TOWARDS HOLISTIC CLOUD MANAGEMENT

JOHAN TORDSSON, KARIM DJEMAME, DANIEL ESPLING, GREGORY KATSAROS, WOLFGANG ZIEGLER, OLIVER WÄLDRICH, KLEOPATRA KONSTANTELI, ALI SAJJAD, MUTTUKRISHNAN RAJARAJAN, GEORGINA GALLIZO, AND SRIJITH NAIR

Despite significant attention and substantial efforts both in industry and academia, cloud computing has yet to reach its full potential. Commonly stated obstacles for cloud adoption include confusion due to multiple delivery models (SaaS, PaaS, IaaS) and deployment scenarios (public clouds, private clouds, cloud bursting etc.). Other frequent concerns relate to risks with outsourcing, data legislation issues, inability to assess trust in external providers, etc. In addition to these obstacles to cloud computing as a concept, there are also technical thresholds in today's cloud offerings, making cloud infrastructure provisioning and service lifecycle management tedious processes. We aim to address these issues by research in five main directions: cloud service lifecycle optimization, adaptive self-preservation, self-management based on non-functional criteria, support for multiple cloud architectures, as well as market and legislative studies. The main outcome of our research is the OPTIMIS Toolkit, a set of flexible tools for service and infrastructure self-management.

Introduction

With its promise of close to infinite scalability, rapid provisioning, and a pay-per use charging model, cloud computing appears to be a compelling paradigm for service provisioning. However, as contemporary cloud offerings typically target functionalities too close to infrastructure, organizations must undertake significant efforts to manage their services in a cloud. In addition, the at times confusing cloud landscape with multiple models (SaaS, PaaS, IaaS) and architectures (private, public, federated, and bursted clouds) introduces additional thresholds to adoption, due to a lack of tools that abstract and hide the complexity. With its outsourcing nature, the cloud paradigm in itself introduces challenges related to trust and risk, as well as security and privacy of services and data, making organizations hesitate to migrate sensitive or mission-critical services to the cloud. For the rest of this chapter, we use the term *service* exclusively to describe an application that is accessed online by end users, and which is delivered by a cloud infrastructure provider through the use of one or more Virtual Machines (VMs).

We summarize our analysis of cloud computing obstacles in five high-level research challenges, discussed in more depth in our previous work [OPTIMIS2011c]:

- *Service life cycle optimization* concerns practices and tools to facilitate and optimize the construction (implementation and/or orchestration), deployment, and operation of cloud services.
- *Adaptive self-preservation* constitutes a wide range of self-management activities with increased autonomy as the ultimate goal, enabling fewer administrators to handle increasingly more services and larger infrastructures.
- Cloud and service *self-management based on non-functional management criteria* extends beyond traditional performance tuning to also incorporate non-functional aspects such as trust, risk, eco-efficiency, and cost (henceforth TREC) in self-management decision making.
- With *support for multiple cloud architectures*, a single abstraction can be used to construct a wide range of clouds. These include private clouds where one organization provisions both services and the infrastructure needed by these; bursted clouds where organizations temporarily outsource non-critical workloads; federated clouds of collaborating cloud infrastructure providers that mutually share workloads; multi-cloud deployment, where a service owner negotiates with, and deploys a service across, multiple cloud infrastructures, with the negotiation process potentially being mediating by a cloud broker.
- Studies of *market and legislative issues* includes identification of new market roles and business models for clouds, as well as investigation of legal aspects related to the acquisition, transfer, and storage of service data.

Our main research objective is to make significant progress in these five areas and to incorporate our results into the OPTIMIS Toolkit. The OPTIMIS Toolkit is a set of software components aimed to simplify and optimize construction, deployment, and operation of services as well as operation of the virtualized infrastructure required to deliver these services. The toolkit is not a turn-key solution but rather a set of software components that can be selected and composed for use in a variety of cloud deployment scenarios, as well as be customized through policies to meet different Business Level Objectives (BLOs).

The rest of this chapter is organized as follows: Section 2 outlines the working process and requirement engineering method used in the development of the OPTIMIS Toolkit. Section 3 describes the architecture of the toolkit, highlighting self-assessment aspects, service lifecycle management, and secure interconnection of multiple clouds. Section 4 presents our conclusions.

Process and Requirements

A multi-faceted project such as OPTIMIS with a wide range of overall objectives requires a structured working process. In the following, the used goal-oriented requirements process is outlined along with the overall requirements identified. The quality of requirements has a direct impact on the chances of success of the project. Therefore, an important task is to capture and structure the project requirements in order to ensure a number of qualities such as precision, completeness, consistency, traceability, and testability [OPTIMIS2010a]. In order to achieve these qualities, the requirements were elaborated based upon:

- A full project glossary capturing the domain vocabulary.
- A scenario approach to discover functional requirements of various stakeholders, using precise UML 2.0 sequence diagrams, both for normal and exceptional cases.
- A goal oriented methodology, to structure those scenarios into goal trees with a clear distinction between requirements under the responsibility of some component to design, assumptions on the existing environment, and the ability to check for consistency and completeness.
- The ISO-9126 standard to document all non-functional requirements.
- The definition of traceability mechanisms to reference requirements and also show the coverage between goals and scenarios, directly useful for verification and validation test cases.
- A methodology that ensures the different sources of the requirements (general, use cases, legal, business) are consolidated in a way that ensures requirements are specific, measurable,

attainable, relevant, and time bound (SMART) to the software development.

These approaches were combined to build a requirements document that follows the IEEE-830 recommended practices for software requirements specifications. This document also helps understanding the project objectives and their rationale; the discovery of the main actors, their responsibilities and the information they need; high quality testing based upon the various comprehensive scenarios for quality assurance. The following OPTIMIS actors are identified:

- Service Provider (SP): Provides a service/application to the Customer.
- End User / Consumer: Entity (person or process) who is actually using the service provided by the SP.
- Infrastructure Provider (IP): The provider of the infrastructure resources, in IaaS fashion. It provides easy access to computing, storage and networking resources, according to the request from the SP to deploy and execute the application.
- Customer (C): Organization who makes the contract with the SP, in order to request a service (or application).
- Broker (B): Sits between the infrastructure provider and the service provider. In addition to providing the abstraction layer of the individual IPs and their heterogeneous interfaces to the SPs, it provides service arbitrage, a matching engine, aggregation, and intermediation. The Broker can also offer other value-added services between the IP and the SP, such as benchmarking tools, performance prediction (based on monitoring historical data), security, SLA negotiation, among others. In a general case, multiple IPs play into the game (1-N relationship), such as in multi-cloud scenarios, but the Broker can also offer its services in the case of a 1-1 relationship between SP and IP.

Requirements from High-Level Objectives

A goal-oriented methodology has been used to elicit the functional requirements since it allows a clear separation of concerns between the requirements by relating them to distinct objectives. The methodology is based on the progressive refinements of abstract and general objectives into more concrete goals and requirements that lead to the identification of the main entities with their relationships and of the main agents responsible for the satisfaction of the requirements. The methodology is based on the following models:

Goal model, capturing higher level objectives and more operational ones, connected by refinement links. Figure 1 provides a summary of the global goal model, its underlying properties, and the operational measures to satisfy these properties.

Object model, giving a structured view of the pertinent vocabulary necessary to express the goals. This model is expressed using the classical UML class diagram with entities, relationships, attributes, cardinalities, etc.

Agent model, gives a view of all the requirements under the responsibility of each agent. This means that each agent instance in the system (end-user, SP, IP) is responsible for enforcing the requirement instance assigned to it. This model also shows the structure of the interaction among agents in order to cooperate to achieve a higher level goal.

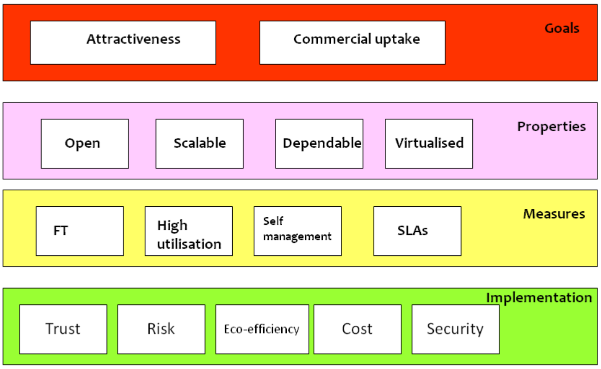

Figure 1 Simplified View of OPTIMIS Objectives and Operations.

To reinforce the attractiveness and the commercial uptake of the cloud services, several issues need to be addressed such as openness, scalability, dependability, and virtualization. For end-users the operation of cloud services should occur transparently, i.e. without undue technical overheads. End-users must feel confident that the Quality of Service (QoS) they require will be met in the fulfillment of the contracts passed with the SP, which have the same TREC requirements regarding the contracts passed with IPs. SPs want to determine the best possible way to deploy their services. IPs, on their side, want to determine an optimal allocation of their cloud resources. To this end, they need an effective monitoring of the workload and a QoS based scheduling. Also, the design of suitable business models that define the relationships between the end-users, SPs, and IPs are necessary so that demand and offering of cloud services meet in the most efficient way.

These issues are supported by measures like the use of SLAs, self management, high utilization of resources, and fault-tolerance mechanisms. The use of SLAs is a prerequisite and a building block for the wider adoption of the cloud services. Self management, cost effectiveness, high utilization, and fault-tolerance all refer to supporting processes that the SPs and IPs will set up in order to meet the cloud services demand in the most profitable way. The IP has to monitor its resources, to act before SLA violations, and to manage its infrastructure to provide QoS. Most of the time, the IP will favor the highest possible utilization rate and if necessary use fault-tolerance mechanisms to ensure service operation.

The goal of the SP is to find the best possible IP to run the service (according to trust, risk, eco-efficiency, and cost goals) whereas the IP attempts to maximize its internal business goals (guided by revenue as well as other trust, risk, eco-efficiency, and cost goals).

The business requirements (primarily non-functional) need to be translated into and implemented as technical functions. These requirements come from interviews and conversations with a variety of stakeholders in the cloud industry, supplemented by additional primary research in the context of the work carried on business issues in clouds, and are grouped as follows [OPTIMIS2010b, OPTIMIS2011a]: 1) flexibility and agility; 2) value added services, trust and control, and 3) cost efficiency. From the legal perspective, at the outset the requirements are split into three main aspects [OPTIMIS2011b]: 1) data protection and data security; 2) intellectual property rights, and; 3) green legislation issues.

Requirements from Use Cases

The studied use cases represent different scenarios in which OPTIMIS results are applicable, and have impact in previously identified roles. The main beneficiaries of the results of the project are the following. A *Cloud Programming* use cases that will demonstrate the OPTIMIS Toolkit applicability for SPs both for the construction of new services but also to enable existing legacy applications to take advantage of the cloud deployment. A *Cloud Brokerage* scenario will demonstrate the OPTIMIS toolkit applicability for cloud Brokers or Aggregators as well as SPs. This use case focuses in the demonstration of the Multi-Cloud scenario. Finally, a *Cloud Bursting* use case will demonstrate the OPTIMIS toolkit applicability for final users, with the interaction between internal and external clouds, and for infrastructure providers that rely on third parties to offer their services. This use case focuses in the demonstration of the federated cloud scenario.

Architecture

The following description of the OPTIMIS Toolkit is focused around the challenges outlined in the introduction. The tools for self-assessment of non-functional criteria such as trust, risk, eco-efficiency, and cost are outlined first as these are used in all management decision making. To highlight the importance of the OPTIMIS Toolkit for service lifecycle optimization, next follows a description of toolkit components for service construction, service deployment, and service operation, respectively. The last part of the section is an overview of the challenges associated with securing multiple connected clouds and the technologies applied to address them.

Tools for Self-assessment of Trust, Risk, Eco-Efficiency, and Cost

As discussed earlier, one of the overall objectives of the OPTIMIS Toolkit is to provide self-management capabilities for cloud infrastructures and services based on functional as well as non-functional criteria. The following describes how SPs and IPs assess aspects of TREC during service deployment and operation. A SP is responsible for matching the end-user requirements with the correct IPs to ensure the required demand is met. To achieve this, the SP service deployment process uses the TREC tools to rank IPs accordingly. On the IP side, TREC factors increase the performance and quality of an IP. For example, the estimated risk for an SLA violation supports the IP's decision of agreeing an SLA as well as of initiating fault-tolerance mechanisms to prevent SLA violation [DAKM2011]. Thus, TREC tools improve the IP's reliability and QoS. In the following, the TREC tools are described.

Trust. The main function of the Trust Framework tool is to assess the trust associated with SPs and IPs. This trust assessment involves the following: 1) *Main Trust*: this value is calculated using information from the monitoring and the SLA system of OPTIMIS plus an internal social network generated within the framework to manage the relationship between the actors; 2) *Social Network Trust*: which is calculated using the information stored in the social networking tool only, and 3) *Self Experience Trust*: this value is calculated using the SP's experience only.

The trust framework includes the following components: 1*) Social Networking Tools*: to interconnect all the actors making use of the Trust Framework. Thus, it creates links between end-users, SPs and IPs with an associated satisfaction rate. This rated interconnection gives the possibility to assess the trust in relation to one of the actors based on the opinion of the end-users who have interacted in the past with the SP or IP; 2) *Trust Framework Core*: supports the business logic of the Trust Framework, and is comprised of: a) data Asset: which stores the information obtained from the monitoring system and the social networking tools, as well as the historical information about the actor ranked, and b) Trust Engine: where all the logic to generate the reputation of the actors is, and 3) *Monitoring*: a set of monitors that collect data from different sources, e.g. performance of a service on the SP side, SLA monitoring, scalability rules on the IP side.

Risk. To support the risk assessment functionality on the SP side, a number of components are identified: a *Confidence Service* (comprising a *Risk Assessor* and a *Provider Assessor*), a *Risk Inventory* and a *Historical Database* for recording past SLA transactions. The confidence service will take into account the various risks of working with the different IPs accessing the providers. This is used in the service deployment process to make decisions based on a stored database of history of working with the different IPs and the risk inventory associated with the different assets involved. The Risk Assessor provides the functionality to evaluate the risk associated with an offer returned by the IP's Admission Controller (AC), by determining the reliability of the offered Probability of Failure value from the IP's SLA quote [Djemame2011]. The Provider Assessor verifies the expected integrity of an IP's guarantees when they make any SLA offer through an implementation of Dempster-Shafer Analytical Hierarchy Process (DS-AHP), whereby each decision alternative (in this case each provider) is mapped onto a belief and plausibility interval [Huan2008]. A risk inventory is a simple database of risks associated with each asset, their vulnerabilities and threats. This also contains risk mitigation strategies following risk assessment.

Performing risk assessment at the IP level increases the performance and quality of the IP. Thus, the IP also assesses the service to be deployed by determining an estimated risk if it were to accept the SLA taking into account fault tolerance mechanisms and actions following an SLA violation, in turn improving the IP's reliability and QoS. Four categories of risk are identified: *technical, policy, general*, and *legal*.

The IP would forward the *service manifest* request to the AC to evaluate the feasibility of admitting the new service, with respect to current infrastructure load, predicted future capacity, as well as risk. This helps the IP to determine where to place the Virtual Machines (VMs) by combining its local management policy with the function and non-functional requirements. The *Cloud Optimizer* (CO) also supports risk assessment at the service operation to ensure the SLA fulfillment. In addition to *a risk assessor, a risk inventory, and a historical database, a consultant service* component is identified at the IP side, which uses data mining tools on the history of events of running similar services or working with the same SP to estimate risk. The consultant service can also have access to all the monitoring information keeping the IP on track with the changes. This data can be static or dynamic in nature about its resources and the current service execution. Examples of such information are the current workload, system outages, temporary performance shortages, monitored network traffic, experts' availability, or general information regarding the number of services to operate. The monitored data helps to determine bottlenecks in the IP’s infrastructure so that the provider can improve its capacity planning, administration, and management of its resources. This leads to high, cost-effective productivity of virtualized resources. Some of the risk models are probabilistic, possibilistic, or hybrid.

Eco-Efficiency. This tool is in charge of assessing energy-related aspects, such as energy efficiency and carbon emission levels, in a given cloud infrastructure. In this sense, it uses the information collected by the respective monitoring tool of the basic toolkit. This assessment is fully guided by ecological-aware rules defined according to the energy-related parameters, e.g., carbon emission levels, specified in service manifest. Once this assessment is performed, it is used by self-management policies that act in accordance to provider’s energy requirements and BLOs. Between the possible management actions we find the consolidation and migration of VMs where services run. The tool is composed of the following components: 1) Eco-efficiency assessor: is the core sub-component and is the responsible for performing the aforesaid assessments and predictions. The assessments in this component result in action requests to be performed by the VM Manager, such as, requests for migrating a VM, or consolidating several VMs in a single node; 2) Energy efficiency monitoring: contains the information collected by the respective monitoring tool and is used by the assessor, and 3) Eco-efficiency rules: where rules are defined according to the energy-related parameters specified in services’ manifest, and which fully guide the assessments.

Cost. This tool is responsible for assessing and predicting the relevant costs for service deployment and operation from both the SP and IP perspectives. The module can be deployed independently by both the SP and IP. It is used by these actors to evaluate and predict their costs and subsequently to optimize their profitability. The tool is composed of three main components: 1) Cost Assessor: is responsible for managing all operations within the cost tool, and is used to accurately trace the absolute costs of service operation and deployment. In addition the Cost Assessor acts as a prediction engine to assess various scenarios, for example how much would it cost to host a service with specific SLA requirements e.g.

CPU, Disk, Memory etc. The cost assessor may also recommend the most cost effective policy to employ when required to violate SLAs; 2) Cost Modeler: is responsible for defining the mathematical models to accurately record and predict the cost of service deployment and/or operation. To build these models the modeler utilizes statistical techniques such as data mining on the data stored within the cost database. For the SP this may include data regarding hosting charges (from IP), service usage, SLA levels etc. For the IP this may include hosting income (from SP), service usage, SLA levels, infrastructure utilization, energy usage etc; and 3) Cost Database: acts as persistent store for all previous cost analyses (predicted / realized) and relevant cost data which has been gathered from the monitoring and SLA/SLO managers. This is subsequently used for the definition and refinement of the cost models.

Service Construction

This section describes the different aspects of defining a service with for an OPTIMIS IP. We give an overview of the service manifest, which describes a service to be deployed. Moreover, we discuss how software licenses are managed in the OPTIMIS environment.

Service Manifest

The OPTIMIS manifest is the global description of a service that is delivered by an OPTIMIS IP. It is generated by the OPTIMIS SP during service construction phase and consumed by the different IP components during the service negotiation and instantiation phases. The manifest basically consists of four sections: (i) the service description section, (ii) the TREC sections, (iii) the elasticity section, and (iv) the data protection section. Figure 2 outlines the service manifest structure.

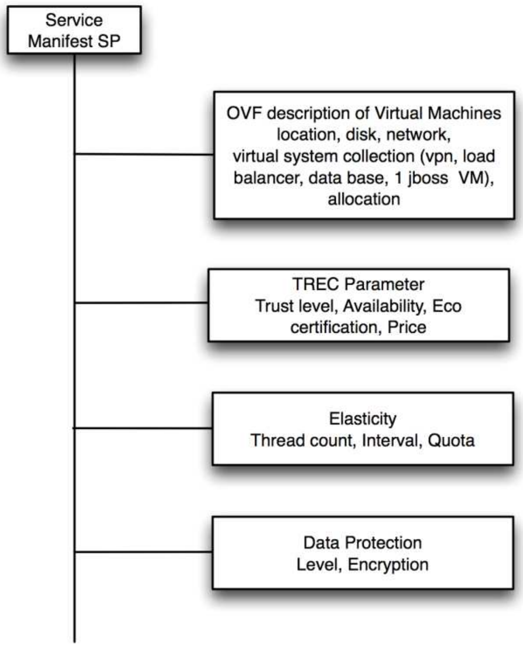
Service Manifest SP
OVF description of Virtual Machines location, disk, network, virtual system collection (vpn, load balancer, data base, 1 jboss VM), allocation
TREC Parameter Trust level, Availability, Eco certification, Price
Elasticity Thread count, Interval, Quota
Data Protection Level, Encryption

Figure 2 Overview of the service manifest structure.

The service description section is a functional description of the cloud service to provide. It consists of two parts, an abstract service description and an allocation section. Since the infrastructure resources are provided in form of VMs, the Open Virtualization Format (OVF) is used for describing these. The OVF document defines a collection of virtual systems, where each virtual system corresponds to one service component and is uniquely identified by its OVF id. In addition, the allocation section defines a set of allocation constraints such as minimum and maximum instances of each virtual system (service component). The service description section is complemented by a TREC section. The TREC section defines non-functional parameters of the service, i.e., requirements related to the energy efficiency, reliability, and availability of the allocated resources. Within the elasticity section of the service manifest the SP can specify a set of elasticity rules, i.e., in order to start up a new VM when a defined system load is reached respectively to shut down a virtual system instance when system load falls below a given threshold. Finally, the SP can specify constraints related to data placement in the data protection section, e.g., it is possible for the SP to specify a list of countries where service data may (or may not) be placed.

License Management

Access to licenses for authorizing the execution of an application in a cloud located beyond the administrative domain of the site running the license server usually leads to applications aborting during startup because of unreachable license servers, e.g. due to firewall issues or network address mismatch. In OPTIMIS we address these problems with a prototype for licensing and license management developed in the European project SmartLM [SmartLM2010], which provides licensing technology for location independent application execution. Trough decoupling of local authorization for license usage from authorization for remote application execution, SmartLM provides the necessary mechanism for transparent enabling license-protected applications in a cloud environment.

Software tokens are created when usage of a particular software license is granted to a user upon request by the local license management system. The tokens carry the authorization information based on the user's request, locally defined policies, and availability of a license or feature for running an application. As software artifacts the tokens are mobile thus providing the necessary flexibility for licenses to follow applications into clouds.

At deployment time the VM Contextualizer retrieves and embeds license tokens into the VM hosting the application. No communication between the application and the license server that issued the token is required at runtime.

In the future, OPTIMIS will extend the SmartLM solution by providing: (i) dynamic deployment of a trusted instance managing a number of tokens for one or more applications and (ii) dynamic deployment of a full license service with a subset of the licenses available in the users' home organisation. The configuration of the dynamically deployed license service will be managed by the VM Contextualizer. This approach is especially useful when the same cloud is used to run license protected applications for long time. In the first approach the contextualization tools are responsible for configuring and deploying the trusted instance for the respective network environment and to transfer tokens.

Service Deployment

After service construction, deployment of the service to an IP is assisted by the *Service Deployer* (SD). The SD offers a unified interface for service deployment and management, encapsulating the internal component interactions triggered as a part of deployment. This process is illustrated in Figure 3, and the below list presents the general steps of service deployment from the perspective of the SD, although the exact interactions may vary slightly between different deployment scenarios (multicloud, bursting, etc.).

1. Prior to deployment, the SD is configured with properties relevant to the different scenarios, e.g. the endpoints of several IPs in the multi-cloud scenario. The deployment process is initiated by passing the service manifest to the SD.
2. The SD depends on an internal component, the *Service Deployment Optimizer* (SDO) to rank existing deployment alternatives according to the needs of the service. The SDO negotiates with QoS components of different IPs, and ranks the alternatives based on, e.g., the TREC factors.
3. When a suitable IP for deployment has been found, the SD triggers *contextualization* (configuration) of the VMs, and uploads the VM images to the repository of the chosen IP.
4. Once the data transfer is completed, execution of the service is initiated by calling the local *Service Manager* component, which is responsible for managing the service in the operation phase.

Each general step outlined above triggers a series of underlying interactions not visible to the SD. For instance, in the IP, the QoS component relies on an Admission Control component in the negotiation process, encapsulating the potentially complex statistical models and state management of the infrastructure. The following sections describe the inner workings of the major supportive components in more detail.

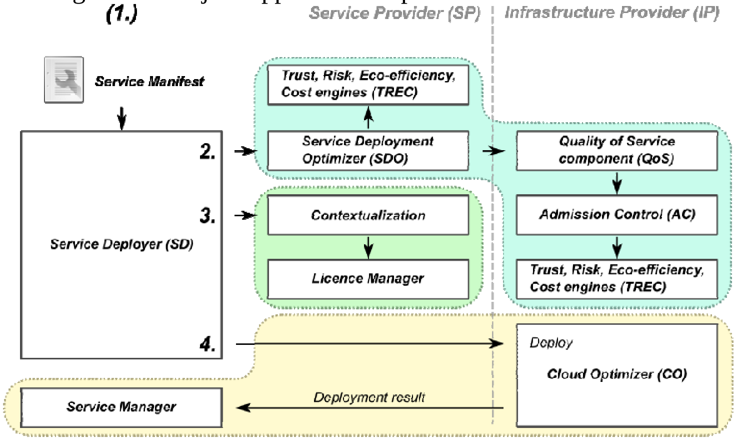
(1.)
Service Provider (SP)
Infrastructure Provider (IP)
Service Manifest
Trust, Risk, Eco-efficiency, Cost engines (TREC)
2.
Service Deployment Optimizer (SDO)
Quality of Service component (QoS)
3.
Contextualization
Admission Control (AC)
Service Deployer (SD)
Licence Manager
Trust, Risk, Eco-efficiency, Cost engines (TREC)
4.
Deploy
Cloud Optimizer (CO)
Service Manager
Deployment result

Figure 3. The general steps of service deployment. The deployment is initiated with the Service Manifest (1). A suitable placement is found by contacting the

SDO (2). The raw VM image is contextualized with the necessary information (3), before the service is finally deployed (4).

Quality of Service

Current cloud environments are offered to their customers in a best effort approach. Instead of guarantees a statistical uptime expectation is communicated to the user with minimal compensations in case of unexpected downtime. In contrast, in OPTIMIS, SPs can rely on SLAs with IPs, e.g., to extend their own resources dynamically with cloud resources in case of peak demands of their customers. These SLAs cover aspects like TREC factors, security, legal requirements for data-placement, etc.

The OPTIMIS QoS SLAs are based on the WS-Agreement for Java (WSAG4J) implementation (see Figure 4). WSAG4J [WSAG4J2011] is a full implementation of the WS-Agreement standard of the Open Grid Forum [WS-Agreement2007]; it provides comprehensive support for common SLA management tasks such as SLA template management, SLA negotiation and creation, as well as SLA monitoring and accounting. The OPTIMIS QoS component interacts with the following three components of the OPTIMIS toolkit:

- Admission Control: In order to negotiate and create SLAs with a service consumer (the OPTIMIS SP), the service provider (the OPTIMIS IP) must assert that the agreed service can be provided. When a SLA is created with the service consumer, the SLA management component of the service provider initiates the service deployment process.
- Service Monitoring: Each provider that offers SLAs for a particular service should perform SLA monitoring in order to detect possible SLA violations and to take appropriate counter measures. The SLA monitoring component therefore integrates with the Service Monitoring, which provides the required monitoring data to assess the fulfillment of the service level objectives defined in a SLA. SLA monitoring is a continuous process during the lifetime of an SLA and is performed on a regular basis. The SLA guarantee evaluation results in corresponding events, which are then processed by other components, such as SLA accounting (penalties and rewards), the TREC toolkit, or the Cloud Optimizer.
- Cloud Optimizer: The Cloud Optimizer (CO) consumes events generated by the SLA monitoring of the IP. According to the event type and its importance, the CO takes appropriate measures in order to fulfill the SLA guarantees. These measures may include

starting up or shutting down a VM upon events from the Elasticity Engine, or to migrate or restart a VM.

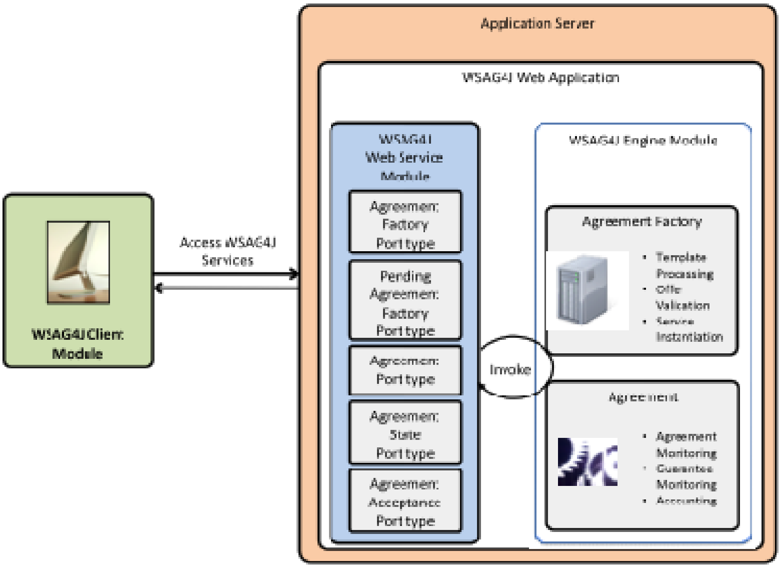

Figure 4. WSAG4J components.

Admission Control

Admission Control (AC) in OPTIMIS has the main goal of deciding whether a new service can be admitted by the IP, and deciding what the optimum allocation of the VMs of the service to the available underlying resources is, in case of acceptance. This task is far from trivial, since in many cases, admitting a new service increases the risk of already deployed ones failing. Conversely, a strict acceptance policy results in increased rejections, reduced utilization, and thus lost revenue.

Hence, in order to come to a decision, AC must weight critical factors such as the current workload of the infrastructure, the requirements (in terms of computing, memory, storage and networking resources) of the new service, and the IP’s business policies in terms of risk, trust, eco-efficiency, and cost. For example, in some cases it may be more lucrative to admit a highly profitable service even under the risk of jeopardizing existing, but less profitable, ones. Likewise, given a very sensitive IP in terms of eco-efficiency and two services competing for the same resources, the less profitable service may be accepted if it is sufficiently more eco-efficient.

In the context of OPTIMIS, an IP is a business entity that establishes SLAs with SPs to allow for booking a set of virtualized resources for hosting distributed services, whose components are encapsulated inside VMs. According to the expected usage of the service, the SP specifies a lower and an upper limit for the number of VMs (called basic and elastic VMs respectably) that may be required during runtime for each service component. At the first activation of the service, only the basic VMs are participating in the service. During runtime, elastic VMs may be added to the service according to the workload and the SP's policy. This number of additional elastic VMs cannot exceed the upper limit defined in the SLA.

In finding the optimal allocation, AC makes use of an optimization model that builds upon factors such as the performance requirements of the services as defined in the SLA, and information about the current workload status of the infrastructure's resources acquired by communicating with OPTIMIS Monitoring system. Furthermore, the AC also accounts for business rules, expressed in TREC terms that are provided to the AC by the IP. These factors are mathematically modeled in terms of constraints and an objective function to be optimized.

In addition, differently from the traditional admission control mechanisms, the OPTIMIS AC builds on a probabilistic basis. This is due to the fact that the number of elastic VMs may be quite large compared to the basic requirements for the service. For example given a service with a high variation in the number of users, the number of elastic VMs that may be required to be added at runtime can be many times more than the basic VMs. Therefore the number of elastic VMs plays a significant role in the total requirements and the cost for hosting a service. Thus, the IP has a strong interest in investigating the possibility of narrowing the resources that need to be booked for elasticity reasons when accepting a service. At the same time, such an approach may increase the possibility of deviating from the agreed QoS level, i.e., the required availability of the resources, and the imposed penalties may as well outgain the advantages of this approach.

With dynamic elasticity of the number of VMs of a service in focus, OPTIMIS AC tackles the aforementioned problem of optimum allocation of such services on virtualized physical resources, by incorporating a probabilistic approach in terms of availability guarantees (i.e. minimum probability of finding the required number of VMs available when actually needed). The proposed approach relies on the actual probabilities of requiring extra elastic VMs for the services, which are incorporated into the admission control test, allowing a reduction of the number of physical resources that need to be booked for elasticity reasons. The output of the optimization problem is the number of accepted elastic VMs per service component as well as the location (host) of each VM that comprise the service (both basic and elastic ones). The resulting optimization problem constitutes an admission control test that may be run by the IP. Optimal allocation of distributed services under QoS constraints across multiple resources is an NP-hard problem. For modeling and solving this problem, the AC uses the General Algebraic Modeling System [GAMS].

Contextualization

The VM Contextualizer uses data gathered by the SD to set the context of a VM at runtime. This involves the SD assembling data from several sources and providing it as input to the public interface of the VM Contextualizer along with the ID of the base image to contextualize. The Data Manager is used to propagate VM images to the VM Contextualizer. Once the VM image has been propagated, it is mounted and modified to include scripts that interact with the guest operating system at boot time, preparing it to receive context in a reusable fashion. After the image has been prepared, it is returned to the VM Image Repository for storage and used later by the VM Manager. The lightweight scripts embedded in a VM image bootstrap to additional scripts held within an ISO CD ROM image, as per the OVF recommendation [OVF2011], mounted within the guest operating system. At run time the bootstrapped scripts access contextualization data stored within the ISO CD ROM image, giving a VM an identity. The inclusion of the ISO image as a mechanism to store contextualization information provides a facility to separate the contextualization data from the VM image. This removes the need to create unique VM images for each VM to be contextualized, while also improving the security of the contextualization process as security certificates and keys are not stored in the VM image itself, but rather in the dynamically generated ISO image.

The VM Contextualizer uses QEMU [QMU2011], a generic and open source machine emulator and virtualizer to manipulate images. The QEMU tool ``qemu-img" enables the conversion of VM images. In addition, Linux system tools (such as ``mount", ``iosetup" and ``kpartx") are used to mount VM images as loop devices for write manipulation. For the creation of ISO images, the Linux system tool ``mkisofs" is used in addition to the previously mentioned tools to create and modify ISO images. The VM Contextualizer interacts with other components in the OPTIMIS Toolkit at the SP level during image-level contextualization. The SD initializes the preparation of a VM image for receiving context data at runtime. When the VM Contextualizer receives a request to prepare a VM image, it retrieves the image from a VM image repository. After manipulating the image, it is sent back to the repository for safekeeping.

Figure 5 illustrates the instance-level contextualization process of a VM at the beginning of its execution. During the boot sequence of a VM the contextualization tools mount an ISO CD image that contains contextualization data and scripts to process this data into a useable form. These scripts communicate through a known interface to the OS specific contextualization scripts embedded in the VM image. These OS specific scripts manipulate configuration files of associated OPTIMIS components and software dependencies that support the service, setting their context. If a component or software dependencies requires continuous update to its context, the scripts can remain in a daemon-like mode.

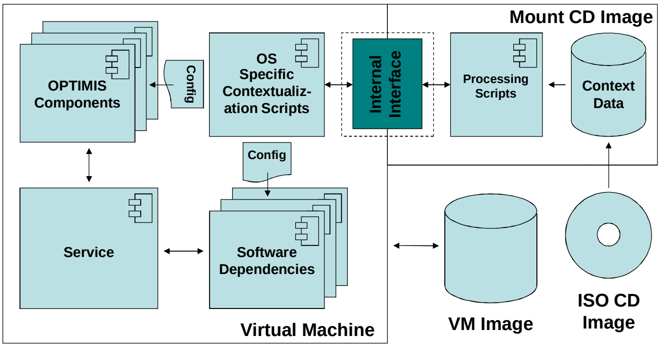

Figure 5. Interaction between VM image and ISO image at run time.

Service operation

An OPTIMIS IP makes use of several internal management procedures during service operation to supervise and fine-tune the execution of the service and optimize the utilization of the infrastructure. The relation between different components is described in Figure 6, and the following sections describe some of these internal procedures in more detail.

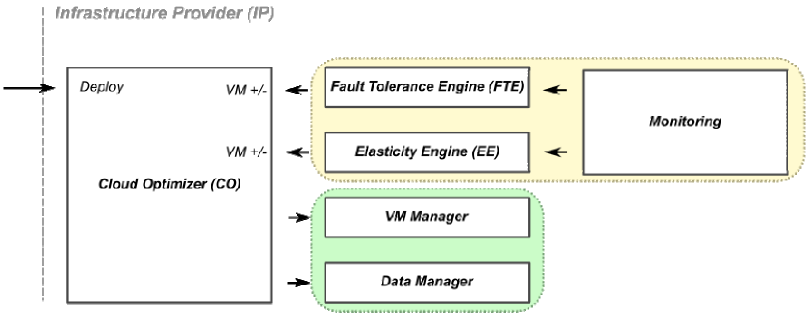

Figure 6. The CO finalizes service deployment and is the central component for service operation. Monitoring data is consumed by the FTE and EE, and may result in a change in the number of allocated VMs per service, which is

managed by the CO with the support of the VM Manager and Data Manager components.

Monitoring

Deployment, execution, and management of applications in highly dynamic infrastructures such as clouds introduce a whole new set of requirements in respect of monitoring that must be addressed by developers and providers of the related services. The introduction of the virtualization layer along with the energy efficiency directives applied into the data center operation field, have increased the amount of data that must be collected and processed (i.e., further parameters must be measured and monitored, such as e.g., the amount of memory used by a concrete VM or the CO2 emission of a datacenter. Additional parameters are discussed in [Katsaros2011]). Furthermore, the elastic character of the infrastructure, essential in a cloud environment, requires that the monitoring tool is able to handle that a service or infrastructure scales up or down dynamically. Apart from scalability, such system should adopt the service-oriented design pattern, which is the keystone of the cloud computing paradigm. The existence of multiple layers (physical infrastructure, virtual environment, application), each one ultimately associated and dependent on the other, results into the need of collecting and aggregating all information towards an effective decision taking mechanism. Furthermore, the storage of these data must be performed in an efficient way so that they can be reused by other components of the OPTIMIS Toolkit such as performance estimation mechanisms.

As far as the monitoring infrastructure concerns, the proposed architectural model incorporates the conceptual *Information Provider* layer (Figure 7). By that we mean the different sources from where monitoring data are being collected. As Information Providers we consider the main entities producing information: the physical infrastructure, the virtual infrastructure, the application specific metrics, and the energy efficiency measurements. The actual collection of information is performed by respective components known as *Collectors*. We have designed our solution in a way that this layer is extendable, allowing the incorporation of additional sources through corresponding collectors. This introduces a first degree of scalability (in the sense of extendibility), which enables the monitoring system to handle a growing number and type of monitoring parameters. Data collection is realized either with a push model where the Collectors gather data and pushes a monitoring report towards *Aggregator* components in the higher conceptual layer, a corresponding pull model, or a combination of the two. More details about the operation and the implementation of the monitoring collection mechanism can be found in [Katsaros2011].

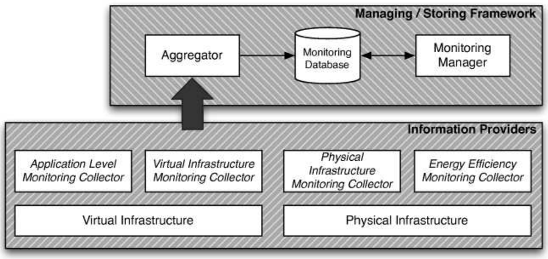
Managing / Storing Framework
Aggregator
Monitoring Database
Monitoring Manager
Information Providers
Application Level Monitoring Collector
Virtual Infrastructure Monitoring Collector
Physical Infrastructure Monitoring Collector
Energy Efficiency Monitoring Collector
Virtual Infrastructure
Physical Infrastructure

Figure 7. Monitoring infrastructure overview.

The second hierarchical level of the proposed approach is the *Managing and Storing Framework*. This layer includes the components in charge of managing the data collected from the monitoring information providers and storing them in an aggregated way in a monitoring database. The aggregation is done in terms of data model consistency, which is designed in a way that eases both the process of storing the information and the subsequent retrieval of those data for analysis and usage. Each monitoring report must thus contain an identifier that relates it to the source of information. This is based on the concept of entities (IP, SP, service manifest, service) and the relationship between them. This contributes to a second level of scalability. As presented in [Katsaros2011], the collected data is classified and aggregated according to those entities, which results in a number of database records, following the schema of the designed data model. The structure of the database conforms to best-practices for database scalability, but a discussion of these is out of the scope for the work presented here.

The *Monitoring Manager* component serves as the orchestrator of the whole monitoring process. The role of this component is twofold and includes the capabilities of controlling the monitoring process (start/stop actions) and providing the necessary interfaces towards other components of the cloud ecosystem as well as to other external consumers of monitoring information (e.g. graphical user interfaces, administrators etc.). The collected and aggregated data can be provided in two different ways: on the one hand, the most recent monitoring information is provided in an on-the-fly manner and, on the other hand, the monitoring data are being stored in a local database with historical information of the deployments and executions. Notably, the monitoring components do not provide event processing capabilities themselves, but the *Monitoring Manager* provides the necessary interfaces for the TREC self-assessment tools to acquire and analyze the required data and generate the corresponding events, if needed.

A third level of scalability is introduced by the general architecture design and the concept of the OPTIMIS Toolkit. By this, some of the toolkit components are decentralized, being part of the SP domain, while others are located in the IPs [OPTIMIS2011c].

Elasticity Engine

One of the most prominent features of cloud computing is the ability to quickly increase or decrease the amount of resources allocated to a service, either by modifying the allocation to each running VM (*vertical elasticity*) or by modifying the number of copies of VMs running on the infrastructure (*horizontal elasticity)*.

Several different models and approaches can be applied to elasticity. Ali-Eldin et al. [Ali-Eldin2011] present a queuing theory based approach with different combinations of reactive and proactive models for scaling up or down services. Gambi et al. [Gambi2010] employ mathematical surrogate models to protect SLAs though the means of elasticity. Elasticity support in the OPTIMIS framework is designed to be agnostic to the actual elasticity decision algorithm used, creating a software base upon which several different models can be used. The *Elasticity Engine* (EE) component contains a framework for interaction with supportive components, and an easy to modify module containing the actual model for elasticity. The primary interactions of the EE are with the CO (to start or stop VMs) and the monitoring system (to gather runtime status) and these interactions are isolated from the internal model for elasticity. The EE is also designed to be a stand-alone tool usable on the SP side, in which case the SP itself replaces the interactions between the EE and the CO. Even though the below description focuses more on IP-driven elasticity, the process in the SP side is similar.

The elasticity process is initiated by the CO on a per-service basis, supplying the EE with a set of *elasticity profiles* (or elasticity rules [Rodero-Merino2010]). The rules specifies e.g., the values, or *Key Performance Indicators* (KPIs) to monitor along with the relation between the KPI and the amount of VM resources (either size of each VM or number of instances) granted to the service. Updating the elasticity rules of a service is one way of adapting the service to different objectives. Note that the exact interpretation of elasticity rules and the manner in which they are enacted is different depending on the internal model used, but the different interpretations do not affect the general management procedure.

Once the process is initiated, the EE collects data on the relevant KPIs by communicating with the Monitoring system. The received data is an input to the internal model in the EE, which returns an estimation of the required amount of resources for this particular service. The estimation is passed along to the CO, which is responsible for the process of actually adjusting the amount of resources dedicated to the service. Notably, the EE requests may be ignored (likely resulting in SLA violations) if the CO policy deems this to be more aligned with the overall management goals of the IP.

Cloud Optimizer

The CO combines the Monitoring system and the fundamental TREC assessment tools with a set of management engines to perform cloud infrastructure self-management governed by BLOs of the IP. Notably, goals in these higher-level objectives (policies) typically differ from those in the SLAs established with SPs, thus self-management decisions become tradeoffs between the BLOs and SLAs.

The CO is also responsible for tracking the relationship between services and the associated set of VMs where they are running. The CO updates this matching between services and VMs to the Service Manager when a new service is deployed, a VM is added or removed (due to elasticity), or a VM is restarted upon a fault. Upon deployment of a new service, the CO notifies the EE and Fault Tolerance Engine (FTE), the latter a self-management component responsible for ensuring robust VM provisioning.

In addition to communication with these engines, the CO interfaces with the VM Manager (based on Emotive Cloud [Emotive2011]) and Data Manager, two high-level actuators for handling of VMs and Data, respectively. The Data Manager includes, in addition to data transfer capabilities, a framework for handling service data protection issues, e.g., performing storage location selection based on legal deployment constraints. The CO also contains a policy repository that enables IP administrators to change or reconfigure policies for how fault tolerance, admission control, VM management, and data placement should be performed. In summary, these policies in the CO determine the most suitable management actions of the OPTIMIS components in order to meet the IP’s BLOs.

Service Manager

The Service Manager (SM) component serves as a management point for services during deployment and operation. In essence, it keeps the necessary information about a service: details about the selected IP where the service is being deployed, how many and which VMs it uses as well as the Internet Protocol addresses of the VMs. The SM component is a part of the SP’s tools of the cloud framework but it also provides external interfaces towards the IP level. The SM interacts with the following components:

Service Deployer (SD): during deployment a set of service state is being created within the SM for every service, including all the above-mentioned information. After the SD acquires the initial set of information during the service negotiation process, it triggers creation of service state data in the SM.

Elasticity Engine (when used at SP level): the SM initiates the EE for a service, each time a new service resource is being created. Thereafter, the EE evaluates the data received by the monitoring infrastructure and performs elasticity actions accordingly.

Cloud Optimizer (IP level): in case a service management action is taken (e.g. additional VMs are added for a service), the CO notifies the SM of the related change on the service deployment. Thereafter, the adequate service state data is updated.

Inter-Cloud Security Framework

Of the various high-level concerns being addressed by the OPTIMIS project, a major concern is the provisioning of a secure communication framework to the services utilizing the resources of different cloud IPs. The usage pattern of these services should be quite flexible i.e. on one hand they might be directly accessed by end-users or on the other hand they might be orchestrated by other SPs for their customers. The presence of the multiple IPs in the cloud computing ecosystem is a key issue that needs to be addressed by any inter-cloud security solution. In addition to the IPs, the other three main actors in the cloud ecosystem are SPs, end-users, and cloud brokers. For the numerous interaction possibilities among these actors, whatever the usage scenarios may be, the security of data and communication between the service consumers and its multiple providers is of paramount importance.

In the light of the previous discussion, it is clear that an inter-cloud security solution is highly desirable that would provide a framework enabling seamless and secure communication between the actors of a cloud ecosystem over multiple cloud platforms. Such a solution, however, has a number of challenges associated with it because of architectural limitations, as most of the current cloud service platforms and the multi-tenants environments they offer make it difficult to give the consumers of their services flexible and scalable control over the core security aspects of their services like encryption, communication isolation and key management, etc. Secure communication is also challenged by lack of dynamic network configurability in most cloud providers, caused by the inherent limitations of the fixed network architectures offered by these providers. We address the secure, flexible and scalable communication concerns that in our view must be overcome in order to provision holistic cloud services to consumers from multiple cloud service providers. We present the architecture and design of an inter-cloud secure communication framework that offers the features of dynamic and scalable virtual network formation, efficient and scalable key management and optimized peer discovery etc. all on top of secure and private communication between the consumers of the service across multiple cloud platforms.

Our solution provides a single virtual network to the consumers of cloud services using the infrastructure and resources from multiple cloud providers and offers all the management capabilities to efficiently and transparently run services on top of this network while catering for the dynamic growth and shrinkage of the network and its participant entities. At its core, it provides the ability to automatically establish peer-to-peer overlay networks comprising of the VMs and other infrastructure resources constituting a cloud service. Using the same P2P techniques, we also offer a distributed key management service which facilitates the automatic discovery of the peers participating in a service and the binding of cryptographic constructs like keys, certificates and fingerprints to their identities. Several key management models exist currently that can be utilized for our purposes, ranging from centralized key distribution centers to those based on multicast communication [Qui2006] or distributed hash tables [Rafaeli2003]. The Inter-Cloud VPN (ICVPN) architecture consists of the following main components:

Peer-to-Peer Overlay. The core technique employed by the ICVPN is the secure tunneling of network traffic over a scalable, resilient and self-managing peer-to-peer overlay with minimal manual configuration and administration. This approach acts as an aggregation service for the peered resources (which in this case are VMs) spanning across multiple cloud domains to help form a virtual private network. The overlay can be further leveraged to customize the virtual topology of the network, by either emulating a completely distributed mesh topology at one end, or a centralized star topology by automatically electing a master peer to act as the gateway of the virtual network at the other end, or an in-between hybrid topology using a number of selected super-peer nodes. The adaptation of a suitable network topology is indeed a major communication benefit for services using P2P overlays for their operational requirements [Dinger2005]. Figure 8 shows the presence of all the above mentioned topologies in the three cloud infrastructure providers.

In our use case scenario, the end-users or the SPs are responsible for provisioning VMs from cloud providers to deploy and run their services. These VMs can be considered as the peers of the overlay network if they want to join the virtual private network and the addition of a peer to the overlay network is handled by a P2P client embedded in the virtual appliance image used to instantiate a VMs on a cloud platform. Upon joining an overlay network to become part of a service, a peer starts the process of creating secure tunneled connections to the other peers of the service according to the network topology selected for the operation of that particular service. The typical constructs used for this purpose are TAP and TUN devices, which are virtual network kernel devices set in the operating system of the peer node [VTUN]. A TAP device simulates an Ethernet interface card and operates with link layer datagrams such as Ethernet frames. A TUN device simulates a network layer device and operates with network layer packets such as IP packets. TAP is mainly used to create a virtual network bridge, while TUN is used with routing, so choice of their selection mostly depends upon the network topology chosen for the overlay network [TUNTAP].

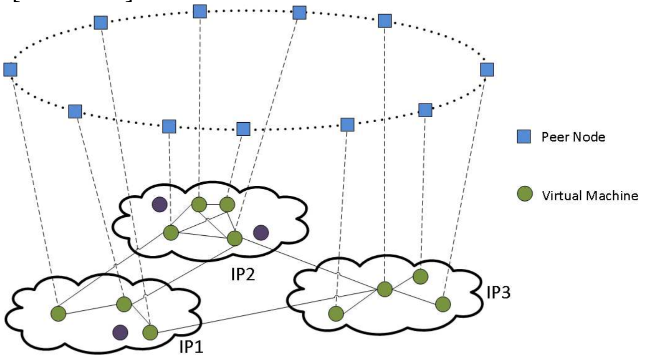

Figure 8. Establishment of a peer-to-peer overlay based virtual network on top of multiple cloud infrastructure providers.

Secure Virtual Private Connections. The key feature of the ICVPN is establishing a secure communication network between the peers of the overlay formed over a collection of cloud providers' infrastructure. To achieve this, we make use of public key cryptography which supports the Public Key Infrastructure (PKI) model. This model makes it possible to create authenticated and confidential end-to-end tunnels that provide protection against eavesdropping, message tempering and message forgeries. Another practical advantage of using PKI is the reuse of existing frameworks and tools which have been thoroughly tried and tested in a myriad of different domains, are widely used and have been adopted and integrated in countless security techniques.

The P2P client software sets up and configures the TAP/TUN device automatically and assigns unique IP addresses to their virtual interfaces and DNS hostnames to the peers themselves which can also be used as peer IDs and this record is put on the Distributed Hash Table (DHT) for the operation of the peer discovery service of the overlay network, which works by obtaining the list of unique peer identifiers and mapping them to the locally assigned IP addresses and host names. The same DHT is used for binding security credentials like certificates and keys to the unique peers and identify which service the peers are a part of. Once a peer has obtained the X.509 certificate of a peer it wants to communicate with from the DHT, they can directly negotiate and generate symmetric session keys for encrypted tunneled communication. These certificates are signed by a Certificate Authority (CA) trusted by the end users of the service or the SP. The CA model could be replaced by a cloud-oriented trust model, such as the one in the OPTIMIS toolkit.

Once equipped with the required security credentials and peer locations, the peers can proceed to form secure tunnels to complete the VPN construction. The DHT is used as the control channel that provides the meta-data to carry out the necessary configurations and connection setups.

Concluding remarks

With the five identified challenges for future clouds as a point of departure, our diverse research activities range from control theory for elasticity to cloud business models. Our research findings are incorporated into the OPTIMIS Toolkit, a set of self-management tools for cloud service and infrastructure optimization that can be used to implement clouds of multiple possible architectures. All toolkit management actions are harmonized by policies that incorporate aspects of trust, risk, eco-efficiency, and cost, without neglecting performance. Future directions for our work includes, e.g., research on policy-driven service management and investigations of migration and service relocation for continuous service lifecycle optimization.

Acknowledgments

We thank all members of the OPTIMIS consortia for their contributions to this work. We also acknowledge the anonymous reviewers for their valuable feedback. Financial support has been provided in part by the European Commission’s seventh framework programme under grant agreement no. 257115.

References

[OVF2011] Open Virtualization Format (OVF) - A standard from the Distributed Management Task Force. http://www.dmtf.org/standards/ovf.

[QMU2011] QEMU - An open source machine emulator and virtualizer. www.qemu.org

[DAKM2011] A Risk Assessment Framework and Software Toolkit for Cloud Service Ecosystems. K. Djemame, D. Armstrong, M. Kiran, and M. Jiang. To appear in the proceedings of the Second International Conference on Cloud Computing, GRIDs, and Virtualization, Rome, Italy, September 2011

[OPTIMIS2010a] Requirements Analysis – Deliverable 1.1.1.1, OPTIMIS Project, September 2010

[OPTIMIS2010b] Cloud Business: Needs, Desires, Concerns and aspirations in the Cloud ecosystem – Deliverable 7.1.2, OPTIMIS Project, December 2010

[OPTIMIS2011a] The Impact of OPTIMIS on Market Behaviour – Deliverable 7.1.3, OPTIMIS Project, July 2011

[OPTIMIS2011b] Cloud Legal Guidelines – Deliverable 7.2.1.2, OPTIMIS Project, May 2011

[OPTIMIS2011c] OPTIMIS: a Holistic Approach to Cloud Service Provisioning, A.J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R.M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S.K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forgó, T. Sharif, and C. Sheridan. Future Generation Computer Systems, 2011, doi:10.1016/j.future.2011.05.022

[Ali-Eldin2011] An Adaptive Hybrid Elasticity Controller for Cloud Infrastructures. A. Ali-Eldin, J. Tordsson, and E. Elmroth.Submitted. 2011

[Gambi2010] Protecting SLAs with Surrogate Models. A. Gambi, G. Toffetti, and M. Pezze. 2010. ACM.

[Rodero-Merino2010] From Infrastructure Delivery to Service Management in Clouds. L. Rodero-Merino, L.M. Vaquero, V. Gil, F.

Galán, J. Fontán, R.S. Montero, I.M. Llorente. Future Generation Computer Systems 26(8), p. 1226-1240.

[Katsaros2011] A Multi-level Architecture for Collecting and Managing Monitoring Information in Cloud Environments. G. Katsaros, G. Gallizo, R. Kübert, T. Wang, J. O. Fito, and D. Henriksson. International Conference on Cloud Computing and Services Science (CLOSER), 2011.

[GAMS2011] General Algebraic Modeling System, GAMS, http://www.gams.com/

[Rafaeli2003] A survey of key management for secure group communication. S. Rafaeli and D. Hutchison, ACM Comput. Surv., 35, 2003, pp. 309-329.

[Qiu2006] EKM: An Efficient Key Management Scheme for Large-Scale Peer-to-Peer Media Streaming. F. Qiu, C. Lin, and H. Yin. Advances in Multimedia Information Processing - PCM 2006. 2006, pp. 395-404.

[Emotive2011] Emotive Cloud Barcelona. www.emotivecloud.net

[Dinger2005] On the Challenge of Assessing Overlay Topology Adaptation Mechanisms. J. Dinger and H. Hartenstein, IEEE International Conference on Peer-to-Peer Computing (P2P'05). pp. 145-147.

[VTUN] VTun, Virtual Tunnel. http://vtun.sourceforge.net/.

[TUNTAP] TUN/TAP. http://en.wikipedia.org/wiki/TUN/TAP.

[APNIC] IPv4 Exhaustion. http://www.apnic.net/community/ipv4-exhaustion

[SmartLM2010] C. Cacciari, D. Mallmann, C. Zsigri, F. D'Andria, B. Hagemeier, D.García Peréz, A. Rumpl, W. Ziegler, M. Gozalo and J. Martrat. Software licenses as mobile objects in distributed computing environments. Euro-Par 2010 Parallel Processing Workshops (LNCS 6586). pp. 279-286

[WS-Agreement2007] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement specification (WS-Agreement). GWD-R (recommendation), Open Grid Forum, 2010. GFD.107 recommendation, http://www.ogf.org/documents/GFD.107.pdf.

[WSAG4J2011] WSAG4J - Web Services Agreement for Java. October 2010. http://packcs-e0.scai.fraunhofer.de/wsag4j .

[Huan2008] A DS-AHP Approach for Multi-attribute Decision Making Problem with Incomplete Information. Z. Huan, B. Gong and X. Xu. Journal of Expert Systems with Applications, Vol. 34, pp. 2221-2227, 2008

[Djemame2011] Brokering of Risk-Aware Service Level Agreements in Grids. K. Djemame, J. Padgett, I. Gourlay, and D. Armstrong. Concurrency and Computation: Practice and Experience, Vol. 23, No. 7, May 2011