Abstract—The cloud computing landscape has recently developed into a spectrum of cloud architectures, leading to a broad range of management tools for similar operations but specialized for certain deployment scenarios. This both hinders the efficient reuse of algorithmic innovations within cloud management operations and increases the heterogeneity between different management systems. Our overarching goal is to overcome these problems by developing tools general enough to support the full range of popular architectures. In this contribution, we analyze commonalities in recently proposed cloud models (private clouds, multi-clouds, bursted clouds, federated clouds, etc.), and demonstrate how a key management functionality - service deployment - can be uniformly performed in all of these by a carefully designed system. The design of our service deployment framework is validated through a demonstration of how it can be used to deploy services, perform bursting and brokering, as well as mediate a cloud federation in the context of the OPTIMIS Toolkit.

Index Terms—Cloud Computing; Cloud Architecture; Service Deployment

I. INTRODUCTION

In the context of cloud computing, deployable services are encapsulated in virtual machines (VMs), and deployment is performed by instantiating VMs on top of a virtualized infrastructure. This new way of service deployment enables a traditional on-premises application to be rapidly redeployed as Software as a Service.

In this paper, we present a novel approach to service deployment, general enough to meet the requirements of a range of common cloud scenarios, including private clouds, bursted clouds, federated clouds, multi-clouds, and cloud brokering. We identify key requirements for service deployment in these scenarios and present the architecture for a service deployment tool to meet these requirements. Our proposed tool interacts with components for data management, service contextualization, and service management in its orchestration of the service deployment process.

Our approach is validated by implementation and integration in a private cloud, a bursted cloud, and a brokered multi-cloud scenario using tools from the OPTIMIS Toolkit [15] providing the required complementing functionalities. The verification study is performed in a cross-European testbed consisting of cloud resources at Atos (Spain), BT (UK), Flexiant (UK), and Umeå University (Sweden).

The remainder of the paper is organized as follows. Section II describes fundamental cloud concepts and outlines the service lifecycle. Section III discusses the studied cloud deployment scenarios. Core requirements for service deployment in cloud environments are described in Section IV. Section V describes the service deployment process. Section VI presents the design of our service deployment solution. Section VII contains a validation study of our approach in the context of OPTIMIS toolkit. Related work within service deployment is described in Section VIII. Finally, our conclusions are presented in Section IX, followed by a presentation of future work, acknowledgments, and a list of references.

II. CLOUD SERVICE CONCEPTS

Cloud services can be categorized into Software as a Service, Platform as a Service, and Infrastructure as a service, or SaaS, PaaS, and IaaS for short [31]. In a cloud service deployment scenario the two stakeholders are the Infrastructure Provider (IP) and the Service Provider (SP). An IP offers infrastructure resources such as VMs, networks, and storage which can be used by SPs to deliver SaaS solutions to their customers. The SPs can also use PaaS tools to develop their services, or offer this functionality to their customers who may want to construct and deploy custom services. Without loss of generality, we concentrate in this contribution on the cases where a SP or IP deploys services to an IP providing IaaS.

A. Deployable Services

IaaS is based on virtualization technology which means that a deployable cloud service is in fact a VM or a collection of VMs. We refer to a VM of a certain type as a component and note that a service can consist of multiple components. For example, a three-tier web application service may consist of a database component (e.g., MySQL), an application component (e.g., Weblogic server [8]), and a presentation layer component (e.g., Apache server).

The information about which components the service is composed of along with functional and non-functional requirements for a deployment target is described in a document, the service manifest. The service manifest can also define elasticity bounds for the service, i.e., upper and lower limits for how many instances of a component that may be provisioned at any
time. These bounds are commonly associated with elasticity rules for when to scale up or down the number of instances of a component, and such rules can range from simple condition-action statements to complex expressions that reason about statistical properties of the service workload. In addition, a service manifest typically contains various constraints such as desired geographical location, and data protection requirements.

B. The Service Lifecycle

The lifecycle of a cloud service can be summarized as construction, deployment, operation, and undeployment. In the construction phase, the service applications (Virtual Appliances) are implemented and packaged into a set of VMs. The construction of the above discussed service manifest ends the service construction phase. The service deployment includes identification of a suitable deployment target, installation of the service VMs in the selected provider, and initialization of these VMs by the provider, i.e., VMs are booted, configured, and start to deliver the service. In the operation phase, the IP, and potentially also the SP, perform a set of management actions to ensure efficient and robust provisioning of the service. Once the service is no longer needed, it can be undeployed by the SP, upon which the IP shuts down the running VMs and removes any assets of the service. Notably, multiple instances of the same service can be created from a single service manifest and these instances can be shutdown or restarted as needed.

III. CLOUD DEPLOYMENT SCENARIOS

Cloud environments can be set up differently depending on the types of interaction between the collaborating providers. The main differences between the scenarios are the number of involved actors and which actor is in control during the deployment process. The scenarios described in this section have been proposed and discussed in previous research [10], [15], [28], [31], [34], albeit typically in isolation and they have not been compared in the context of service deployment.

Despite the rapid adoption of public clouds, there are several security and privacy concerns associated with service deployment to public IPs. To address these issues, a SP can set up a cloud infrastructure for its own internal use, commonly referred to as a private cloud, which is illustrated in Figure 2. Private clouds can circumvent many of the security and privacy concerns related to hosting sensitive information in public clouds, and may also offer stronger guarantees on control and performance as the service as well as the whole infrastructure is administered from within the same domain.

Private clouds may offload capacity to other IPs under periods of high workload, or for other reasons, e.g., planned maintenance of the internal servers. In this scenario, the providers form a hybrid architecture commonly referred to as a bursted cloud as seen in Figure 3. Typically, less sensitive tasks are executed in the public cloud instead while tasks that require higher levels of security are provisioned the private infrastructure.

Federated clouds are IPs collaborating on a basis of joint load-sharing agreements enabling them to offload capacity to each other [28] in a manner similar to how electricity providers exchange capacity. The federation takes place at the IP level in a transparent manner. In other words, a SP that deploys services to one of the IPs in a federation is not notified if its service is off-loaded to another IP within the federation. However, the SP is able to steer in which IPs the service may be provisioned, e.g., by specifying location constraints in the service manifest. Figure 4 illustrates a federation between three IPs.

If the SP itself is involved in selecting which IP a service should be deployed or re-deployed to the scenario is known as a multi-cloud. In multi-cloud deployments, such as in Figure 5,
the SP is responsible for planning, initiating and monitoring the execution of services. Notably, we are implicitly considering split deployment scenarios, i.e., when the components of the service are deployment across multiple IPs.

![Cloud federation](Image)

**Fig. 4.** Cloud federation.

A related scenario is that when a cloud broker [34] handles the complexity of prioritization and selection of IPs, and may also offer value-added services to IPs and SPs. In this case, the broker may have pre-arranged agreements with a number of IPs and selects the best match for a service based on the SP's desired criteria. The broker operates between the SP and the IPs, offering an IP-like interface to SPs and a SP-like interface to IPs, as illustrated in Figure 5.

![Multi-cloud scenario](Image)

**Fig. 5.** Multi-cloud scenario.

**IV. REQUIREMENTS FOR SERVICE DEPLOYMENT**

A general approach to service deployment should work transparently in all types of clouds. Based on the deployment scenarios discussed in Section III, we identify a number of requirements for service deployment. Notably, these requirements for service deployment have significant similarities with the tasks identified in the overall process for resource selection (scheduling) in Grid computing environments [30]. In the below, the SP is normally the actor who negotiates terms with the IPs, initializes the service deployment, and performs any associated tasks. However, in the cases of cloud federation and bursting, these interactions occur between two IPs, with the IP initializing the deployment process acting as the SP.

- **Discovery of IPs.** In order to deploy a service, the deploying actor must identify the IPs that are available for deployment. IPs can be discovered by looking them up in a registry or by using auto-discovery mechanisms. We remark that discovery (along with the later filtering and selection) of an IP is trivial in the private cloud case, as only a single IP is available.
- **Filtering of available IPs.** In order not to add overhead by negotiating deployment with IPs that fail to fulfill fundamental requirements for the particle service to be deployed, an initial filtering of the list of IPs retrieved during IP discovery must be possible. Criteria for filtering include both functional aspects, e.g., support for certain hypervisors and VM image formats, as well as non-functional criteria such as constraints based on the country in which the IP is based (for legal and/or data-protection reasons).
- **Service Manifest construction.** Each service must include a service manifest that describes the functional and non-functional parameters of the service. A service manifest is an abstract definition of the service, which is used to negotiate with IPs and later becomes part of the service agreement with the IP. Data specified in the service manifest, i.e., VM disk images, must also be prepared. A set of utilities for creation, modification, etc. of service manifests would greatly simplify this procedure.
- **Negotiation and deployment optimization.** A SP must be able to negotiate with available IPs for service hosting offers. Note that it is not always desirable to deploy the whole service to the same provider. For reasons such as security, performance, and fault tolerance it can be preferable to split the service between several IPs which means that a negotiation can be for part of a service. Based on the results of these negotiations and data such as reputation statistics, that could be gathered and evaluated by third-party entities, the SP must decide where to deploy the service.
- **Service contextualization.** When a service component’s disk images are generated all required information is not known. Data such as locations of shared network resources and security credentials depends on the context in which the service is deployed. In order to launch the service successfully, this information must be propagated to the VM. A possible mechanism for this contextualization process is to embed various scripts in the VM images that dynamically retrieves information upon boot, enabling VM instances to self-contextualize.
- **Service data transfer.** The contextualized VM images, along with any other data required by the service, must be transferred to the IP. To guarantee properties such as confidentiality and data integrity during this transfer, reliable security mechanisms are required.
- **Service Level Agreement creation.** To ensure that the service operates according to the SP’s expectations, it must be possible to establish a Service Level Agreement (SLA), that governs the relationship between the SP and
IP for the provisioning of the service. SLA’s for service provisioning commonly include segments that address service definition, performance measurements, problem management, customer duties, warranties, disaster recovery, as well as conditions for termination of the agreement [1]. Penalties may be agreed upon in the case of non-compliance with the terms in the SLA.

V. THE SERVICE DEPLOYMENT PROCESS

Based on the requirements study in the previous sections, we derive a general sequence for service deployment. This process, consisting of tasks to be performed, is illustrated by the sequence diagram in Figure 7. Notably, the complexity and details of the service deployment process may vary with the deployment scenario, but the process remains similar.

Step 1, service construction, is identical in all scenarios. The SP constructs (implements, packages, etc.) a service in the same way no matter how it will be deployed. Step 2, discovery and filtering of IPs to find out what IPs are available for the deployment, is trivial in the private cloud case as the IP is already known. It is also relatively simple in cloud brokering scenarios as only the broker needs to be known. However, in federation, public, and multi-cloud cases this step can be quite complex, involving auto-discovery mechanisms and/or IP registries.

Most of the algorithmic complexity in service deployment is associated within the related tasks of SLA negotiation and IP assessment (Steps 3 and 4 in Figure 7). In scenarios where the SP interacts with a single provider, these tasks are simplified. Conversely, for federation, bursting, and multi-cloud deployments, interaction with more than one IP complicates the process. The richness of the negotiation protocol can range from simple versions with primitives such as offer, accept, and refuse, to more complicated versions with counter-offers, and approaches based on auctions. An in-depth analysis of negotiation protocols is beyond the scope of this paper and further details on this topic are given, e.g., by Sarangan et al. [29] and Jennings et al. [16]. Similarly, for IP assessment, the complexity of estimating the utility associated with deploying the service in each potential provider can differ significantly based on the modeling method used. Algorithms proposed for optimizing provider selection include scheduling-inspired combinatorial optimization approaches such as integer programming, which are commonly suggested [13], [34], but tend to scale very poorly with the number of IPs. Other approaches include heuristic solutions [21] that trade optimality for faster decision-making.

Once the most suitable provider (or potentially, set of providers) is identified, the SP performs contextualization (Step 5 in the sequence diagram) to prepare the service VM images with any dynamic information that is needed for these to boot and configure themselves properly. This step is more complicated if the service is split among several IPs, as an external rendezvous mechanism is typically required in order to initialize cross-provider networking for the VMs of the service.

After the VM images are properly configured, they are uploaded to the selected provider(s) as illustrated in Step 6 of Figure 7. As VM images typically are very large, significant performance gains can be achieved by proper tuning of network parameters. In private clouds where a network file system may connect the SP and the IP, image transfer is much less of an issue. Alternatively, if an IP does not support upload of SP-defined VM images, a custom service image must be pre-created (based on templates from the provider) and stored at that IP. In such a case, contextualization abilities are significantly reduced.

When the contextualized VM images are stored in the IP’s repository, the SP confirms the offer negotiated in Step 3 and a SLA is created between the SP and the IP for the operation of the service, as illustrated in Step 7. Once again, this step becomes more complex if the SP needs to aggregate multiple SLAs from different IPs.

Finally, Step 8 in Figure 7 illustrates that once the service is deployed, the SP stores information about the deployment in a registry to enable subsequent service monitoring, management, and undeployment.

VI. PROPOSED SERVICE DEPLOYMENT ARCHITECTURE

To meet the requirements of service deployment, we propose a service deployment architecture. The purpose of architecture is two-fold - it is responsible for generating optimal deployment solutions for a service, and for coordinating the deployment process in order to provision a service according to the deployment plan. In order to separate the placement optimization from the deployment coordination functionality, our proposed software, referred to as the Service Deployment
Optimizer (SDO), is divided into two components, the Service Deployer (SD), and the Deployment Optimizer (DO), both illustrated in Figure 8. The DO is a decision-making component and the SD is a module that orchestrates the DO and various utility functionalities in order to perform the deployment sequence described in Figure 7. Notably, to provide a complete solution for cloud deployment, the SD and DO interacts with external components for service contextualization, data management, service management, and IP assessment, all illustrated in Figure 8 and further discussion in Section IV. We remark that these external components may need customization and/or replacement depending on, e.g., the protocols and data formats used by the IPs.

We outline the main design rationale for the SD and DO components below, as well as discuss how they interact with each other and related utility functionalities for data transfer, etc.

A. Service Deployer

The purpose of the SD is to coordinate the deployment and interact with the other involved parties in a deployment. The SD takes a service deployment request, contacts the IP discovery service to obtain which providers are available and performs filtering (see steps 1-2 in Figure 7). To retrieve an optimal placement scheme, SD contacts the DO who performs calculation for placement optimization. Once an optimal placement solution is returned, the SD deploys a whole service following steps 5-8 in Figure 7 with the support of external components. Service Contextualization is in charge of contextualizing VM images, Data Management is responsible for data transfer from the SP side to the IP side, Service Management creates service resource and updates resource accordingly, and SLA Management handles the IP side creation of agreement.

B. Deployment Optimizer

The DO’s role in a deployment is to perform placement optimization based on the inputs from the SD, including a service manifest, the optimization objective, and available IP info, etc. Based on this information, the DO generates an optimal placement scheme for the service. In order to achieve an optimal placement objective, the DO may split services that contains more than one component into several sub-services, and map them to different IPs. This is provided it can do so without breaking affinity constraints specified in the service description. During the calculation, the DO negotiates with IPs and the IP assessment tools, see steps 3-4 in Figure 7. Optimization techniques such as combinatorial optimization, problem relaxations and heuristic approaches such as greedy formulation can be applied in this component.

VII. Validation Study

In order to verify that our service deployment architecture is suitable for the envisioned cloud architectures, we perform a validation study. The study is carried out in the context of the OPTIMIS Toolkit [15], which includes a set of independent components that can be adopted, either in full or in part, by IPs that provide infrastructure resources, and by SPs that use these capacities to deliver services. The study comprises three cloud service deployment scenarios: private cloud, cloud brokerage, and cloud bursting.

In these three scenarios, the service we use for validation is a composite service for gene detection presented in [33]. This service contains five components. First, there are four functionality components which contribute to the overall gene detection process: translation of the input genomic database to a given format (component GA); obtention of a list of amino acid sequences which are similar to a reference input sequence (component GB); search of the relevant regions of the genomic database (component GC) and execution of the GeneWise [12] algorithm on them (component GD). Additionally, there is one component for coordination (component GP). Each component is encapsulated in a VM sized approximately 9.8 GB. To avoid repetitive data transfer, only one VM image is transferred from the SP to the IP in case multiple components are deployed to the IP. This way, multiple VM instances can be started from the same image by associating each instance with different contextualization data.

For the validation, we use a distributed testbed with four IPs located across Europe: Atos [2] (Spain), BT [3] (UK), Flexiant [4] (UK) and Umeå University (Sweden). Each IP site hosts selected parts of the OPTIMIS Toolkit, as well as fundamental management software such as Xen [5] and Nagios [6]. The role of the IPs in the different scenarios is summarized in Table I. Notably, our goal is not to evaluate the various providers but rather to investigate how well our proposed approach adapt to real scenarios.

<table>
<thead>
<tr>
<th>Use case configurations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private cloud</td>
</tr>
<tr>
<td>Cloud brokerage</td>
</tr>
<tr>
<td>Cloud bursting</td>
</tr>
</tbody>
</table>
A. SDO in OPTIMIS

We below outline how the SDO is integrated with selected components of the OPTIMIS Toolkit, i.e., how the architecture outlined in Figure 8 is achieved in an OPTIMIS cloud scenario.

- **IP Discovery**: In OPTIMIS, IP information is registered in a simple on-line registry accessed through a REST interface. In this registry, information such as IP identifier, IP name, and endpoints for negotiation, etc., are stored.
- **VM Contextualization**: The OPTIMIS VM Contextualization component provides an interface for constructing service context data, such as security certificates, VPN hostnames, VPN DNS and Gateway IP addresses, mount points for network data stores, monitoring manager hostnames, off-line software license tokens, as well as list of software dependencies [11].
- **Data Manager**: In OPTIMIS, a Hadoop-based [7] Data Management service enriched with RESTful APIs is combined with a series of tools that aim to extend Hadoop’s functionality beyond its well known scope - heavy data processing [19]. In all, these tools provide data management functionalities to cloud services as well as the capabilities needed to deploy VM images to IPs.
- **SLA Management**: A service and client based on WS-Agreement protocol [9] is used in OPTIMIS for negotiating and creating Service Level Agreements between IPs and SPs [20].
- **Infrastructure Provider Assessment**: for OPTIMIS, we implement a two-step IP assessment strategy, where in a first step, IPs that do not fulfill functional requirements or have unsuitable data protection levels, etc. are filtered out. In a second step, the DO negotiates deployment terms (cost, etc.) with the remaining IPs and ranked these on basis of assessment of four non-functional properties: trust, risk, eco-efficiency, and cost [18], [27].

B. Scenarios Descriptions and Statistics

- Private cloud:

  In the private cloud scenario, the SP (also located in the Atos cloud) submits the gene detection service deployment request to the Atos cloud. All components (GA, GB, GC, GD, and GP) are deployed to the Atos IP.

- Cloud brokerage (multi-cloud):

  In the cloud brokerage scenario, two SDO instances are running: one in the Umeå cloud, which plays the role of the SP. The other one is located in the BT cloud, which plays the role of a cloud broker. The SP submits the gene detection service deployment request to the Umeå cloud. Instead of deploying the service by itself, the SDO in the Umeå cloud calls the SDO on the broker to complete the deployment. There are three IPs registered in the IP registry which can be queried by the SDO in the BT cloud. After the by Deployment Optimizer’s calculations (including IP assessment, negotiation, and placement optimization) two IPs are selected to host the service. Specifically, two components (GC and GD) are to be deployed to Flexiant cloud, the other three (GA, GB, and GP) are to be deployed to the Atos cloud. For the purpose of this demonstration, VM images are stored on the broker in advance.

- Cloud bursting:

  In the cloud bursting scenario, the service is already deployed in the Flexiant cloud. To fulfill a demand for increasing service capacity from the SP, the Flexiant cloud needs to launch two more instances respectively for two of the five components (i.e., GC and GD) in the service. For financial reasons, the Flexiant cloud decides to outsource this demand to a more cheaper cloud provider, i.e., Atos cloud, while maintaining its SLA-agreement with SP.

C. Experimental Results

In order to assess the performance of the SDO and the complexity of the service deployment process as such, we measure the duration of the main steps of deployment for each studied cloud architecture. Table II presents statistics of time consumed in each phase of service deployment for each scenario.

From our experiments, we conclude that the major part of the time (around 90-95% depending on the scenario) is used to transfer VM images from the SP to the IP. Notable, the differences in image transfer time among the scenarios are due to the complexity of the deployment solution. For the private cloud scenario, all components are deployed to Atos and only a single VM image thus needs to be transferred, over an internal network. In contrast, for the multi-cloud scenario, two VM images are transferred, one from BT to Atos (for components
GA, GB, and GP), the other one from BT to Flexiant (for components GC and GD), both transfers taking place over Internet.

Another observation is that placement optimization becomes more complex in the multi-cloud case. Due to the possibility of split deployment, the number of potential service configurations is much larger in a multi-cloud scenario than for the private and bursting cases. In the brokering case, multiple negotiations are performed between the broker deployed at BT and the Atos and Flexiant IPs. The actual assessment of the IPs’ suitability for provisioning the service includes complex statistical modeling techniques [18], [27] to assess the trust, risk, eco-efficiency and cost factors for each potential placement. These models all have in common that they in the assessments make extensive use of a database with historical information about past IP behavior. Notably, the detailed elaboration of optimization techniques used in the section is outside the scope of this paper, yet it can be found in [23].

In summary, the private cloud scenario demonstrates how the SDO can be used to complete a service deployment in general. The cloud brokerage scenario demonstrates brokerage across multiple cloud providers. The cloud bursting scenario shows how organizations can utilize the SDO to scale out their infrastructure, using resources from third-party providers based upon a range of factors such as trust, risk assessment, eco-efficiency and cost [18], [27].

### VIII. RELATED WORK

Talwar et al. [32] review approaches for service deployment before the emergence of Cloud Computing. They compare and evaluate four types of service-deployment approaches: manual, script-, language-, and model-based solutions, in terms of scale, complexity, expressiveness, and barriers for first time usage. They also conclude that service deployment technologies based on scripts and configuration files have limitations in expressing dependencies and verifying configurations, and usually result in erroneous system configurations. Conversely, language- and model-based approaches handle these challenges better, but at the expense of comparatively higher barriers for first usage.

With the emergence of cloud computing, services are provisioned using VMs. Service deployment can be done by initializing VMs with their virtual appliances. Cloud service developers are thus enabled to deploy applications without confronting the usual obstacles of maintaining hardware and system configurations. Much work have been done in the context of this new service-deployment technology. Most focus has been on deployment optimization approaches. For example, Kecskemeti et al. [17] propose an automated virtual appliance creation service that aids the service developers to efficiently create deployable virtual appliances. They reduce the deployment time of the service by rebuilding the virtual appliance of the service on the deployment target site. For optimal VM deployment across multiple cloud providers, Chaisiri et al. [13] propose an stochastic integer programming (SIP) based algorithm aimed at minimizing the cost for a placement plan for hosting VMs in a multiple cloud provider environment under future demand and price uncertainty. Similarly, Vozmediano et al. [26] [25] explore the use case of deploying a compute cluster on top of a multi-cloud infrastructure, for use by loosely-coupled Many-Task Computing (MTC) applications. They conclude that cluster nodes can be provisioned with resources from different clouds to improve the cost-effectiveness of the deployment, or to implement high-availability strategies.

Our previous contributions in this field include cloud brokering mechanisms [34] for cost- and performance-optimal placement of VMs across multiple cloud providers in static scenarios, and extensions to this with a linear programming model to dynamically reschedule VMs (including modeling of VM migration overhead) upon changed conditions such as price changes, service demand variation, etc. in dynamic cloud scheduling scenarios [22]. In addition, we have proposed an approach to optimal VM placement within data centers for predictable and time-constrained load peaks [21].

Although algorithms for optimizing service deployment is a very active area of research, with lot of interest is given to the various deployment architectures in general, the topic of this contribution, namely architectures and tools general enough to support multiple cloud deployment scenarios, has received far less attention to date.

### IX. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a general approach to automatic service deployment in cloud environments, based on our study of cloud architectures and deployment scenarios and the core requirements for service deployment derived from these. A validation study performed in the context of the OPTIMIS Toolkit verifies the feasibility of a general service deployment component that can be reused across multiple cloud architectures. Our validation study also gives some indications about the performance aspects of cloud service deployment, identifying transfer of VM images as the most time-consuming task.

Future directions for this work includes in-depth studies of algorithms for optimized selection of deployment targets. Another topic of future research is the incorporation of re-deployment, i.e., migration of the full service, or some of its components, to other IP(s) during operation [14]. Reasons for re-deployment include improved performance, and improved cost-efficiency. In such scenarios, a careful tradeoff between re-deployment overhead and expected improvement must be
considered [22]. Additionally, a model of interconnection requirements that can precisely express the relationships between components within a service to be deployed can be another promising direction to investigate. Such a model can help SDO optimizing the service deployment with e.g., less communication cost between service components. In addition, we are working on a specific scenario where cloud users can specify hard constraints and soft constraints when demanding resource provisions. A hard constraint is a condition that has to be satisfied when deploying services, i.e., it is mandatory. In contrast, a soft constraint (also called a preference) is optional. An optimal placement solution with soft constraints satisfied is preferable over other solutions. We are also investigating how to apply multi-objective optimization [24] techniques to this scenario.

ACKNOWLEDGMENTS

We acknowledge the members of the OPTIMIS team for their valuable help in integration and testing for the validation scenario. We also thank Atos, BT, and Flexiant for providing servers for the distributed testbed, as well as Daniel Espling for helping with figures 1-6. Financial support has in part been provided by the European Community’s Seventh Framework Programme (FP7/2010-2013) under grant agreements no. 257115 (OPTIMIS), and the Swedish Government’s strategic effort eSSENCE.

REFERENCES