Autonomic Provisioning of Hosted Applications with Level of Isolation Terms

Carsten Franke and Philip Robinson
SAP Research CEC Belfast
TEIC Building, University of Ulster
BT37 0QB Newtownabbey, UK
{carsten.franke, philip.robinson}@sap.com

Abstract

Autonomic provisioning of hosted applications in Enterprise Data Centers must be investigated as more and more business applications are executed in such an environment. In this scenario, the multiplicity and variability of software components and data to be dynamically deployed, interconnected and adjusted leads to more complex system management. In parallel, all customer requirements and constraints must be fulfilled. These requirements and constraints are specified in corresponding Service Level Agreements (SLA). Protection aspects are one important part of such agreements, as in a shared, open infrastructure, perceived risks with respect to loss of information confidentiality, integrity and availability increase. Legislation and compensation mechanisms make it imperative for the Data Center to acknowledge these perceived risks. Therefore, robust application isolation mechanisms must be considered. However, potential drawbacks due to the used protection mechanisms that might influence the overall system performance must be considered. In order to initialize an autonomic provisioning process, a concept for configuring the Data Center based on different Levels of Isolation is presented. Customers and Data Center providers can agree on the isolation guarantees they expect given their individual preferences specification between the perceived risks and the resulting costs associated with the chosen Level of Isolation.

1 Introduction

The way Enterprise software is composed, delivered and executed is currently changing. That is, the specific customer software systems are generated by combining individual services of potentially different service providers. Furthermore, the individual services are provided by Data Centers [10], such that customers access the individual services over the Internet. This leads to the clear separation between the service consumption and the service provisioning [28]. Note that the service provider is not necessarily equivalent to the software provider. In many cases, a service provider and a software provider agree on certain conditions in which the service provider can offer the different software services to its customers.

The offered service is in most cases already a composition of lower level services. Resources are then dynamically allocated corresponding to the service specification and the customer requirements. However, changing the resource allocation, the service composition or the service usage will inevitably result in the customer or Data Center provider changing their perception of risk regarding their potential for SLAs to be violated. These effects combined with the high frequency of new service requests and the dynamic of the different services make the management of Data Centers more and more difficult. As suggested by Kephart and Chess [16], "the only option remaining is *autonomic computing*: computing systems that can manage themselves given high-level objectives from administrators".

A Data Center's provisioning process must incorporates many constraints, service characteristics as well as the objectives of the Data Center provider and the objectives of the individual customers. One important aspect that needs to be better automated surrounds protection, security and isolation of hosted application services. Here, the perceived risks of customers, the available security mechanisms and the service provider's capabilities and objectives need to be equilibrated. This process implicitly determines the best trade-off solution between the risk of a customer to lose some business critical data and the costs caused

by protecting the mentioned application and data. Relations between information security and risk are being researched today in many areas, for example within the context of *cyber-insurance* [1, 2, 13, 27].

This paper introduces the *Level of Isolation (LOI)* concept: a new approach to initialize an application provisioning process in a Data Center that enables the differentiations in perceived risks of different customers. It provides a concrete mapping between isolation guarantees offered to satisfy the perceived risks of customers and the technical templates that configure the Data Center's infrastructure in order to implement the LOI agreed with the customer. Different LOI configurations are related to different customer perceptions of the trade-off between risk and LOI costs. These costs here are qualitative approximations of the operational costs associated with the selected technical configuration. This is realized by specifying a Compartment data model as an architectural metaphor for application isolation. High level Service Level Agreements (SLAs) that include an agreed level of isolation can be then used to initialize and manage the behavior of an autonomic application provisioning process.

Section 2 describes the autonomic Data Centers and related approaches in more detail. This is followed by the description of the LOI concept in Section 4 and architectural description in Section 5. In Section 6, we discuss the merits and drawbacks of the LOI solution, ending with a conclusion in Section 7.

2 Background

The constantly increasing demand for compute power is a well-known and ongoing topic of research, as discussed, for example, by Foster and Kesselmann [6]. Furthermore, the scheduling aspects to further improve the efficiency of individual clusters is for example addressed by Franke et al. [7, 9]. The majority of the developed methods and concepts focus on increasing the compute power as the main criterion that drives the development. Beside the goal to increase the compute power of Data Centers, Franke et al. [10] have shown, that the autonomicity and autonomy of such system is of highest importance as only such systems can provide the necessary reliability and failure robustness.

As described by Franke et al. [8], business applications require a well-defined statement of security throughout the entire execution, as well as flexibility. These security requirements are currently addressed in a limited way in the scientific domain. However, some scientific projects currently start to also address the security demands of businesses such that the corresponding outcome could be applied in the scientific as well as in the business scenario, see Yang et al. [29].

In their study, Yang et al. [29] have identified isolation as being one of the key issues for securing the Data Center processing. Isolation is a property of an entity (system, application, component, program or process), that defines its ability to perform reliably regardless of the behavior of other entities. The theory of Isolation in information systems goes back to the late 1960's and 1970's, where the Virtual Machine and Virtual Machine Monitor were introduced to support resource sharing of largescale computers such as the IBM 370 [22]. Isolation was achieved by multiplexing accesses to the physical network and machinery, making one physical machine appear to be many. Since then, this basic model of isolation has been adapted and extended in various forms including Operating System Virtualization, Virtual Private Networks, Transactions, Virtual Disks and Virtual Memory. However, Data Center isolation today entails having separate physical machines per customer, which will not scale with the expected increase in customers and concurrent dynamic resource requirements.

In general, a customer's concerns regarding the protection of its hosted applications and data in a Data Center can be addressed in many ways. For example, some physical firewalls can be used to clearly disconnect dedicated servers running applications of different and maybe competing companies. As one can see, the different protection approaches correspond with different costs necessary to implement the specific protection mechanism. Thus, different customers at data centers will have different perceptions of the best trade-off solution between a high level of security and the related costs. Some research has been done in the area of multilevel security, see for example Foley et al. [5] or in the area of IT business outsourcing, e.g. Karabulut [14]. However, these research frameworks do not offer comprehensive solutions for the handling of trade-offs, alternatives and technical configurations.

In order to establish a contract between the service provider and the customer in many business environments, the customers still need to sign paper-written contracts that clearly specify the provision of specific services and the corresponding behavior rules and costs for the service consumers. This becomes a bottleneck as the number of customer requests per time interval is constantly increasing. Thus, automatic mechanisms need to replace the manual negotiation and specification of the agreements. The specification and usage of *Service Level Agreements* (SLA), see for example Keller and Ludwig [15] or Ludwig et al. [21], that can automatically be evaluated by a system is already established. However, the application of such SLAs is still

limited to scientific applications. That is, the SLAs mainly specify the compute power needed to perform certain tasks. The SLAs in the business case must address additional requirements. Here, we see a clear demand for protection requirements that are poorly addressed in the existing approaches. Ludwig et al. [20] investigated the usage of templates for the SLA specification as well as for the corresponding provisioning. However, the authors do not investigate the implications on the protection or security aspects and the corresponding provisioning methods.

Additionally to the specification of SLAs, the negotiation between service providers and service consumers need to be automated. Again, driven by scientific applications, some approaches for negotiation protocols exists, see for example Li and Yahyapour [17, 18, 19]. However, these attempts again focus on the computational requirements for certain services and omit the customer's protection and security concerns.

In this paper, we present a systematic approach to define possible Level of Isolation and suggest a way to include them into SLAs and the corresponding negotiation protocols. This enables the data center management to automatically and autonomously establish the necessary contracts between the service customers and providers.

3 Analysis of Potential SLA Violations

This section defines a structural model of Data Center components used to analyse potential SLA violation sources and targets. A SLA violation is any malicious or accidental occurrence in the Data Center that disrupts the fulfilment of one or more SLA terms. These may be driven by physical or logical entities in the Data Center. This is used to classify the different types of isolation mechanisms that could be employed to reduce the possibilities of these violations. We define a Data Center structurally with respect to its Participants and Managed-Elements. Participants are human or organizational entities that include the roles of Administrator, Customer and User. Managed-Elements are entities that provide functionality to the set of Participants. Figure 1 illustrates the structural dependencies between different types of Participants and Managed-Elements in a simplified Data Center instance.

The Data Center serves several Customers, which might access different entities in the Data Center via the Internet. The Data Center's facilities may consist of several Buildings, where each Building houses different, individual network Routers that act as gateways to different Network Domains. Within each Network Domain are multiple Machines, which may

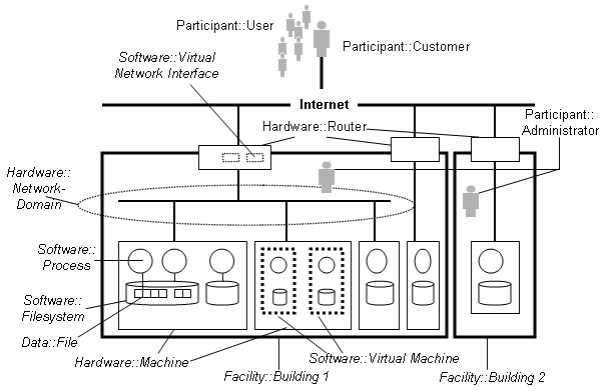

Figure 1. Illustration of Data Center Entities

be physical Hardware Machines or Virtual Machines. Each Machine then hosts one or more Software Elements, including Application Servers, Filesystems and Databases, which subsequently contain Processes, Files and Data.

Using the above description of entities in the Data Center, Table 1 provides a description and classification of potential SLA violations.

Hierarchical dependencies between violations are also implied by the listing in Table 1; for example, damage to facility entities will potentially lead to damaged Hardware, hence corrupted Software and hence lost Data. Therefore, depending on what infrastructure elements two different customers share to support their separate applications, the individual risk of SLA violations varies.

4 The Level of Isolation Concept

This section proceeds to explain the *Level of Isolation* (LOI) concept. The basic idea follows the proposition that the isolation guarantee offered by a Data Center for a particular application should be relative to the customer's perceived risk of interrupted service as a result of another customer's application and allocated Managed Elements failing or acting maliciously.

The negotiation of a SLA between the customer and the Data Center provider is normally a multi-step process in which many functional and non-functional parameters are addressed. The customer and the Data Center provider negotiate the appropriate Level of Isolation and the corresponding costs. The perceived risks of the customer will in many cases directly lead to an LOI. However, the customer needs to implicitly determine the best trade-off between the avoidance of his perceived risk by choosing an appropriate LOI and the related costs caused by the specific LOI.

Entity Type	Examples	Potential Violation
Participants: people or organizations with privileges that allows access to services of the Data Center as specified in SLAs	Administrators, Customers, Users, Software Vendors	May abuse privileges in order to gain knowledge about other Participants' data and applications, or block their access
Facilities: physical, non-computational entities that enclose Hardware and provide access to utilities	Buildings, Rooms, Server racks, Cabinets, Cables	Physical damage may lead to loss of multiple items of Hardware and hence services
Hardware: physical, computational entities that provide processing, storage and networking	Servers, Workstations, Harddrives, Tapedrives, Routers	Failure or destructions leads to unavailability of software
Software: executable programs, scripts, procedures and rules that provide application and platform functionality	Application servers, Database management systems, Operating system, Virtual machines	Software may contain flaws or be compromised, causing alterations in normal behavior that corrupt and make data unavailable; this may be propagated to applications of other customers
Data: series of bits used to store and transport information digitally	Program files, Documents, Models, Personnel information, Catalogues, Certificates, Namespaces	Corrupted data provides inaccurate information used to perform calculations, make predictions

Table 1. Description of Data Center Entities and their SLA violation threats

The overall process of negotiating the LOIs and the corresponding system configuration should remain relatively transparent to an *autonomic application provisioning process* that schedules, deploys, disables, migrates and adjusts resources allocated for the application within minimal human supervision. While different customers will choose different sets from this set of hosted applications, they may also show variation in their relative perceived risks. This is the case for which the LOI concept has been developed. There are two main aspects of the concept: (1) the idea of including LOI terms in a SLA negotiation process, and (2) a generalized Compartment model that encapsulates the deployment and runtime of a single application under the governance of a single SLA.

4.1 Introducing the Level of Isolation to SLAs

In the context of application hosting, a SLA contains a collection of measurable terms that represent *guarantees* provided by the Data Center to a customer. Note, an SLA typically consists of many other additional terms between customer and Data Center provider. In the Data Center scenario, guarantees typically refer to response times, storage capacities, periodic costs and availabilities. An autonomic provisioning process would use these terms to modify its behavior with respect to selection and adjustment of infrastructure resources in the Data Center's fabric. We now introduce the LOI as an additional term in a SLA, representing an agreed isolation guarantee. This LOI term is associated with a quantitative approximations of the related cost of the technical operation associated with providing the isolation guarantee.

We have identified seven LOIs, based on the entities illustrated in Figure 1. Figure 2 now enhances Figure 1 to depict an exemplary Data Center with two Customer SLAs, configured with at least one instance of each of the LOIs we have identified.

While Figure 2 illustrates a number of conceivable physical and virtual ways of implementing isolation, we only go to the extreme of having applications housed in different buildings, which could be the case for very critical, sensitive applications. The 7 LOIs we identify are exemplary, as there could be any number of options offered by a Data Center, depending on their resources, technical knowledge and scope of business, but represent in our opinion a reasonably comprehensive classification of possibilities. Our naming of the different LOIs starts with the isolation level 1 which has the highest security risk associated with the lowest cost and ends with LOI 7 that represents the isolation

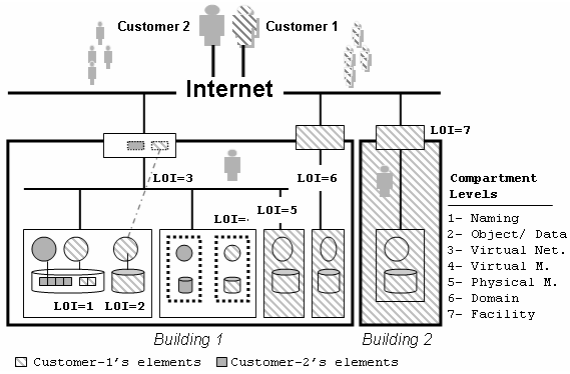

Figure 2. Reference architecture for the LOI concept

level that provides the best security protection mechanism but also leads to highest implementation costs. Furthermore, we establish exact specifications for isolation levels that are stored as *templates* accessible to a *provisioning service*.

`LOI 1- Naming Isolation`: objects and processes are associated with a namespace unique to the customer but may be stored in the same filesystem, use same server software instance and executed in same memory space; for example, consider hosting webpages on the same application server, as depicted in Figure 2. Royon et al. [26] present an architecture for a Virtualized Service Gateway that implements this concept, such that applications in different namespaces cannot see each other; *Risks are highest* as failure or compromise of one database or software instance affects many customers; *Costs are lowest* as the same application instances and physical machines can be used to support the objects and processes of several customers, given the capacities of the software and hardware;

`LOI 2- Object Isolation`: objects are placed on a separate disk partition of a machine but processes execute in the same physical and logical memory space as processes of other customers. Figure 2 depicts this as a database and an application running as separate instances as opposed to the depiction of LOI 1. Disk partitioning is a very commonplace technique for isolation and is also used for gaining performance advantages as presented by Ghandeharizadeh et al. [11], where they partition using a multiplexing technique. Linux's `chroot` is also an example of this pattern, as it logically changes the root directory of a disk and protects access to files outside of the paths under this new mount point. *Risks are high*, as there is a mechanism for protecting against propagation of data compromise but not against execution or machine failure; *Costs are low* as physical resources are still shared and the partitioning mechanism is simple and does not add significantly to the overhead;

`LOI 3- Virtual Network Isolation`: processes and objects can only be accessed and interacted with over a reserved [virtual] network interface, by multiplexing using encryption over the same physical network interface. Figure 2 shows the multiplexing of the shared Internet gateway leading from Building 1. Each customer's compartment would therefore communicate using a totally different logical subnet, interface and IP address. This method is implemented by Virtual Private Networks (VPN), where various, flexible models for provisioning and resource management have been proposed, such as by Duffield et al. [3]. *Risks are lower than for LOI 2* as there is better protection of the customer's confidentiality, integrity and access control but fail-independence and non-interference are still not guaranteed; *Costs are higher than for LOI 2* as there is no necessity to use separate hardware, although the computation may be increased with the additional, computational overhead of encryption, assuming a resilient keylength is chosen. Nevertheless, encryption overheads are becoming more and more negligible, with faster algorithms, implementation in specialized hardware and more powerful machines;

`LOI 4- Virtual Machine Isolation`: process and objects are executed and stored in a separate logical memory space and partition of a machine. Figure 2 shows one physical machine being shared by the processes and objects of two customers, each running in a dedicated Virtual Machine (VM). VMs have already been introduced as underlying the general theory of isolation above, but there are many more products such as VMware and XEN that implement this pattern for commodity machines. Matthews et al. [23] report on a comparison of performance isolation properties of various types of Virtual Machine technologies, comparting the performance impact that a malicious VM may have on another, where they have the same host. In that these results show some variance in response to stress test due to architectural difference, we would advise finer granularity of this particular LOI into classification by Full-Virtualization,

Para-virtualization and OS Virtualization. The results showed that Full-Virtualization was capable of providing true performance isolation and Para-Virtualization was the most susceptible to malicious stress from co-guest VMs. *Risks lower than for LOI 3* as the propagation of failure or compromise of one Virtual Machine is difficult, without significant effort and privileged access as reported by Ormandy [25]; *Costs are higher than for LOI 3* as using these solutions incorporates having sufficient licenses for the software, as well as the additional computational requirements placed on the underlying physical machines.

`LOI 5- Hardware-Machine Isolation`: processes and objects of customer are stored on individual physical machines or on physically, separated, stand-alone computational units (such as Blade Servers), reserved for sole access by the customer and possibly an appointed administrator. Figure 2 shows this level as the processes and objects of a single customer being hosted by a single, dedicated physical machine. *Risks are lower than for LOI 4* as physical isolation is used and machine failures are easier to isolate, although there are cases where viruses and attacks can be propagated over a shared, physical network; *Costs are higher than for LOI 4* when considering that a data center may have 10s of 1000s of customers and users, as well as additional machines for fail-over and archival.

`LOI 6- Physical Network Isolation`: processes and objects of customer are stored on a physical machine reserved only for that customer and is also connected to a physical gateway reserved also for the customer, as shown in Figure 2. This is usually requested by premium customers that can afford to lease their own backbone with the provider. *Risks are lower than for LOI 5* as this is almost perfect operational isolation; *Costs are higher than for LOI 5* as the customer demands an additional network of machines to be maintained.

`LOI 7- Location Isolation`: the resources reserved for the customer are stored in a separate, physical room or building e.g. in Building 2 in Figure 2. *Risks are lowest* as there is no interference or dependence on other customers - this is the ultimate in isolation in the context of hosting, as the customer's objects and processes are completely sealed off from others; *Costs are highest* as an entire physical space and utilities are dedicated to customer, limiting the data center's ability to make best usage of its utilities;

Each LOI represents a different configuration in response to perceived risk of SLA violation due to resource sharing. At a high level of abstraction, we can state that the same basic isolation policies and maintenance protocols are implemented using different assumptions, mechanisms and hence offering different guarantee levels of protection. This satisfies the rule that the cost of security should be commensurate with the calculated risk.

4.2 Generalized Compartment Model

This section introduces *Compartments* as the architectural metaphor for isolation used to realize the LOI concept. Figure 3 depicts model of a Compartment in relation to Managed-Elements, SLAs, LOI terms, Gateways (physical or logical access points) and Participants. The model is then described, at the same time establishing a set of security and design principles for autonomic provisioning with isolation requirements:

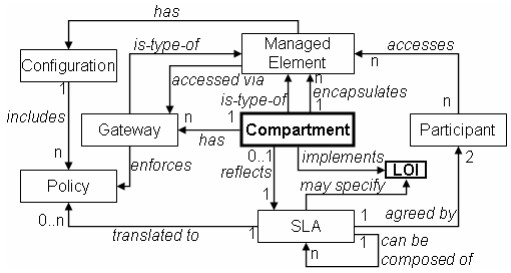

Figure 3. Compartment model to support LOI concept

Firstly, the Data Center itself is representative of the top-level or root Compartment, where each Compartment maintains a registry of its Managed-Elements including other (sub) Compartments. From a software engineering perspective, all Compartments have an inheritance relationship with the Managed-Element class, and are modeled as specializations of Managed-Elements with containment and isolation capability. Note that encapsulation, containment and isolation are used seemingly interchangeably but represent different aspects of the Compartment description: containment refers to its ability to maintain a registry or spatially contain Managed-Elements, encapsulation refers to the property that Managed-Elements in a Compartment cannot be externally referenced by name without some reference to the encapsulating Compartment and isolation maintains the operational property that has been introduced throughout the paper. Returning to the engineering aspect of Compartments, in that they are

subclasses of the Managed-Element class, Compartments implement the management interface below - noting that for Compartments, the operations are considered in an aggregate sense:

- `init`: initializes the instance of a Managed-Element/ Compartment with a set of configuration variables
- `start`: deploys the Managed-Element/ Compartment, creates a registration in the registry of its encapsulating Compartment (i.e. sub-Compartments are supported) and makes it available for usage
- `stop`: disables the availability of the Managed-Element/ Compartment but does not remove it from the appropriate registry
- `terminate`: removes the properties and state of the Managed-Element from the appropriate registry
- `check-health`: checks predefined parameters of a Managed-Element's state for compliance with a set of terms defined in a governing SLA
- `set-config`: adjusts the configuration of a Managed-Element to an updated set of variables

Secondly, multiple Compartments provide the same logical operational interface but vary with respect to their internal implementation and guaranteed protection against SLA violation, in accordance with the LOI stated in the governing SLA. Managed-Elements isolated by the same Compartment are governed by the same SLA and share the same referential namespace. Encapsulation in this context suggests the following security principles, within the extremes of the LOI's guarantee of protection against SLA violation:

- **Confidentiality**: data maintained by Managed-Elements in one Compartment cannot be read outside the Compartment without possession of a secret stored within the Compartment
- **Integrity**: operations performed on data maintained by Managed-Elements in a Compartment must be verifiable with keys known and trusted in the Compartment
- **Access control**: access to operations on Managed-Elements in a Compartment from external Managed-Elements or Participants, via their interconnected *Gateways*, can only be granted if there is an explicit Policy that permits this operation
- **Fail independence**: the failure of Managed-Elements in a Compartment A do not cause the Managed-Elements of another Compartment B to fail, given that $A \cap B = \emptyset$ holds
- **Non-interference**: data exchanges and interactions between Managed-Elements in a Compartment cannot be externally monitored, unless there is an explicit Policy that allows notifications and alerts to be propagated externally

As Compartments are instances of Managed-Elements, they may contain other Compartments, representative of a spatial or logical topological orientation. Compartments therefore provide the additional management operations *addElement*, *removeElement*, and *getElement* in order to add, remove, or get certain Managed-Elements.

Each Compartment provides one or more Gateways with explicit Policies that restrict entry and access to the Compartment and Managed-Elements respectively. These compartment oriented Policies are assumed to be of the format `<subject, target, function, constraint>`:

- `subject`: either a Managed-Element or Participant that initiates an access request
- `target`: a Managed-Element targeted by an access request
- `function`: the operation or action on the `target`'s interface or state
- `constraint`: a constant or logical expression that defines under what conditions the `subject` can perform the `function` on the `target`

With reference to Figure 1, different types of Compartments can be compared when making provisioning decisions. For example, two different Customers may want to have the same application and data hosted but request and pay for different facilities. Customer 1, with a very high perceived risk might go to the extreme of demanding an exclusive Building or Room in the Data Center to be reserved for providing the execution environment for their application processes and Data. Customer 2 might however be satisfied to pay a reduced price for having a software Virtual Machine represent the encapsulation of its execution environment. There are clearly two different extremes for specifying and guaranteeing isolation, but they can be abstractly treated in the same manner. That is, regardless of if Compartments are physical, logical or virtual (i.e. simulating physical using logical methods), a top-level provisioning process performs only the operations specified in their Managed-Element or Compartment

class interfaces. The inner-workings of the Compartment, in response to different provisioning actions, are based on the type of Compartment instance affiliated with the respective LOI. The next section proceeds to explain the provisioning process that acts on the set of Compartments in the Data Center.

5 Technical Realization

This section proceeds to define the technical system architecture for realizing the LOI concept using the Compartment model. In doing so we describe how a set of *decision-making components* use the LOI terms in an SLA and the Compartment model to implement an *autonomic provisioning process*. According to Ludwig et al.[20], a provisioning process - in particular one that is automated and agreement-driven - (1) identifies the parts of a SLA which are relevant for supplying resources, (2) derives quantities for different resource types, determines and outlines alternative resource assemblies that might be used, and (3) compiles a provisioning plan detailing the required resources, their configuration, and their provisioning. The technical architecture we present, supports such a process by manipulating instances of the Compartment model defined in Section 4.2. A Compartment can be provisioned after a Customer and the Data Center have negotiated and reconciled differences between what SLA violation risks the Customer perceives, what the Customer is prepared to pay and what technical infrastructure options the Data Center can provide at the time of offering. With reference to the 7 LOI configurations proposed in Section 4.1, the technical infrastructure options would result in 7 classes of SLA templates and 7 corresponding Compartment configuration templates. Note that there may be other templates available for selecting specific software service product families, again depending on the needs of the Customer. In addition, to support the autonomic execution of the entire process, we propose the inclusion of three additional templates that define the *autonomic strategies* employed by the decision-making components. The interfaces for loading these strategy templates are indicated in Figure 4 for (A) Negotiation, (B) Infrastructure resource selection and (C) Incident response respectively. We however do not present any details on these strategies at the moment.

The above-mentioned SLA, Compartment configuration and strategy templates are stored and instantiated by their respective decision-making components. However, the component with the highest storage demands is what we refer to as the Infrastructure Registry and Compartment Manager, see Figure 4.

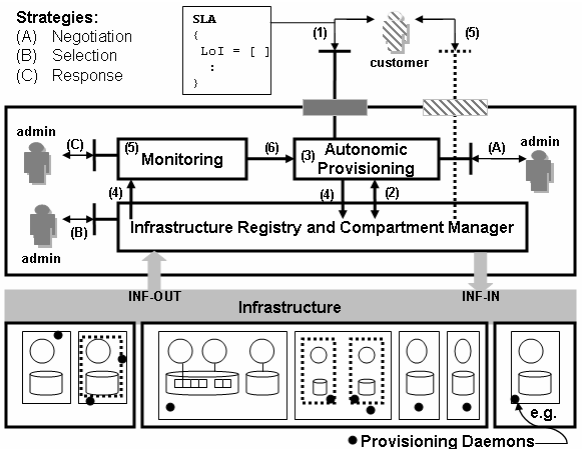

Figure 4. Simplified system architecture for autonomic provisioning

A SLA Template could be based on the schema specified by WS-Agreement [24], where one of the `guarantee` terms in the schema could be reserved for specifying the eventually agreed LOI. In addition, the `context` section of their schema would be reserved for including the 2 Participants that have agreed to the terms in the SLA (recall the conceptual model in Figure 3). A Compartment configuration template defines a set of Managed-Element descriptions and deployment procedures, according to an agreed specification. For example, the Smart Frog framework [12] provides technology for describing distributed software systems as collections [i.e. Compartments] of cooperating components [i.e. Managed Elements], as well as their activation and management. In order to support management and activation, each machine in the Data Center's infrastructure should be equipped with very simple software agents, which we refer to as *Provisioning Daemons* in Figure 4, that run and listen in the background, awaiting commands from the Provisioning Service. Each Provisioning Daemon has platform specific logic installed for executing the specification of Compartment configuration templates according to the level of isolation (LOI).

We now proceed to describe the technical functionality of each component of the architecture, as depicted in Figure 4. For each component, we describe its basic purpose, followed by a description of its basic inputs, processing, and outputs.

Autonomic Provisioning Service: this is the main component in the architecture for making infrastructure component selection based on SLAs, integrating

the Compartment configuration templates and enacting the resultant deployment specification of the Compartments to be activated. It hence provides an interface for negotiation between the customer and the data center administrator.

INPUTS: a Customer provides a set of service requirements with SLAs, as in labels (1) of Figure 4, where SLA's are associated with one or more services and define agreed costs and guarantees (e.g. LOI) per service;

PROCESSING: the service determines the compartmentalization required to achieve the SLA, labelled as (3) in Figure 4; an administrator may be involved at this stage in order to load the Negotiation Strategy using interface (A);

OUTPUTS: the output is a selected Compartment configuration template

Infrastructure Registry and Compartment Manager: this large, critical component provides a common management interface (`INF-IN` and `INF-OUT`) to the set of heterogeneous infrastructure resources. In addition, this maintains all compartment schedules labeling them as active or inactive, along with their templates.

INPUTS: the first input to this component would be the query by the provisioning service to its resource registry, as shown in label (2) of Figure 4. Secondly, the provisioning service passes compartment configuration templates as shown in label (4) of Figure 4;

PROCESSING: a query of the resource registry is performed to determine their capability, capacity and availability. This may be assisted by the Selection Strategy loaded by an administrator via interface (B). Secondly, the major processing following label (4) in Figure 4 is the allocation of resources for the compiled compartment templates.

OUTPUTS: returns available resources, again depicted by interaction (2) of Figure 4.

Monitoring: this is a passive set of components that collect information about the infrastructure's state and capacity, and pushes this information to the Provisioning Service, such that migration or resizing decisions can be made e.g. due to failures.

INPUTS: low level events from the Provisioning Daemons are collected. The creation of a new Compartment, as triggered by label (4) of Figure 4, is also an example of an event.

PROCESSING: label (5) of Figure 4 shows the filtering of lower level information and creation of higher-level health check and compliance events, depending on the types of applications hosted by the data center. This may happen in parallel to the Customer using the compartment, as indicated by the duplication of label (5).

OUTPUTS: health check and compliance events of form `<compartment, LOI, hw-flag>`, where `flag` indicates whether or not the current physical host is in a healthy state. Label (6) of Figure 4 shows such an event being sent to the provisioning service.

In the following, we add some technical notes about the Compartment lifecycle and the provisioning process. Firstly, the setting up, instantiation and activation of a Compartment includes enabling roles, access control lists and other special configuration parameters on the physical hosts. When the compartment is completely initialized and activated (this may be based on an agreed schedule), a service endpoint to its Gateway is generated, customized for the customer and returned via some online or offline medium. Secondly, in that a Compartment is instantiated per SLA as opposed to per Customer, a customer may have multiple compartments and may change the SLA on an existing compartment. This may include changing the LOI of one or more components in the original agreement, resulting in possible migration and readjustment of the Managed-Elements supporting the Compartment. This may also have an impact on other Compartments from different customers; for example, a decision to support a customer A at LOI 6 may result in moving another customer B's application components at say LOI 3 from one machine to another to accommodate customer A. Note that there my also be specific administrators assigned to dealing with specific customers. Even for LOI 7, this would imply that only they carry the key to the particular location; For example, the RFID key of their entry card is associated with access rights to the room and case.

6 Discussion

In the seminal paper from Kephart and Chess introducing the Vision of Autonomic Computing [16], they identify four key objectives: (i) self-configuration, (ii) self-optimization, (iii) self-healing and (iv) self-protection. We discuss the LOI concept and Compartment model in the context of these four objectives, with the aim to validate the technical architecture for autonomic provisioning that we have proposed.

Self-Configuration refers to the automated configuration of components and systems based on high-level policies. As the LOI terms are mapped to specific, concrete Compartment templates with specifications of Managed-Element deployment steps, achieving self-configuration is an inherent property of the concept. It requires a significant effort to develop the initial Compartment specifications, but, once this is completed, the task of instantiating and activating Compartments is performed by the autonomic provisioning service.

Self-Optimization refers to the continuous, systematic improvement of performance and efficiency. Maintaining logs of transaction histories and accounting data are critical throughout the Data Center. Instances of the compartment templates can be used as instruments for keeping track of where specific customer objects and processes are currently running, what collective resources they have consumed and what operational conditions they have been subjected to. In doing so, optimization strategies can be developed per Compartment, as well as across the Data Center.

Self-Healing detection and repair of localized software and hardware problems. During monitoring, using the technical architecture described in Section 5, there may be states reached where a system's health check fails, but, as a result of the agreed LOIs, across the set of customer compartments, this reduces the number of possible recovery scenarios that need to be determined. Furthermore, using our solution, the Data Center provider can make the temporary decision to sustain risks and liabilities during a critical condition such as node failure. Such decisions should be guided by the agreed compensation costs in the respective SLAs, for responding to data center attacks, failures or mischances. This leads to another trade-off decision for the provider, where he has to select an application in case there is the need to perform some hot fixes.

Self-Protection refers to automatic alerts and defense against malicious attacks and propagating failures. With the LOI concept, quantitative approximations of cost and risk are mapped to specific, technical protection mechanisms, procedures and constraints, such that automated support for negotiation is possible, without an opportunity for misunderstanding and violation of compliance. Our estimates of cost and risk are trends based on intuition and experience with managing different system configurations. There are however many other factors including number of components, size of database, type of software and machine specification that will also cause different weights to be considered in the resource selection decisions. Nevertheless, we see the inclusion of security concerns in the automatic negotiation process and explicit inclusion in the SLAs as a key advantage.

7 Conclusion

We have shown how to include concerns of application hosting costs and perceived risks of SLA violation into the process executed by an autonomic provisioning service. The autonomic provisioning service is still able to act autonomously, maintain the agreed risk to the isolation of the customer's application and give them the resource usage that they agreed to.

In order to support this concept, we have defined a set of concrete isolation level specifications, an abstract Compartment object model and a technical architecture, which can be implemented using various existing and emerging technologies. These have been packaged as 7 isolation configurations that show an inverse dependency between provisioning costs and SLA violation risks. This enables the establishment of negotiation strategies based on reconciling conflicts between these terms in the SLA offers exchanged between Data Center providers and Customers.

Other foreseen benefits include a means for simple accounting per compartment, as resource usage may be distributed, as well as keeping track of where objects and processes belonging to one customer are currently physically hosted. Secondly, a single management component can be used to manage different types of compartments, unlike the state of the art today that focus mainly on managing large amounts of Virtual Machines and Virtual Centers or uniformed clusters.

Future work includes the integration of performance concerns in the decision process, such that we will proceed to a better estimation of resource requirements, costs and returns. Secondly, migration is an important topic in data center management but there is not a large amount of work on *migration strategies* and how to go about organizing extensive data center reorganization. We believe that the LOI concept and Compartment model provide a framework for better enabling the realization of such strategies.

Acknowledgments

We would like to acknowledge the support from the European Commission under IST program #FP6-033576. Thanks to the anonymous reviewers from the workshop for their detailed comments that have significantly improved the paper, as well as to Bryan Stephenson from HP Research for his feedback.

References

[1] W. S. Baer and A. Parkinson. Cyberinsurance in it security management. *IEEE Security and Privacy*, 5(3):50–56, 2007.

[2] R. Bohme. Cyber-insurance revisited. In *Workshop on the Economics of Information Security (WEIS)*. Infosecon, 2005.

[3] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive. A flexible model for resource management in virtual private networks. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 95–108, New York, NY, USA, 1999. ACM.

[4] C. Ernemann and R. Yahyapour. *"Grid Resource Management - State of the Art and Future Trends"*, chapter "Applying Economic Scheduling Methods to Grid Environments", pages 491–506. Kluwer Academic Publishers, 2003.

[5] S. N. Foley, S. Bistaelli, B. O'Sullivan, J. Herber, and G. Swart. Multilevel security and quality of protection. In *Proceedings of the First Workshop on Quality of Protection*, 2005.

[6] I. Foster and C. Kesselmann. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1998.

[7] C. Franke, F. Hoffmann, J. Lepping, and U. Schwiegelshohn. Development of scheduling strategies with genetic fuzzy systems. *Applied Soft Computing Journal*, 8(1):706–721, January 2008.

[8] C. Franke, A. Hohl, P. Robinson, and B. Scheuermann. On business grid demands and approaches. In *Proceedings of the 4th International Workshop on Grid Economics and Business Models (GECON 2007)*, volume 4685 of *Lecture Notes in Computer Science*, pages 124–135. Springer, 2007.

[9] C. Franke, J. Lepping, and U. Schwiegelshohn. Genetic Fuzzy Systems applied to Online Job Scheduling. In *Proceedings of the 2007 IEEE International Conference on Fuzzy Systems*, pages 1573–1578, London, June 2007. IEEE, IEEE Press.

[10] C. Franke, W. Theilmann, Y. Zhang, and R. Sterritt. Towards the autonomic business grid. In *Proceedings of the Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASE07)*, pages 107–112, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[11] S. Ghandeharizadeh, S. H. Kim, and C. Shahabi. On configuring a single disk continuous media server. In *SIGMETRICS '95/PERFORMANCE '95: Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 37–46. ACM, 1995.

[12] P. Goldsack, J. Guijarro, A. Lain, G. Mecheneau, P. Murray, and P. Toft. Smartfrog: Configuration and automatic ignition of distributed applications. Technical report, HP, 2003.

[13] L. A. Gordon, M. P. Loeb, and T. Sohail. A framework for using insurance for cyber-risk management. *Communications of the ACM*, 46(3):81–85, 2003.

[14] Y. Karabulut, F. Kerschbaum, F. Massacci, P. Robinson, and A. Yautsiukhin. Security and trust in it business outsourcing: a manifesto. *Electron. Notes Theoretical Computer Science*, 179:47–58, 2007.

[15] A. Keller and H. Ludwig. Defining and monitoring service-level agreements for dynamic e-business. In *Proceedings of the 16th Conference on Systems Administration*, pages 189–204, 2002.

[16] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[17] J. Li and R. R. Yahyapour. A strategic negotiation model for grid scheduling. *International transactions on systems science and applications*, 1(4):411–420, 2006.

[18] J. Li and R. Yahyapour. Learning-based negotiation strategies for grid scheduling. In *IEEE Int'l Symposium on Cluster Computing and the Grid (CCGrid 2006), Singapore*, pages 567–583. IEEE Press, 2006.

[19] J. Li and R. Yahyapour. Negotiation strategies for grid scheduling. In *Proceedings of the 1st International Conference on Grid and Pervasive Computing*, pages 42–52, 2006.

[20] H. Ludwig, H. Gimpel, A. Dan, and R. Kearney. Template-based automated service provisioning - supporting the agreement-driven service life-cycle. In *Proceedings of the 3rd international conference on Service-Oriented Computing 2005 (ICSOC)*, volume 3826 of *Lecture Notes in Computer Science*, pages 283–295. Springer, 2005.

[21] H. Ludwig, T. Nakata, O. Wäldrich, P. Wieder, and W. Ziegler. Reliable orchestration of resources using ws-agreement. In *Proceedings of the second International Conference on High Performance Computing and Communications*, pages 753–762, 2006.

[22] S. Madnick and J. Donovan. Application and analysis of the virtual machine approach to information system security and isolation. In *Proceedings of the workshop on virtual computer systems*, pages 210–224. ACM, 1973.

[23] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens. Quantifying the performance isolation properties of virtualization systems. In *ExpCS '07: Proceedings of the 2007 workshop on Experimental computer science*, page 6. ACM, 2007.

[24] O. G. F. (OGF). Web services agreement specification (ws-agreement). OGF Specification, 2006.

[25] T. Ormandy. An empirical study into the security exposure to hosts of hostile virtualized environments. In *CanSecWest '07: Applied Security Conference*. CanSecWest, 2007.

[26] Y. Royon, S. Frénot, and F. L. Mouel. Virtualization of service gateways in multi-provider environments. In *CBSE*, pages 385–392, 2006.

[27] B. Schneier. Insurance and the computer industry. *Communications of the ACM*, 44(3):114–115, 2001.

[28] M. Turner, D. Budgen, and P. Brereton. Turning software into a service. *Computer*, 36(10):38–44, 2003.

[29] E. Y. Yang, B. Matthews, A. Lakhani, Y. Jegou, C. Morin, O. D. Sanchez, C. Franke, P. Robinson, A. Hohl, B. Scheuermann, D. Vladusic, H. Yu, A. Qin, R. Lee, E. Focht, and M. Coppola. Virtual organization management in xtreemos: An overview. In *Proceedings of the CoreGRID Symposium 2007 Rennes, France (CoreGRID07)*, pages 73–82. Springer, August 2007.