and constraints necessary for the deployment of XaaS offerings, as shown in the scenario in Section 2. The Cafe application and component templates in [15] are maybe the most relevant approaches that provide the customization through an orthogonal variability model. However they still lack non-functional and constraint descriptions for XaaSs. In practice, an attempt to provide template for utilizing cloud services is available from Amazon through their AWS CloudFormation product [16]. This template provides AWS developers with the ability 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.

In summary, existing works mostly aim to propose standards for only certain aspects of cloud computing and thus fail to cover the full picture of a XaaS offering, 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 XaaS developers during the various engineering phases of a XaaS offering, i.e. the customization, deployment, provisioning, etc. We propose the *Blueprint* concept in the next section as a uniform representation to capture the comprehensive knowledge of an XaaS offering to support developers during the various engineering phases.

3 Blueprint: Concept Definition, Data Structures and Lifecycle

Taking into account the lack of a common standardized description of XaaS offerings that allows also flexible manipulation, composition and customization, our work in this section revolves around the concept *Blueprint* as our proposed solution. Section 3.1 first explores some intuitive ideas about the Blueprint and is then followed by our proposed structure of the Blueprint in Section 3.2. A lifecycle of designing, manipulating, and composing blueprints is introduced in Section 3.3 from the blueprint provider's perspective.

3.1 Blueprint Definition

We define a Blueprint as a uniform abstract description of an XaaS offering. In general, existing XaaS offerings fall into one of these types: complete and monolithic, complete and customizable, or incomplete. We distinguish the three types of blueprints available in the marketplace depending on the types of their XaaS offerings

Complete and monolithic blueprints A blueprint is complete if its XaaS offering is already deployed and ready to be consumed. It is monolithic if the offering is a predefined and pre-configured cloud stack that cannot be customized for a certain type of consumers. For instance in our scenario in Section 2, the blueprints used to describe the *CE1-PaaS* and *CE2-PaaS* offerings fall under this category. Although monolithic XaaS offerings have very low flexibility for XaaS consumers, they are unfortunately still being widely adopted by the current cloud computing providers in the market. As an example, although the CRM-related SaaSs of SalesForce provide some customizations for the offered functionalities, they are still considered as monolithic SaaS offerings since no customizations of non-functional properties, rules or the underlying resources are allowed.

Figure 3 presents an intuitive view on the complete and monolithic blueprints. For SaaS blueprints, the concrete implementation is the SaaS artefacts deployed on a given platform that runs on a given infrastructure, whilst for PaaS blueprints, it is the PaaS artefacts that can be deployed on a provided platform. Hiding the concrete implementation behind, a XaaS provider can use the blueprint to offer his resources in 3 different ways:

- As a complete monolithic SaaS

– As a complete monolithic PaaS that can host a given set of software applications.
– As an IaaS that can runs a given set of platforms

In principle each different way refers to a different product that the XaaS provider can sell. The blueprints are used by the consumers in order to have a technical and non-technical understanding of the various XaaS offerings. Therefore a customer can decide to purchase different items according with his particular needs.

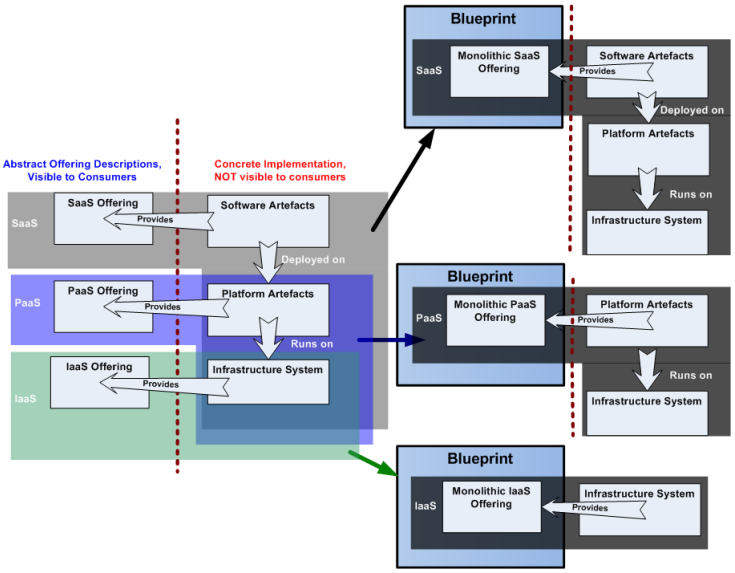

Fig. 3. Complete and Monolithic Blueprints

Complete and customizable blueprints Consumers also desire complete XaaS offerings (SaaSs, PaaSs, or IaaSs) with customization capabilities. Customizations may be triggered by the consumers (e.g., through a contract) or by the provider, according to its internal strategies. While the customizations of functionalities do not affect much the offering configuration, those of the offered non-functional properties, policies and rule constraints will most probably lead to customizations of the underlying resource configurations. As an example in our scenario in Section 2, Tele1 offers their *SMS-Delivery-SaaS* blueprint with two different configurations of the underlying resources probably regarding different performance levels or policies. According to the customization request of AutoInc, the artefact *art-sms-delivery* of the *SMS-delivery-SaaS* blueprint needs to be deployed on the *CE1-PaaS* blueprint of CE1.

As pointed out in the introduction in section 1 our approach is to decompose the monolithic XaaS offering in three different layers (SaaS, PaaS, IaaS) and use the blueprints to describe each of these layered resource offerings. The advantage is that these blueprints can be flexibly re-assembled in different configurations. Hence, there could exist different customizations for a blueprint in terms of different combinations of its underlying dependent blueprints. Figure 4 illustrates this idea of customizable blueprints. In this figure, the Blueprint S is a complete SaaS offering with the Blueprint P1 as its default platform configuration. However, the provider of S still can switch to another platform offered by the Blueprint P2, in case some customization requirements force him to do so. Similarly, P1 is a PaaS blueprint with the Blueprint I1 as its default configuration, but the provider of P1 totally can switch to the infrastructure resources provided by the I2 blueprint. The blueprint provider should keep track of the dependen-

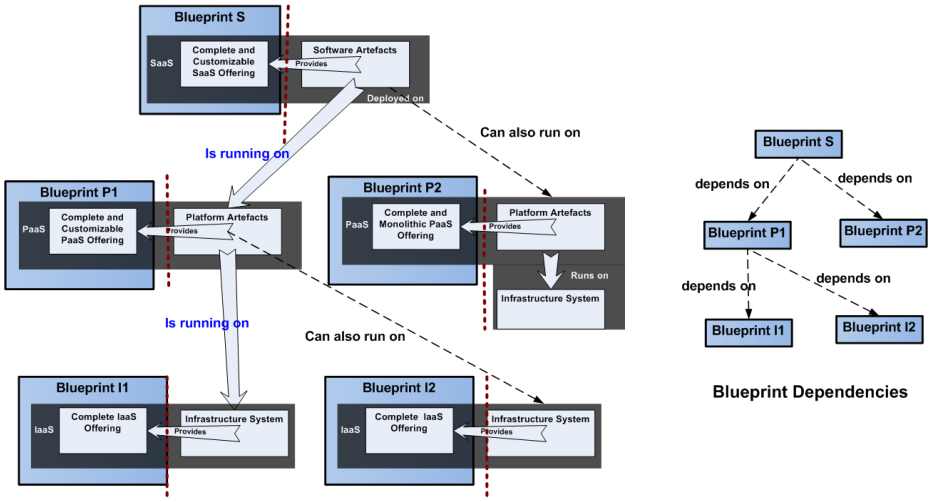

Fig. 4. Complete and Customizable Blueprints

cies among all of his blueprints for configuration management purposes, which will enable different customizations for a certain blueprint. As an example, the right side of Figure 4 shows a simple representation of the blueprint dependencies between S, P1, P2, I1 and I2. The Blueprint Dependencies (BD) graph is an important data structure aiming to capture these dependencies, and will be introduced in the next section **??**. Given all the blueprint dependencies available in the BD graph, customization of a blueprint, e.g. the blueprint S, is conducted by simply selecting a path in the BD graph.

Incomplete blueprints The XaaS offerings captured in this type of are incomplete, i.e. they are not yet ready to be consumed and still need the underlying

resources for the deployment. An example of incomplete blueprints is the *VM-SaaS* blueprint of AutoInc in our scenario in Section 2. The XaaS provider in this case needs to specify in the blueprint the resource requirements, as well as the architectural requirements in terms of a Virtual Architecture Topology (VAT) that indicates how the required resources should be organized. He relies on a system integrator who is responsible for making the blueprint complete. Based on the resource requirements and the VAT, the task of making a blueprint complete includes searching for available blueprints in the marketplace that offer the needed XaaS offerings to fulfill the stated resource requirements, and then integrating all these blueprints together. Incomplete blueprints are inherently customizable since the consumers will also be given the ability to customize the their configurations after they have been made complete. Figure 5 visualizes the incomplete blueprints in part (c) on the top right side, in which the implementation artefacts, resource requirements, and the architectural requirements (in terms of the VAT) are now parts of the blueprint. Next section proposes the

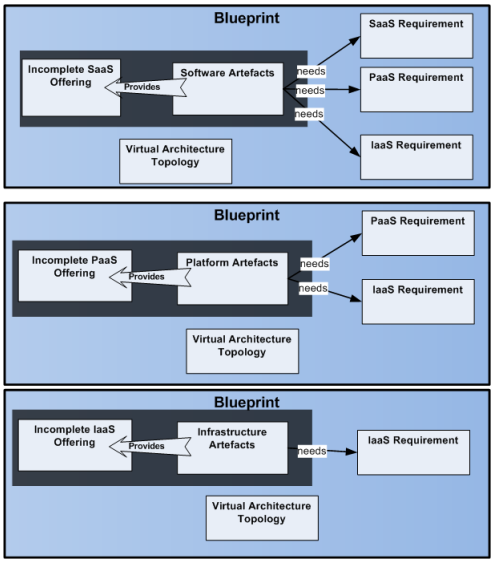

Fig. 5. Blueprints that describe configurable XaaS offerings

common structure for the Blueprint that can capture all the information needed for all three types of blueprints

3.2 Fundamental Structure of a Blueprint

We propose the following fundamental structure for the Blueprint, as illustrated in Figure 6:

– Basic properties: Some basic properties concerning the unique ID of the blueprint, the ownership, the release date, version info, etc.

– Offering: The description of the functional and non-functional offerings containing the functional capabilities, the technical specification, elasticity offering, non-functional properties and the business rules that constrain the offering.
– Implementation Artefacts: Description of the artefacts that implement the Offering
– Resource Requirements: specifying the required XaaS resources for this offering including the non-functional requirements.
– Architectural Requirements: specifying the VAT that expresses how the required resources should be organized and which artefacts require which resources, etc.
– Invariants: constraints that must not be violated by all elements (the offering, artefacts, requirements, VAT) of the blueprints.

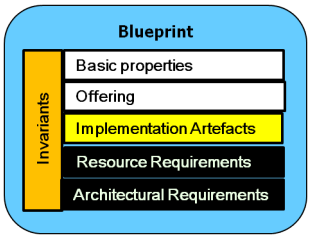

Fig. 6. The Blueprint Fundamental Structure

Complete blueprints can be described without information about the implementation artefacts and resource and architectural requirements, as they aim to provide only abstract information about how to consume their the offerings. Incomplete blueprints, on the other hand, contain also information about the artefacts and the resource and architectural requirements. These requirements drive the search for further blueprints in the marketplace, and lead the way for a provider to compose multiple blueprints in order to construct a customizable offering for customers.

3.3 The Blueprint Lifecycle

The Blueprint Lifecycle is visualized in Figure 7

– Lifecycle Phase 1: The lifecycle starts when a blueprint provider wants to design a new Target Blueprint using some kinds of Blueprint Template capturing all necessary information for describing his XaaS resource. The result of the design is a complete or incomplete Target Blueprint. If the Target Blueprint is complete, it comes together with a resolved VAT graph showing the dependencies with other available Blueprints.

– Lifecycle Phase 2: This phase is carried out for incomplete blueprints only. Apart from the Target Blueprint, the Blueprint Provider can get access to a number of Source Blueprints available in the Blueprint Repository. The Target and Source Blueprints serve as inputs for resolving the VAT graph. The Target Blueprint together with its resolved VAt graph could be submitted back to the repository as a new Blueprint with multiple configuration alternatives
– Lifecycle Phase 3: Based on a particular contract, the resolved Graph is customized to become the final customized VAT Graph. This graph, bound to a particular contract, could be submitted back to the repository together as a configuration solution for the new Target Blueprint for a particular contract.
– Lifecycle Phase 4: Deploy the Target Blueprint based on the Final VAT Graph.

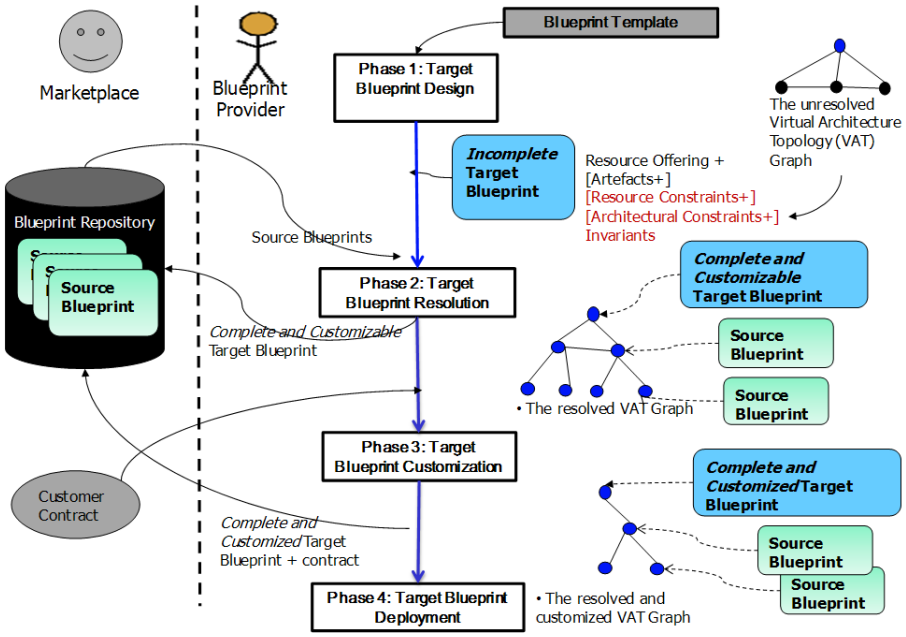

Fig. 7. The Blueprint Lifecyle

During the phase 1 of the lifecycle, the data structure needed to describe a Blueprint is the Blueprint template. The aim of the next section is to propose the structure of this Blueprint Template.

4 Blueprint Template

In the previous section we presented the definition of the *Blueprint* as a common standardized description for XaaS offerings. We also presented the Blueprint Lifecycle of a Blueprint provider, in which the definition of *Blueprint Template* is of particular importance. In this section, we present our approach and solution defining the Blueprint Template, a template for cloud computing that abstracts away from specific technical details and complexities of XaaS deployment.

The proposed Blueprint Template is located on the left-hand side of Figure 8. Using the template, the Blueprint provider can describe (instantiate) blueprints that capture their XaaS Offerings. In the middle of Figure 8, three instantiated Blueprints are introduced that capture the XaaS offerings of AutoInc and Tele1 in our motivating scenario in section 2. The right-hand side of the figure describes some blueprint extensions, which are the add-on data structures, the *non-functional profile* capturing the non-functional properties and the *business rule profile* containing the rules for the blueprints. These add-on extensions can be described using a variety of existing languages or templates such as the RuleML, WS-Policy, SLANG, etc., and will not be discussed further.

The Blueprint Template is divided into template sections, each has a set of proposed properties. Please note that the template is extensible, 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]}. In the following, each template section is dissected with a proposed set of properties.

4.1 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 nested blueprints in case one would like to offer a blueprint containing a bundle of nested XaaS offerings. Apart from the id are the *description, ownership, the version information, and release date* of the blueprint. While other properties can be described using primitive types, the ownership may need a more sophisticated data structure, e.g. a *StakeholderProfile* complex type that contains the name of the blueprint provider, its industry sector, location information, etc.

4.2 Offering section

Of particular interest for the blueprint consumer is the Offering section of the blueprint. We assume there is only one XaaS offering in each blueprint. However, a bundle of XaaS offerings can still be described as a single XaaS offering in a blueprint that can reference to other nested ones. Nesting references are specified in another template section (see section 4.5).

The name of the XaaS resource offered in the blueprint should described in such a way that the consumer can understand and query it from a blueprint repository. Initially, the *resource name* follows a simple string-based approach that allows only simple search and selection of blueprints. Nevertheless, in order

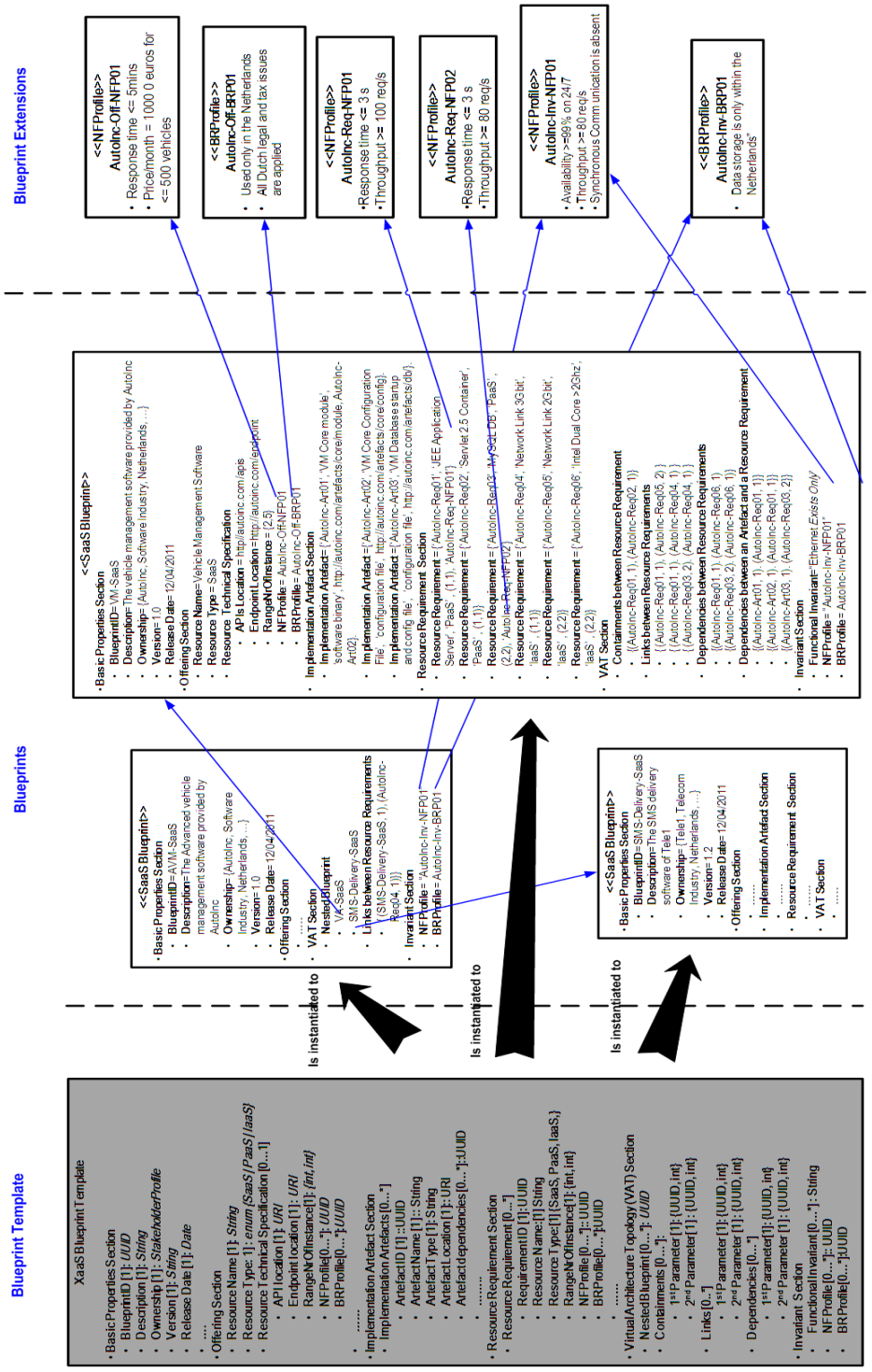

Fig. 8. The Blueprint Template and sample instantiated Blueprints

to enable a more accurate categorization of blueprints in the repository supporting more accurate search and matching, we propose add-on templates for describing XaaS functional capabilities in section 4.7 instead of a simple string-based resource name.

Each XaaS resource offering has a *resource type* indicating if it is a SaaS, PaaS, or IaaS offering, which follows the most fundamental classification of XaaS offering types. More specific types of XaaS offering could also be specified to enable a more accurate categorization in the blueprint repository.

The *Resource Technical Specification* is included in the blueprint for describing the technical interfaces to the blueprint consumers, the elasticity offering, the non-functional offering, and some business rules that constrain the blueprint offering. Technical interfaces comprise of the *API location* to download the necessary API and the *endpoint location* for programmatic interactions with the XaaS resource. The API includes not only the needed libraries but also the documentation for programming on the client side. Elasticity offering is specified in terms of the minimum and maximum number of instances of the XaaS resources (*RangeNrOfInstances*) that can be provided to the consumers. Non-functional properties of the offered XaaS resource can be specified in a number of separate profiles (*NFProfile*) using an add-on templates or external languages, e.g. WS-policy [17], SLAng [18], etc. Hence, the blueprint template allows to specify only the UUID pointer referencing to these separate profiles. Similarly, the business rules that constrain the XaaS offering can be specified in some existing rule languages in a separate business rule profile (*BRProfile*) that can be referenced to using the UUID pointer in the template. Approaches for BRProfile can be referred to the taxonomy of compliance constraints in [19] or the XML-based rule language specified in [20].

4.3 Implementation Artefacts section

This section is not of interest of the blueprint consumers, but is important for the system integrators who are responsible for the provisioning of this blueprint, as it contains the technical information of the artefacts that actually implement the XaaS offering. Each artefact has the following information: an *artefact id* for uniquely identifying an artifact, an *artefact name*, an *artefact type* indicating whether this artefact is a software binary, startup file or some other kinds of configuration files, an *artefact location* for downloading, and some *artefact dependencies* pointing to the other artefacts that have to be executed before executing this one.

Some examples of implementation artefacts can also be found in Figure 8. *AutoInc-Art01* is the UUID of a software binary called *VM Core module* that can be downloaded from a given URL. However, it depends on another artefact *AutoInc-Art02*, which is a provided configuration file that has to be executed before one can actually deploy the *AutoInc-Art01* artefact.

4.4 Resource Requirements section

A list of cloud resource requirements needed for deploying a blueprint is specified in this section. This specification indeed guides the system integrators to

search for the necessary XaaS resources offered in the marketplace. Each resource requirement is specified with a *resource name*, a *resource type* indicating whether it is a needed SaaS, PaaS or IaaS resource, the required *Range Number of Instances*, and a set of references pointing to *Non-Functional Profiles* and *Business Rule Profiles* that contain the non-functional properties and the rule constraints of the XaaS resource requirements. Similarly to the Offering section, the *resource name* can be described in a better way using the add-on template *XaaS functional capability* (described in Section 4.7) and the associated *Non-functional Profile and Business Rule Profiles* can be described using an existing external language.

As an example of resource requirements in Figure 8, the *VM-SaaS* blueprint needs *2 instances* of the resource *MySQL DB*, which is a *PaaS* requirement. This requirement has a unique id *AutoInc-Req03* and is associated with some required non-functional properties defined in the *AutoInc-Req-NFP02* profile, which prescribe that the required resource should respond faster than 3s and provide the throughput greater than 80 request per second.

4.5 Virtual Architecture Topology (VAT) section

The Virtual Architecture Topology (VAT) section specifies the to-be architectural requirements of the blueprint. By referencing certain information entities in the blueprint with their UUIDs, the VAT specifies the architectural requirements through the following relationships:

- The *Containment* relationships: This relationship can be used to describe the nesting relationships between blueprints. There exist also containment relationship between the resource requirements indicating that a required XaaS resource should be provided together with another XaaS resource.
- The *Link* relationships: A Link relationship in the VAT prescribes an abstract link between two elements, e.g. between two resource requirements or between a blueprint and a resource requirement, in the architecture topology. This topology helps the system integrator with the provisioning of the to-be system architecture, i.e. how to organize the available XaaS resources.
- The *Dependencies* relationships: This relationship indicates a deployment dependency between two elements, e.g. an artefact needs a required resource for its deployment.

Figure 9 captures the VATs of all the blueprints VM-SaaS and AVM-SaaS of AutoInc and the blueprint SMS-Delivery-SaaS of Tele 1, using the graph data structure. This joint VAT graph indicates that

- The *AVM-SaaS* Blueprint of AutoInc is a XaaS bundle containing both the *VM-SaaS* blueprint of AutoInc and the *SMS-Delivery-SaaS* blueprint of Tele1.
- The *AutoInc-Art01* artefact of the blueprint *VM-SaaS* depends on (i.e. needs to be deployed on) a JEE application server (*AutoInc-Req01*) that includes a servlet 2.5 container (*AutoInc-Req02*)

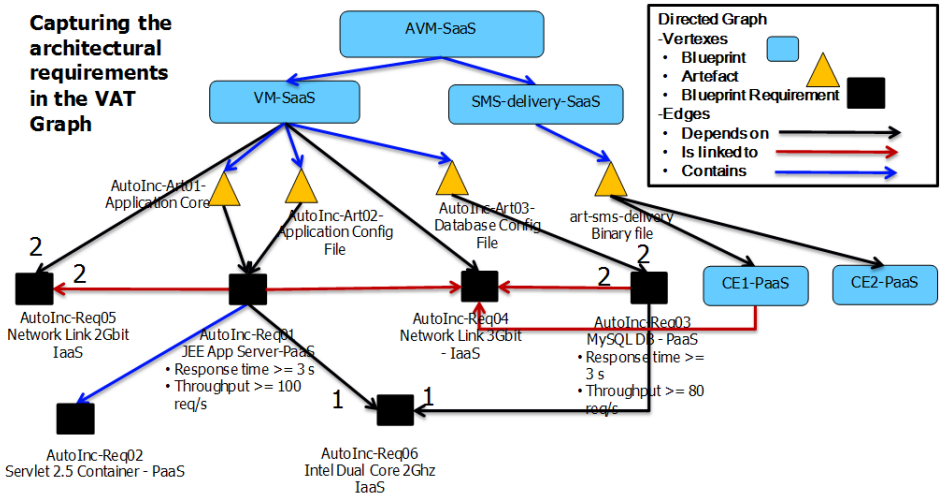

Fig. 9. A sample VAT graph of the AutoInc Scenario

- The required JEE application server (*AutoInc-Req01*) depends on (i.e. needs to be deployed on) an Intel Dual core machine with processor ¿= 2Ghz (*AutoInc-Req06*).
- The required JEE application server (*AutoInc-Req01*) is connected to two lines (i.e. two instances) of network link 2Gbit (*AutoInc-Req05*).
- The AVM-SaaS blueprint requires that the network link 3Gbit (*AutoInc-Req04*) required by the VM-SaaS has to be connected with the required composition engine (*Tele1-Req01*) of the SMS-Delivery-SaaS blueprint.
- ...

4.6 Invariants Section

Invariants are the special conditions that must not be violated by all the elements of a blueprint. The blueprint provider can specify the *functional invariants* that prescribe the functional conditions, as well as the non-functional and rule-based conditions in separate *Non-functional and Business Rule Profiles* respectively.

As an example in Figure 8 the provider of the VM-SaaS prescibes a functional invariant that all the network links required for the VM-SaaS must be Ethernet links. He also constrains his VM-SaaS with further non-functional invariants such as the throughput of the VM-SaaS must always be greater than 80req/s, its availability must be greater than 99% on 24/7, and no synchonous communications are allowed to be used among the systems. Data storage for the VM-SaaS must be only within the Netherlands, according to the rule-based invariant specified by the VM-SaaS provider.

4.7 Blueprint Template AddOn - The XaaS Functional Capability Template

So far in the template we use a simple string-based name for describing an offered or required XaaS resource, e.g. an offered SaaS with the name "Vehicle Management Software", a required PaaS with the name "Composition Engine", or a required IaaS with "Intel Dual core ¿= 2GHz". This simple way of describing a resource may be convenient for implementing the simple searching and selection of resources, yet not expressive enough for conducting correct and efficient search and matching algorithms. To remedy this problem, we propose additional templates for describing the functional capabilities of the XaaS resources, instead of a simple string-based name:

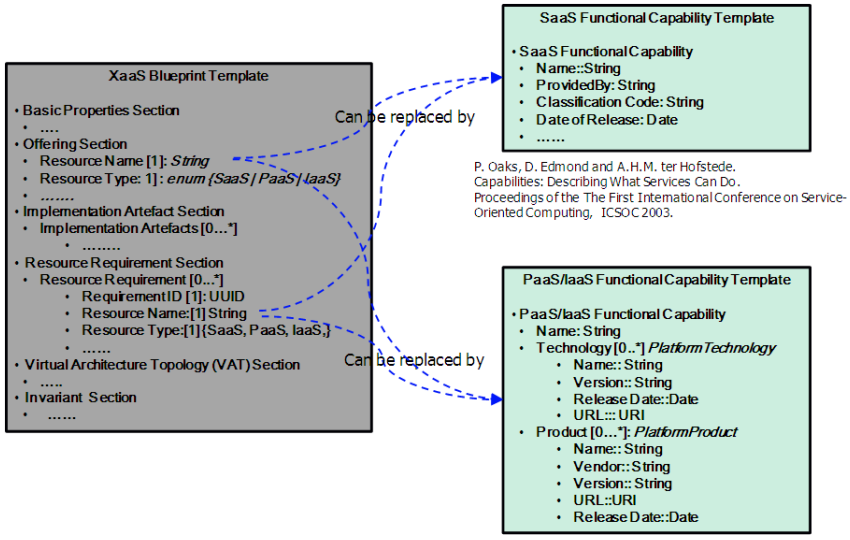

Fig. 10. The XaaS Functional Capability Templates Add-Ons

- SaaS Functional Capability Template: Functional capabilities of a SaaS resource can be described in a more expressive way using the structure proposed by Oaks et al in [21].
- PaaS/IaaS Functional Capability template: The template enables the specification of not only the name, but also more information about the technology and product specifications of the platform or infrastructure resource. For instance, using this template, a PaaS resource can be easily specified that it is actually a Tomcat server (product) from Apache (vendor) implementing the Servlet technology (technology) version 2.5.

5 Conclusion and Future Work

In this report we propose the *Blueprint* concept as the common foundation to address the design, engineering and provision of XaaSs through the cloud. By using templates for designing the blueprints, XaaS providers on the cloud can seamlessly participate in the creation of a true business ecosystem where applications, platforms and infrastructures from different providers can be traded, customized and combined. Such a business ecosystem has been exemplified by applying the template for a running scenario throughout the paper. In the future, we plan for conducting an empirical study on the applicability of the template by developing a prototype tool for designing and editing blueprints. Together with our industry partners in the 4caasT project [22] such as Ericsson, SAP, etc., we are currently in the progress of implementing the first business ecosystem in which the first version of the blueprint prototype tool will be evaluated.

6 Acknowledgment

The author would like to thank Francesco Lelli, Yehia Taher, Michael Parkin, Mike P. Papazoglou and Willem-Jan van den Heuvel for their collaborating work towards this result.

References

1. Mell, P., Grance, T.: The nist definition of cloud computing. National Institute of Standards and Technology, Information Technology Laboratory (July 2009)
2. Andrikopoulos, V., et al.: State of the art report on software engineering design knowledge and survey of HCI and contextual knowledge. Project deliverable PO-JRA-1.1.1, S-Cube Network of Excellence (July 2008)
3. SAP A.G.: The SAP Service Marketplace
4. European Comission: Information and Communication Technologies Unit: 4caast: Building the paas cloud of the future. Project Objectives (September 2010)
5. Monteiro, A., Pinto, J., Teixeira, C., Batista, T.: Cloud interchangeability - redefining expectations. In: In Proceedings of CLOSER'11. (2011)
6. Keahey, K., Tsugawa, M., Matsunaga, A., Fortes, J.: Sky computing. IEEE Internet Computing **13**(5) (September/October 2009) 43–51
7. DMTF: Open Virtualization Format (OVF), http://www.dmtf.org/standards/ovf
8. DMTF: Dmtf to develop standards for managing a cloud computing environment, http://www.dmtf.org/standards/cloud
9. Galán, 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
10. Bernstein, D., Ludvigson, E., Sankar, K., Diamond, S., Morrow, M.: Blueprint for the intercloud - protocols and formats for cloud computing interoperability. In: In Proceedings of the Fourth International Conference on Internet and Web Applications and Services, IEEE Computer Society (2009)
11. Brunelière, H., Cabot, J., Frédéric, J.: Combining model-driven engineering and cloud computing. In: Proceedings of the 4th edition of Modeling, Design, and Analysis for the Service Cloud. (June 2010)

12. Hamdaqa, M., Livogiannis, T., Tahvildari, L.: A reference model for developing cloud applications. In: In proceedings of CLOSER'11. (2011)
13. Cai, H., Zhang, K., Wang, M., Li, J., Sun, L., Mao, X.: Customer centric cloud service model and a case study on commerce as a service. In: In Proceedings of the IEEE International Conference on Cloud Computing. (2009)
14. Thrash, R.: Building a cloud computing specification: fundamental engineering for optimizing cloud computing initialtives. Computer Science Corporation (CSC) Whitepaper (August 2010)
15. Mietzner, R.: A method and implementation to define and provision variable composite applications, and its usage in cloud computing. Dissertation, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany (August 2010)
16. Amazon Web Services: Aws cloudformation http://aws.amazon.com/de/cloudformation/
17. Verma, K., Akkiraju, R., Goodwin, R.: Semantic matching of web service policies. In: Proceedings of the Second Workshop on SDWP, 2005. (2005) 79–90
18. Skene, J., Lamanna, D.D., Emmerich, W.: Precise service level agreements. In: In: Proc. of 26th Intl. Conference on Software Engineering (ICSE, IEEE Press (2004) 179–188
19. Elgammal, A., Türetken, O., van den Heuvel, W.J., Papazoglou, M.P.: Root-cause analysis of design-time compliance violations on the basis of property patterns. In: ICSOC. (2010) 17–31
20. The RuleML Initiative: Rule markup language v0.91, http://ruleml.org (2011)
21. Oaks, P., Edmond, D., ter Hofstede, A.: Capabilities: Describing what services can do. In: Proceedings of the The First International Conference on Service-Oriented Computing, ICSOC 2003, pp 1 - 16. (2003)
22. EC's 7th Framework project 4caasT: http://4caast.morfeo-project.org/