PAEAN4CLOUD
A Framework for Monitoring and Managing the SLA Violation of Cloud Service-based Applications

Yehia Taher[1], Rafiqul Haque[2], Dinh Khoa Nguyen[3] and Beatrice Finance[1]
[1]*Laboratoire PRiSM, Université de Versailles/Saint-Quentin-en-Yvelines, Versailles, France*
[2]*Laboratoire d'InfoRmatique en Image et Systèmes d'information, Université Claude Bernard Lyon 1, Lyon, France*
[3]*European Research Institute in Service Science, Tilburg University, Tilburg, The Netherlands*
{*yehia.taher, beatrice.finance*}*@prism.uvsq.fr, akm-rafiqul.haque@univ-lyon1.fr,dinhkhoanguyen@gmail.com*

Keywords: Business Transaction, Cloud Computing, Blueprint.

Abstract: Recently, Cloud Computing has become an emerging research topic in response to the shift from product oriented economy to service-oriented economy and the move from focusing on software/system development to addressing business-IT alignment. Technically speaking, Cloud Computing enables to build Cloud Service-Based Application (CSBA) which cater for the tailoring of services to specific business needs using a mixture of SaaS, PaaS and IaaS solutions - possibly from various providers. In other words, in the context of CSBAs, cloud services are rented by clients from providers instead of owning the services. Due ti this specific nature, SLA (Service Level Agreement) has become a very important and up-to-date issues in CSBAs. Therefore SLA turns to be critical for both cloud service clients and providers and needs constant monitoring for various reasons mostly detecting if any violation happens but also preventing the violation in efficient way. As in CSBAs a number of providers are involved, it is a challenge to detect and resist violations of multiple SLAs that engage different providers form different locations. To deal with such a problem, this paper introduces a framework called PAEAN4CLOUD. The framework comprises components for monitoring, detecting, and configuring SLAs. An algorithm is proposed for automatic detection of SLA violations. The configuration component underpins assembling CSBAs automatically at runtime. The components help in preventing SLA violations and optimizing application performance as well.

1 INTRODUCTION

Until today, the cloud computing paradigm has given the rise to a wide variety of services that are grouped and called Everything as a Service (XaaS). It promotes opportunities to transform an object into a service and deliver it to users over the Internet. It enables building *cloud Service-based application (CSBA)* that relies on cloud services such as SaaS, PaaS, IaaS for designing, implementing, deploying, executing, monitoring, managing applications. CSBAs are highly customizable and allows building applications by localizing the services for satisfying the very specific needs of a context *e.g.* organization or individual. CSBAs pose various challenges. The major challenge is *detecting* and *managing* different types of violations of service level agreement (SLA). SLA is a predominant entity in CSBAs. In CSBAs, clients rent services from providers instead of *buying services*. This means that a CSBA is composition of list of rented service. Thus, SLA becomes critical to both service clients and providers and it needs to be monitored constantly while a CSBA is running to detect violation and prevent the violation. As in CSBAs a number of providers are involved, detecting and resisting violations (of multiple SLAs that engage different providers form different locations) are enormously challenging. Therefore, an efficient and effective approach is paramount importance for CSBAs.

Monitoring and managing SLAs for service based applications (SBAs) have documented in various bodies of literatures, nonetheless, the solutions proposed in literature are not the best fit to CSBAs since the development landscape of these applications are different. This implies that the existing approaches cannot be reused in cloud environment. In addition, the state of the art lacks an efficient approach for detecting and resisting SLA violations in cloud-based environment.

The goal of this research is to address this shortcoming. To this end, a framework called

PAEAN4CLOUD is proposed in this paper. The framework comprises a list of components for monitoring, detecting, and configuring CSBAs. An algorithm is proposed in this paper for automatic detection of SLA violations. The configuration component enables assembling CSBAs automatically at runtime. It assists in preventing SLA violations and optimizing application performance as well. The remainder of this paper is organized as follows. Section 2 presents a motivating scenario. Section 3 describes the configuration and deployment of CSBAs. The patterns of SLA violation is explained in section 4. The PAEAN4CLOUD framework is described in section 5. Section 6 presents the algorithms that automate the violation detection and reconfiguration of CSBA. The related works is described in section 7. The conclusion is drawn in section 8.

2 MOTIVATING SCENARIO

This section illustrates a simple scenario of today's industrial reality. This scenario is an extension of Taxi Application Scenario that was originally co-developed with several IT companies like Telefonica, Telecom Italia, Ericson, 2ndQuadrant and SAP for the EC's 4CaaSt project (European Comission: Information & Communication Technologies Unit, 2010). The scenario promotes a comprehensive approach to exist so that the design, configuration, deployment, monitoring of SLA, and adaptation of a CSBA can occur.

`TaxiTilburg` is an application service provider that offers the taxi ordering application *TaxiOrdering-CSBA* in the form of an end-to-end process, as shown in Fig. 1. The process is composed of several steps including *Receive Order*, *Allocate Taxi*, *Calculate Route*, *Confirm Order*, and *Reject Order*. In order to implement the steps S1 (Receive Order), S4 (Confirm Order), S5 (Reject Order) of the TaxiOrdering-CSBA process, TaxiTilburg has developed an in-house SMS software component that is able to receive new orders by sms, and send order confirmations or rejections to customers. Unfortunately, due to lack of expertise, TaxiTilburg has to rely on third-party SaaS offerings for managing its taxi fleet and for calculating routes, which are the two necessary functionalities to implement step S2 (Allocate Taxi) and step 3 (Calculate Route) in its TaxiOrdering-CSBA process. These two required functionalities are captured as the two functional requirements *Fleet Management Service* and *Map Service*.

Figure 1 also shows the Quality-of-Service (QoS) offerings $(responseTime(s) \leq 15) \& (throughput(req/s) \geq 100)$ that TaxiTilburg has promised to its customers. To maintain this predefined QoSs, TaxiTilburg has distributed the QoS value of maximum response time to each step contained in TaxiOrdering-CSBA process. The QoS requirement of minimum throughput 100 req/s remains same for all steps, as shown in Figure 1.

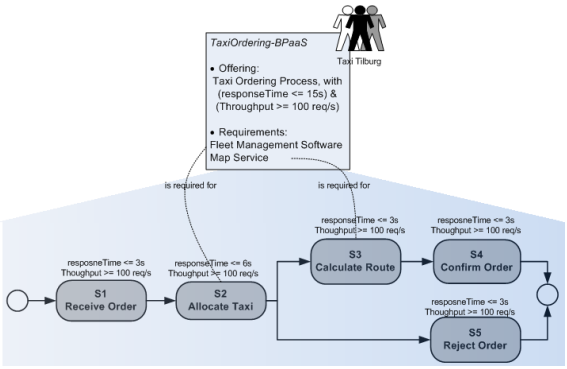

Figure 1: The process definition of TaxiOrdering-CSBA.

2.1 Configuring the CSBA

In order to deploy the TaxiOrdering-CSBA on the cloud, it is necessary to satisfy all the functional and QoS requirements of the TaxiOrdering-CSBA. We follow the *Blueprint Approach* to configure the TaxiOrdering-CSBA, so that all of its functional and QoS reuqirements can be correctly fulfilled. The concept "Blueprint" has defined in (Nguyen et al., 2012a) as "an uniform abstract description of a cloud service, i.e. SaaS, PaaS, or IaaS, that facilitates the CSBA developers with the selection, customization and composition of cross-layered cloud services". Depending on the abstraction layer of cloud services, there are three different types of blueprints *SaaS blueprint*,*PaaS Blueprint*, and *IaaS blueprint*. Figure 2 illustrates the configuration of TaxiOrdering-CSBA using the blueprint approach. There is a SaaS blueprint describing the `TaxiOrdering-CSBA` with its offering and requirements. The blueprint composition approach (Nguyen et al., 2012b) subsequently queries the marketplace repository to retrieve appropriate source blueprints whose offerings can fulfill the requirements of the `TaxiOrdering-CSBA` blueprint. The newly retrieved source blueprints may also contain requirements, e.g. the `FleetMgt-SaaS` blueprint. Hence, this querying and matchmaking task has to be executed iteratively until the entire configuration of the `TaxiOrdering-CSBA` no longer contains any further requirement.

The matchmaking between an offering and a requirement does not only take into account the func-

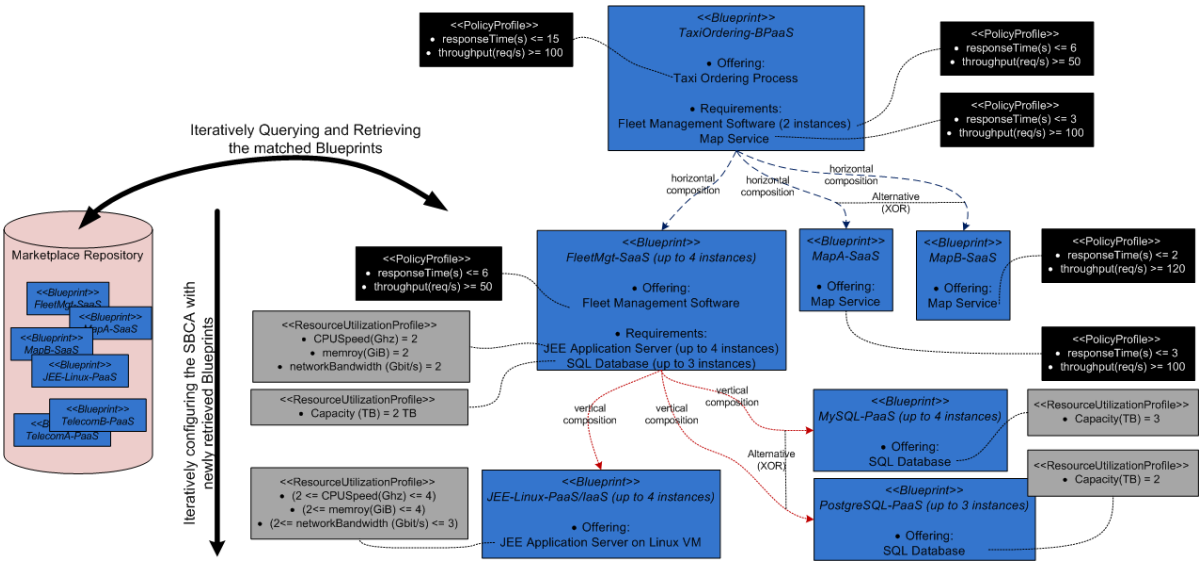

Figure 2: Blueprint Configuration for the Taxi Ordering CSBA.

tional matchmaking, but also the QoS matchmaking, i.e. whether the QoS properties specified in the $<<$ *PolicyProfile* $>>$ of the requirement can be satisfied by the QoS properties specified in the $<<$ *PolicyProfile* $>>$ of the offering. For instance in Figure 2, the two blueprints `MapA-SaaS` and `MapB-SaaS` are selected to fulfill the requirement `Map Service`, since their QoS properties regarding the maximum response time and minimum throughput satisfy the QoS properties specified for the requirement `Map Service`. Furthermore on the PaaS/IaaS level, matchmaking is also based on the resource provisioning specification specified in the so-called $<<$ *ResourceUtilizationProfile* $>>$, e.g. in Figure 2 the `FleetMgt-SaaS` blueprint has some resource provisioning properties specified for its requirement `JEE Application Server` regarding the required CPU speed, memory size and network bandwidth, and these properties can be satisfied by the resource provisioning properties of the `JEE-Linux-PaaS/IaaS` blueprint.

It is worth noting in Figure 2 that there could be alternative matchmaking results for fulfilling a requirement, e.g. for fulfilling the requirement `Map Service` of the `TaxiOrdering-CSBA` blueprint, and for fulfilling the requirement `SQL Database` of the `FleetMgt-SaaS` blueprint. The QoS properties of the alternative matchmaking results are different, which implies that there might exist alternative configurations for the `TaxiOrdering-CSBA` that could trigger a reconfiguration process in case there are some changes in the QoS requirements of the `TaxiOrdering-CSBA`.

The next step after configuring an CSBA is to select among the alternatives the optimum blueprint configuration that will be deployed on the physical cloud infrastructure. The selection strategy might depend on a variety of criteria such as the cost, licensing issues, or some private business constraints. For brevity reason we do not discuss the criteria for selecting a blueprint configuration in this paper. In Figure 2, a configuration has been chosen that involves the `MapA-SaaS` blueprint to fulfill the requirement `Map Service` of the `TaxiOrdering-CSBA` blueprint, and the `MySQL-PaaS` blueprint to fulfill the requirement `SQL Database` of the `FleetMgt-SaaS` blueprint.

2.2 SLA Specification

Each cloud service provider involved in the `TaxiOrdering-CSBA` configuration in Fig.2 promises to satisfy the stipulated Qualities of Services(QoS) through a Service Level Agreement (SLA) with his consumer. Hence, as soon as a blueprint configuration is finalized, SLAs are generated among all the involved cloud service providers. On the SaaS layer, the SLAs mainly contain the promised maximum response time of handling a request and the minimum throughput (in terms of number of request per seconds). On the PaaS layer, the PaaS providers mainly specify in their SLAs the *number of instances* of their offerings that can be provided to a single user, and the resource capacity of their offerings including for instance the *CPU speed* (in GHz), *memory* (in GiB), *networkBandwidth* (in Gbit/s), and storage *capacity* (in TB). Table 1 summarizes the SLA specifications of all the cloud service offerings in the scenario.

Table 1: SLA specification of Cloud Services.

Cloud Service	SLA specification for consumers
TaxiOrdering-CSBA	$(responseTime(s) \leq 15) \& (throughput(req/s) \geq 100)$
MaMapA-SaaS	$(responseTime(s) \leq 3) \& (throughput(req/s) \geq 100)$
MapB-SaaS	$(responseTime(s) \leq 2) \& (throughput(req/s) \geq 120)$
FleetMgt-SaaS	$(numberOfInstance \leq 4) \& (responseTime(s) \leq 6) \& (throughput(req/s) \geq 50)$
MySQL-PaaS	$(numberOfInstance \leq 4) \& (capacity(TB) = 3)$
PostgreSQL-PaaS	$(numberOfInstance \leq 3) \& (capacity(TB) = 2)$
JEE-Linux-PaaS	$(numberOfInstance \leq 3) \& (2 \leq CPUSpeed(Ghz) \leq 4) \& (2 \leq memory(GiB) \leq 4) \& (2 \leq networkBandwidth(Gbit/s \leq 3)$

2.3 Deploying the CSBA

In the next phase, a deployment architecture has to be developed based on the selected optimum blueprint configuration. The purpose of this deployment architecture is to serve as a cookbook for cloud administrator to deploy and maintain the CSBA on the cloud infrastructure. In this architecture, the blueprints will be transformed into a set of concrete, multi-layered cloud service instances and network links that collaboratively form the deployment topology of the CSBA. Figure 3 illustrates the deployment architecture for the `TaxiOrdering-CSBA`, in which:

- Two instances of the `MySQL-PaaS` should be deployed
- Two instances of the `FleetMgt-SaaS` blueprint should be deployed on two instances of the `JEE-Linux-PaaS/IaaS`.
- A network link should be deployed to connect the two `MySQL-PaaS` instances with the two `JEE-Linux-PaaS/IaaS` instances.
- One instance of the `MapA-SaaS` should be deployed
- A network link should be deployed to connect the two `FleetMgt-SaaS` instances with the `Map-SaaS` instance and with all the prospective `TaxiOrdering-CSBA` instances.

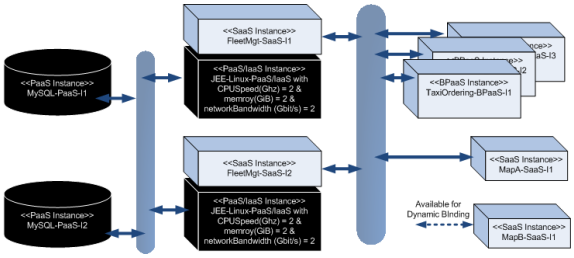

Figure 3: Deployment Architecture of the Taxi Ordering CSBA.

Maintaining the QoS levels that have been specified in the SLAs is indispensable for all the service providers providing platform service, software services, and physical resource services that are used for building the TaxiOrdering-CSBA application that will underpin the process. At runtime, a framework is required for monitoring the execution of TaxiOrdering-CSBA process steps, detecting the occurrence of violations or potential violations of QoS constraints, and preventing these violations. PAEAN4CLOUD is proposed for addressing these requirements as an efficient solution is missing.

3 PATTERNS OF SERVICE LEVEL AGREEMENT VIOLATION

In this section, we describe the Patterns of SLA Violation (PSLAV) that occur in workstreams while a CSBA is running. Additionally, the formal semantics of these patterns is provided in a lightweight style using First Order Logic (FOL). The key purpose of discovering and formalizing these patterns is to make the PAEAN4CLOUD aware of SLA violations and enable it for detecting SLA violations automatically. In addition, the patterns enable PAEAN4CLOUD detecting and preventing SLA violations proactively and efficiently. It is worth noting that, the violations reported in this paper have discovered through a study on the cloud SLA samples provided in various literature e.g. (Wu and Buyya, 2010), (Emeakaroha et al., 2012a), and (DeveloperWorks, 2010).

Before defining the PSLAVs, the concepts SLA and violation of SLA are defined.

Definition 1. *Let 'Σ' be the SLA that contains a list of quality constraints 'Q^C', security constraints 'S^C', privacy constraints 'P^C', and regulatory constraints 'R^C' such that,*

$$\Sigma := (Q^C \wedge S^C \wedge P^C \wedge R^C)$$

Since CSBA relies on third party cloud service providers, an SLA involves a 'consumer' and one to

many 'providers'. For example, a client "X" buys infrastructure services from Amazon[1], platform service from google [2], and software service from SalesForce [3]. These providers must satisfy the obligations they promise to the client "X".

Definition 2. *A violation 'Γ' indicates either a cloud service consumer or a provider fails to satisfy one or more constraints contained in the agreement.*

The SLA violation patterns are classified into QoS constraint violation patterns, security constraint violation patterns, privacy constraint violation patterns, and regulatory constraint violation patterns. The term 'constraint' refers to obligations related to QoS, privacy, regulatory, and security which are stipulated and must be satisfied by the involved service providers. These patterns are described in the following. Notably, this paper covers the well-understood SLA violation patterns only.

- **QoS Constraint Violation Patterns.** Quality constraints violation is denoted by 'Γ_Q^C'. In relation to QoS[4], the following violation patterns are observed.
  - *Reliability Violation Pattern.* $\forall CSBA,\ \exists\Gamma_Q^C$ *if* $(Availability(X^S)) \rightarrow \neg satisfied$, where X^S denotes any cloud service. If availability of a service is not at *agreed level* then quality constraint is violated and the service is not reliable any longer.
  - *Throughput Violation.* $\forall(CSBA),\ \exists\Gamma_Q^C$ *if* $(Throughput(X^S)) \rightarrow \neg satisfied$. In this pattern, throughput of infrastructure, platform, and software services are not met by providers.
  - *Response Time Violation.* $\forall(CSBA),\ \exists\Gamma_Q^C$ *if* $(T_R(X^S)) \rightarrow \neg satisfied$. In this pattern, service providers fail to satisfy response time threshold promised to consumers.
  - *Processing Time Violation.* $\forall(CSBA),\ \exists\Gamma_Q^C$ *if* $(T_P(X^S)) \rightarrow \neg satisfied$. In this pattern, providers fail to meet the processing time threshold.
  - *Delay Time Violation.* $\forall(CSBA),\ \exists\Gamma_Q^C$ *if* $(T_d(X^S)) \rightarrow \neg satisfied$. If the maximum delay threshold is not satisfied then it results quality constraint violation.
- **Security Constraint Violation Patterns.** 'Γ_S^C' represents security constraints violation. The security constraint violation patterns are listed below.
  - *Data Durability Constraint Violation.* $\forall(CSBA),\ \exists\Gamma_S^C$ *if* $(D^d(D^s)) \rightarrow \neg satisfied$ where, $D^s \in P^s$. D^s, P^s and D_d^C represent database service, platform service, and data durability respectively. The data durability violation happens in database services. The violation happens if a database service provider fails to persist data of CSBA users. For database service, durability is an important issue.
  - *Breaching Data Confidentiality.* $\forall(CSBA),\ \exists\Gamma_S^C$ *if* $(D_c^C(D^s)) \rightarrow \neg satisfied$, where D_c^C represents data confidentiality. If confidentiality of a data is breached then it is treated as data confidentiality violation.
  - *Breaching Data Integrity.* $\forall(CSBA),\ \exists\Gamma_S^C$ *if* $(D_I^C(D^s)) \rightarrow \neg satisfied$, where, D_I^C denotes data integrity constraint. Data integrity refers to consistent and accurate data. Data integrity violation happens if a database service provider fails to maintain desired level of data consistency and accuracy.
- **Privacy Constraint Violation Patterns.** $\forall(CSBA),\ \exists\Gamma_S^C$ *if* $(P^C(D^s)) \rightarrow \neg satisfied$, where P^C denotes the privacy constraint and 'Γ_P^C' privacy constraints. In this violation, the providers fails to isolate client's data.
- **Regulatory Policy Violation Patterns.** 'Γ_R^P' denotes the violation of regulatory policies related to cloud services. The list below shows the patterns of the regulatory policy violations.
  - *Data Retention Policy Violation.* $\forall(CSBA),\ \exists\Gamma_R^P$ *if* $(D_R^P(D^s)) \rightarrow \neg satisfied$, where '$D_R^P$' represents data retention policy. In this violation, providers fail to comply with the data retention policies.
  - *Data Deletion Policy Violation.* $\forall(CSBA),\ \exists\Gamma_R^P$ *if* $(D_D^P(D^s)) \rightarrow \neg satisfied$, where '$D_D^P$' is data deletion policy. If a data service provider fails to comply with the data deletion policies then it results violation.

[1] http://aws.amazon.com/

[2] https://developers.google.com/appengine/

[3] http://www.salesforce.com/eu/?ir=1

[4] Most of the QoS constraints listed in this paper have adapted from (DeveloperWorks, 2010)

4 PAEAN4CLOUD FRAMEWORK

In this section, the PAEAN4CLOUD framework is described. This architecture of PAEAN4CLOUD framework is shown in Fig. 4.

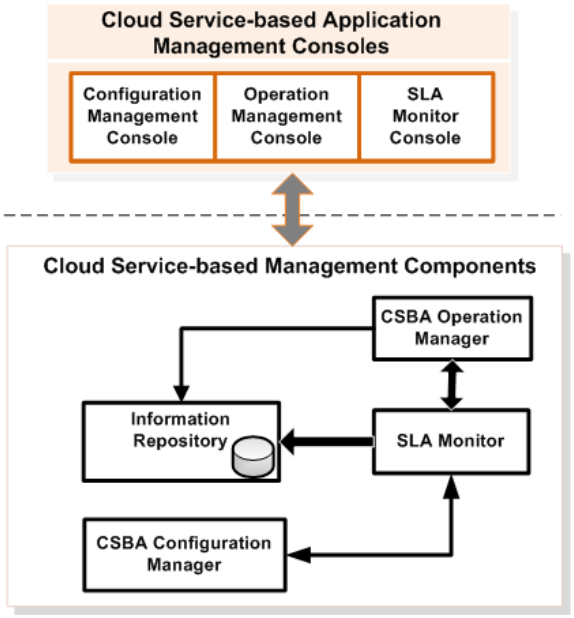

Figure 4: The Architecture of PAEAN4CLOUD.

The architecture comprises the front-end management console and Back-end management components. The console assists end-users in managing and monitoring SLA violations and the back-end management components automate the management and monitoring tasks.

The console contains four *buttons* that cater for performing management tasks. The management console relies on SOAP/RPC protocol to communicate with the back-end components that essentially perform management tasks.

The back-end consists of CSBA configuration manager, CSBA operation manager, SLA monitor, SLA violation manager, and a repository. These components are described briefly in the following,

- *CSBA Configuration Manager.* This component is used for (re-)configuring the cloud services that are incorporated in CSBAs. The configuration manager contains blueprints of CSBAs. An user carries out the modification (if necessary) and the configuration manager that in essence is a component automates these changes. The special purpose of this component is reconfiguring CSBA *on the fly* at runtime for preventing SLA violations.
- *CSBA Operation Manager.* CSBA operation manager is an application management component that manages operations performed at runtime by CSBAs.
- *Information Repository.* While an instance is running, several operations are performed. The operation logs are archived in information repository so that the history of the operations can be used by other components. The repository stores monitring information as well.
- *SLA Monitor.* This component observes the running instances. The roles of the SLA monitor are as follows:
  - watching operation cycles
  - storing monitoring information in the repository
  - detecting violations of SLA
  - reporting violations to the SLA violation manager

  In this paper, we develop an algorithm (presented in 5) for detecting the SLA violations. The SLA monitor relies on this algorithm. The proposed algorithm is designed using the SLA violation patterns defined in section 3.
- *SLA violation manager*: The primary role of the SLA violation manager is preventing violations. The violation manager communicates with SLA monitor that detects violation and sends report to violation manager that acts to prevent the violation. In addition, it communicates with repository and fetch necessary information. A component called *performance analyzer* is integrated with SLA violation manager. The performance analyzer carries out analysis on the history of the instances, predicts the potential violations. Based on this prediction, the violation manager takes necessary actions in particular, reconfigure CSBAs to resist the potential violations proactively. The next section describes reconfiguration of CSBA.

5 DETECTING SLA VIOLATION AND RECONFIGURING CSBA

The SLA monitor of PAEAN4CLOUD relies on the algorithms proposed in this paper for automated reasoning with CSBAs. These algorithms are presented and discussed in this section. Additionally, the reconfiguration of CSBA is discussed here.

5.1 Detecting SLA Violation

The violation detection algorithms are the key contributions of this paper. Algorithm 5.1 and 5.2 are designed for automated reasoning with CSBA to detect SLA violations. It is worth noting that privacy and regulatory policy violations out of the scope.

In this section, the detection of violations is discussed using the TaxiOrdering-CSBA scenario (described in section 2). The TaxiOrdering-CSBA is configured by integrating a business process service, software services, platform services, and infrastructure service that are provided by different vendors. The configuration of CSBA is discussed in Section 2.1.

The central focus of TaxiOrdering-CSBA scenario is the SLA between the client *TaxiTilburg* and the cloud service providers include BestHosting, TelecomProvider A and B, Auto Inc., and MapProvider A and B. As TaxiTilburg is yet another service provider for the end-users, the quality of the services promised by the service providers is critical to TaxiTilburg. Besides, the data security is always crucial for the client TaxiTilburg.

Upon receiving a request placed by the customer 'X', a process instance is created. For this instance, the process execution starts with activity *Allocate Taxi* (S_2). Then, the SLA monitor is called and software services are invoked if they are available. If a required software service is not *available*, the monitor detects the unavailability and reports it to the SLA manager.

For S_2, the *expected values* for the QoS parameters *responsetime* is *6 seconds* and *expected throughput* is 100 requests per second. However, the *actual value* that is taken for response time is *8 seconds* which is greater than the expected value. SLA monitor detects this violation of quality constraints and reports it to SLA Manager (See section 4 for detailed description of the SLA manager). SLA manager prevents the violation of *response time* of *Taxi Ordering Process*.

The *response time* violation of *Taxi Ordering Process* results in end-client dissatisfaction that influences the customer in choosing another taxi service provider. In order to avoid such consequence, the SLA manager acts proactively based on history of completed activities.

Furthermore, for each process instance that is completed successfully, the data is saved in the database permanently. The DatabaseProviderA grantees data security in particular, the durability of the data and consistency of the database to TaxiTilburg. The SLA monitor can detect if data is lost or the database is inconsistent. Algorithm 5.2 is used for automated reasoning with TaxiOrdering-CSBA. After data of an end-client (the client who ordered for a taxi) is stored in the TaxiOrdering database, the application manager of TaxiOrdering-CSBA access the database remotely for retrieving detail of a customer. The application manager executes query and retrieve the detail of the client. If complete detail is not found in the database then the SLA monitor reports this as a violation of security constraint, to the SLA manager.

Algorithm 5.1: Quality Constraint Violation Detection (*CSBA*).

Input:
i. X^s that includes the cloud services such that,
$(B^s_P, S^s, P^s, I^s) \sqsubseteq X^s$
ii. QoS parameters P^{QoS} contained in SLA.
Output: Detect and Report the violation of the QoS constraints
Global Variables:
$\lambda \leftarrow$ cloud service based application(CSBA)
$X^s \leftarrow$ cloud services, $B^s_P \leftarrow$ business process services
$S^s \leftarrow$ software services, $P^s \leftarrow$ platform services
$I^s \leftarrow$ infrastructure services, $Monitor \leftarrow$ SLA monitor
$M^{SLA} \leftarrow$ SLA Manager, $R \leftarrow$ information repository
$\Gamma^C_Q \leftarrow$ quality constraints, $T^R \leftarrow$ response time
$T^P \leftarrow$ processing time, $D^T \leftarrow$ delay time
$Availability \leftarrow$ Service Availability, $A \leftarrow$ Process Activity
$Throughput \leftarrow$ the performance rate of services
$B^P_I \leftarrow$ Process Instance, $EValue \leftarrow$ Expected Value
$AValue \leftarrow$ Actual Value, $ERate \leftarrow$ Expected Throughput
$ARate \leftarrow$ Actual Throughput, $H \leftarrow$ Processing History
procedure (*Initialize*)
- **for each** inbound request 'R_i' to λ
- **if** ($\lambda \neq \varnothing$
  - **then** {*Call Monitor* **and** *invoke* B^S_P;
  - **else** *Cancel* 'R_i';
- **if** $B^s_P \neq \varnothing$
  - **then**
    - **if** (B^s_P.*Availability* = "Yes")
      - **then** *Instantiate* B^I_P
    - (**Comment**: a process instance B^I_P is created in this step).
      - **else** *Report* Γ^C_Q to M^{SLA} **and** *Store* H in R;
  - **else** *Cancel* 'R_i' **and** *Terminate* 'B^P_i';
- **for each** B^I_P
- *execute* ($A_i \in B^s_P$);
- **for each** A
- (**Comment:** Assume that 'A' is an operational activity
- **if** $A \neq \varnothing$
  - **then** {**invoke** (S^s **and** P^s **and** $I^s \sqsubseteq X^s$);
  - **else** *Cancel* 'R_i' **and** *Terminate* 'B^P_i';
- **if** ((S^s. *Availability* = "yes") **and**
- (P^s. *Availability* = "yes")
- **and** (I^s. *Availability* = "yes"))
  - **then**
    - (*Compute AValue* (T^R, D^T for X^s)
    - **and** *ARate* (*Throughput* for X^s)
    - **and** T^P for A);
  - **else**
    - **Report** Γ^C_Q to M^{SLA} **and**
    - *Store* H in R;
- **if** (*AValue* (T^R, D^T) > *EValue* (T^R, D^T)&&
- *ARate* (*Throughput*) > *ERate* (*Throughput*))&&
- *AValue*(T^P) > *AValue*(T^P) for A)
  - **then**
    - **Report** Γ^C_Q to M^{SLA} **and**
    - *Store* H in R;
  - **else** {**Execute** Next A;
- **for all** $A_i \sqsubseteq B^s_P$
- **repeat**
- *Check* SLA Violation
- **until** Complete the execution of B^s_P;

Additionally, the application manager checks consistency of the TaxiOrdering database either period-

Algorithm 5.2: (SecurityConstraintViolationDetection).

Input:
i. $S^s, D^s \in P^s$
ii. Security parameters '$P^{Security}$' contained in SLA.
Output: Detect and Report the violation of security constraints
Global Variables:
$S^s \leftarrow$ Software Services
$P^s \leftarrow$ Platform Services
$D^s \leftarrow$ Database Services
$Monitor \leftarrow$ SLA Monitor
$M^{SLA} \leftarrow$ SLA Manager
procedure $(Initialize)$
/ ∗ *Durability Violation Detection* ∗ /
for each *Query* to D^s
if $(D^s \neq \varnothing)$
then *Call Monitor* **and** *Run* SELECT D_i FROM D^s;
else *Cancel Query*;
if *Query* $(D^s) \rightarrow D^i$ Not Found
then *Report* Γ_S^C to M^{SLA} **and** *Store H* in *R*;
/ ∗ *Consistency Violation Detection* ∗ /
for each Database Checking '*ChkDB*' to D^s
if $(D^s \neq \varnothing)$
then *Call Monitor* **and** *Run* DBCC CHECKDB
/* CHECKDB command includes CHECKALLOC CHECKTABLE, and CHECKCATALOG commands */
else *Cancel ChkDB*;
if *ChkDB* $(D^s) \rightarrow \neg$ *consistent* D^s
then *Report* Γ_S^C to M^{SLA} **and** *Store H* in *R*;

ically or when a database transaction is happened. For any consistency checking request placed by the application manager, SLA monitor is called. The application manager access databases and run query such as, an SQL statement 'DBCC CHECKDB' on TaxiOrdering database for checking consistency. If SLA monitor detects inconsistency in TaxiOrdering database, then it reports a violation to SLA manager. It is worth noting that in this example only SQL commands are used for query database and checking consistency because the TaxiOrdering is a mySQL database. The proposed solution supports executing other commands as well since it supports integrating other database services. In essence, the query statements and consistency checking depends on the database service integrated with CSBAs.

The SLA manager calls the configuration manager - depending on the violation pattern - for reconfiguring the CSBAs. In the next section, the strategies for reconfiguring CSBAs are discussed.

5.2 Reconfiguring the Service-based Cloud Application

After being triggered by the SLA Manager, the Configuration and Deployment Manager is responsible for making an informed decision on the preventive reconfiguration strategies and actions for the BPaaS based on the violation patterns that have been detected in the previous BPaaS instances. The aim is to enable future BPaaS instances to meet the expected SLA with their clients. Figure 5 illustrates the decision tree inside the Configuration and Deployment Manager, in which each SLA Violation pattern detected in the previous section 5 is tackled by specific reconfiguration strategies, which lead to specific reconfiguration actions. In the following, each strategy and action is explained in detail:

- RS1- Find alternative SaaS: This strategy can be used when a constituent SaaS of a BPaaS is not available for invocation (*Availability Constraint Violation*) or its QoS (response time and throughput) performance is decreasing. The idea is to find an alternative substitution for this SaaS. This strategy leads to the reconfiguration action *RA1- Dynamic SaaS Binding*, which dynamically binds all the upcoming BPaaS instances to the new SaaS.
- RS2- Scale up SaaS: This strategy can be used when the response time constraint of a constituent SaaS is violated (*ResponseTime Constraint Violation*). The idea is to prevent this violation for the future BPaaS instances by resizing the infrastructure resources of the running SaaS instances, including resizing and rebooting their Virtual Machines with more computing resources (*RA2-Resizing and Rebooting VM*) and reconfiguring their network links with less latency (*RA3-Reconfiguring Network Link*).
- RS3- Scale out SaaS: This strategy can be used when the throughput constraint of a constituent SaaS is violated (*Throughput Constraint Violation*). The idea is to prevent this violation for the future BPaaS instances by deploying new instances of this SaaS (*RA4- Deploying new Instances*) and reconfiguring the network link of all the SaaS instances with larger bandwidth (*RA4-Reconfiguring Network Link*).
- RS4- Find alternative data service provider: This strategy can be used when a violation in data security and confidentiality is detected by the SLA Manager (*Data Security Constraint Violation*). The idea is to switch to a new Data service provider that provides better security mechanism and guarantee. This strategy leads to the

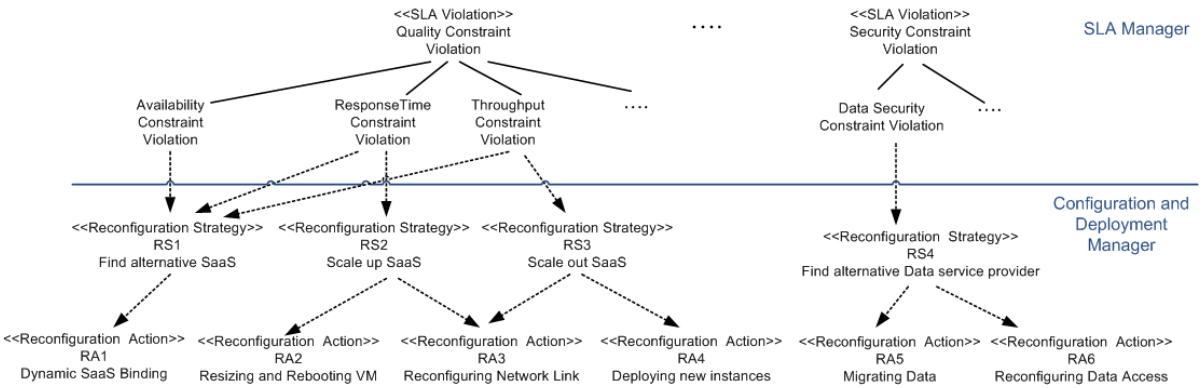

Figure 5: Reconfiguration Strategies and Actions for each SLA Violation Pattern.

process of migrating data from the old to the new provider (*RA5- Migrating Data*), and reconfiguring the Data Access mechanism (*RA6- Migrating Data*)

In case several violation patterns are detected, multiple reconfiguration strategies could be combined to perform preventive actions. For instance, in case both the response time and throughput constraint of a SaaS are violated, one could follow both RS2 and RS3 to resize the current SaaS instances with more resource and to deploy new instances of the SaaS.

In our running example in section 2, the TaxiOrdering-BPaaS is configured using the blueprint approach that results into a blueprint configuration in Figure 2. Based on the blueprint configuration, a deployment configuration has been developed in Figure 3, which serves as a cookbook for cloud administrator to actually deploy and maintain the TaxiOrdering-BPaaS, so that all the SLA constraints between the TaxiOrdering-BPaaS with its clients can be satisfied. Nevertheless, at peak time period (e.g. new year's eve), the number of instances of the TaxiOrdering-BPaaS may increase dramatically and hence lead to several SLA violations. The previous Section 5 points out the four sample violation cases that may occur to the TaxiOrdering-BPaaS.

- V1: Response time of the step (S2-Allocate Taxi) increases, i.e. the *actual response time* is 8 s whilst the *expected response time* is only 6s. This is due to the response time constraint violation of the FleetMgt-SaaS.
- V2: Throughput of the step (S2-Allocate Taxi) decreases, i.e. the *actual throughput* is only 8 req/s whilst the *expected throughput* is 10 req/s. This is due to the throughput constraint violation of the FleetMgt-SaaS.
- V3: Step (S3-Calculate Route) cannot be executed. This is due to the availability constraint violation of the MapA-SaaS.
- V4: Data security is violated. This is due to the data security constraint violation of the MySQL-PaaS.

With help of the reconfiguration decision tree, the Configuration and Deployment Manager is able to perform preventive actions for the future instances of the TaxiOrdering-BPaaS. Table 2 explains which reconfiguration strategy and action are performed for which violation cases in our running example. Please note that the details of how to perform a reconfiguration action strongly depends on the blueprint configuration of the TaxiOrdering-BPaaS introduced in Section 2.1 that provides different alternative configurations. In this paper, we do not discuss the approaches of selecting a particular configuration.

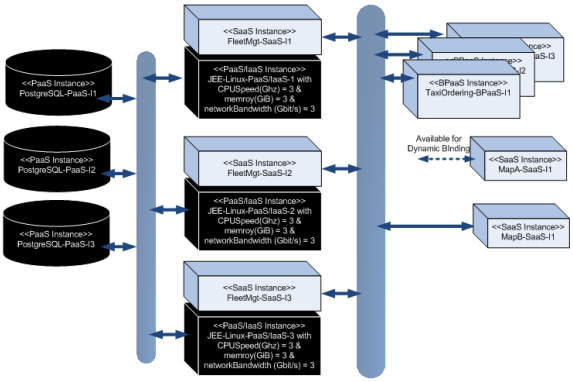

Figure 6: The deployment architecture of the reconifgured TaxiOrdering-BPaaS.

Performing the reconfiguration actions will result in a different deployment configuration of the TaxiOrdering-BPaaS. Figure 6 illustrates this adapted deployment configuration of the TaxiOrdering-BPaaS

- 3 instances of the FleetMMapB-SaaS is deployed on 3 instances of the Linux-JEE-PaaS/IaaS, each

Table 2: Reconfiguration Strategies and Actions for the sample TaxiOrdering-BPaaS.

Cloud Service	SLA Violation	Reconfiguration Strategy & Action	Details
FleetMgt-SaaS	V1: Response Time Constraint	RS2 $\Longrightarrow$ RA2	Resizing the VM with CPUSpeed(Ghz)=3, memory(GiB)=4.
FleetMgt-SaaS	V2: Throughput Constraint	RS3 $\Longrightarrow$ RA3 & RA4	Deploying another instance of the stack FleetMgt-SaaS + JEE-Linux-PaaS/IaaS. Reconfiguring the network link of all three JEE-Linux-PaaS/IaaS instances with networkBandwidth(Gbit/s) = 3
MapA-SaaS	V3: Availability Constraint	RS1 $\Longrightarrow$ RA1	Dynamically rebind to the new MapB-SaaS instance.
MySQL-PaaS	V4: Data Security Constraint	RS4 $\Longrightarrow$ RA5 & RA6	Migrate all the data of the FleetMgt-SaaS to 2 new instances of PostgreSQL-PaaS. reconfiguring the data access layer of the FleetMgt-SaaS.

with 3Ghz CPU speed, 3 GiB memeory and 3 Gbit network link

- 3 instances of the PostgreSQL databases are used by the FleetMgt-SaaS instances to store data.
- A MapB-SaaS instance used to performed the "Step 3- Calculate Route", instead of the MapA-SaaS instance.

6 RELATED WORKS

This section describes the related works in the area of monitoring and management the SLA violation of the CSBA. A number of remarkable works related to monitoring and management SLAs of the service oriented and Grid applications have been presented in many literature *e.g.* (Comuzzi et al., 2009), (Fu and Huang, 2006), (Koller and Schubert, 2007), (Boniface et al., 2007), (Wood et al., 2009). However, a very little works are found in the area of monitoring and managing SLAs of the cloud service based system. From the conceptual point of view, monitoring SLA in the area of cloud, grid and conventional service oriented domains are the same, yet the methods of monitoring SLAs of the systems serving these domains differ. Therefore, since this paper is focused on detecting the violation of the SLAs within cloud environment, the related works focusing on monitoring SLA for detecting violation and reconfiguration of cloud services is discussed in this section.

In (Emeakaroha et al., 2012b), a framework called *DeSVi* is proposed to detect SLA violation in cloud infrastructure. The goal of this framework is similar to PAEAN4CLOUD. The framework is proposed for monitor the resources and component perform casebases reasoning for detecting the violation by utilizing the notion of knowledge database. Additionally, the framework relies on the service level objective. DeSVi contains a application deployment component that facilitates deploying application automatically.

Hammad (Hammadi and Hussain, 2012) proposes framework for monitoring QoS. It comprises two modules include reputation assessment module and transactional risk assessment module. It assists in assessing performance based on QOSs observed and also assists in making decision whether or not a cloud service customer should migrate to another provider.

SageShift (Sukwong et al., 2012) consists of a Virtual Machine(VM) admission control - Sage, and a hypervisor - shift. The key focus of the SageShift is preventing SLA violations. SageShift assess the feasibility of SLA based on the incoming requests and prevent SLA violations by adjusting the required amount of resources dynamically and determining the sequence of VM execution. Unfortunately, the potential SLA violations with respect to software service and platform service cannot be traced as SageShift oversights these service issues.

(Chandrasekar et al., 2012) proposes a QoS monitor that reads data and a offers a solution that is built upon the Markov Chain [5] to establish trust between service consumer and provider.

A holistic approach called OPTIMIS was proposed by (Ferrer et al., 2012) to optimize entire service life cycle including monitoring services. The authors describe various aspects of service monitoring and managing SLAs between service provider and consumer. They stress on a list issues that need to be improved for optimizing the performance of current solutions for monitoring the cloud services. However, the authors do not offer any optimizing method. Essentially, SLA monitoring is rather a small part of the

[5] www.math.rutgers.edu/courses/338/coursenotes/chapter5.pdf

OPTIMIS approach.

Most of the SLA monitoring approaches discussed in the above focus on the QoS aspect of services. Nonetheless, PAEAN4CLOUD is able to detect the QoS as well as security constraint violations in the cloud environment. To the best of our knowledge, existing SLA monitoring solutions do not address the security constraint violations. The SLA monitor of the PAEAN4CLOUD relies on the *category-based violation* detection approach that enhance the capability of the monitoring system for identifying violations efficiently. Additionally, the category-based violation detection approach assists in deciding the most suitable reconfiguration strategy for avoiding the violation of SLAs. None of the SLA monitoring approaches discussed in the above deals with the category-based violation detection of cloud services.

7 CONCLUSION AND FUTURE WORKS

In this paper, we have highlighted the challenges of monitoring and predicting of SLA violations in cloud service based applications. Thus, we motivated emerging requirements for the monitoring, detection, and configuring of SLA violation in a real world scenario. We argued that the specific nature of cloud service based applications using a mixture of SaaS, PaaS and IaaS solutions - possibly from various providers, makes classic approaches incapable of meeting these requirements. Consequently, we proposed a PAEAN4CLOUD, a generic framework monitoring, detecting, and configuring SLA violations. The framework relies on components and algorithms for automated reasoning with CSBA for detecting SLA. Currently we are exploring the idea of root cause analysis for preventing SLA violations in such as way that monitoring can play a pivotal role to predict violations of the QoS requirements as stipulated in the SLA and to suggest (manually or automatically) some adjustment to enforce the running application to comply with the agreed-on SLA. The adjustment could be (1) to change the service infrastructure in such way, that the application will be processed and respect the SLA. Another possibility is to (2) apply self-scaling (up/out) techniques to enable/enforce the application to comply with the SLA of the application as a whole.

REFERENCES

Boniface, M. J., Phillips, S., Perez, A. S.-M., and Surridge, M. (2007). Dynamic service provisioning using gria slas. In *NFPSLA-SOC'07*. Event Dates: 17th September 2007.

Chandrasekar, A., Chandrasekar, K., Mahadevan, M., and Varalakshmi, P. (2012).

Comuzzi, M., Kotsokalis, C., Spanoudakis, G., and Yahyapour, R. (2009). Establishing and monitoring slas in complex service based systems. In *ICWS*, pages 783–790.

DeveloperWorks (2010). Review and summary of cloud service level agreement.

Emeakaroha, V. C., Ferreto, T. C., Netto, M. A. S., Brandic, I., and Rose, C. A. F. D. (2012a). Casvid: Application level monitoring for sla violation detection in clouds. In *COMPSAC*, pages 499–508.

Emeakaroha, V. C., Netto, M. A. S., Calheiros, R. N., Brandic, I., Buyya, R., and Rose, C. A. F. D. (2012b). Towards autonomic detection of sla violations in cloud infrastructures. *Future Generation Comp. Syst.*, 28(7):1017–1029.

European Comission: Information & Communication Technologies Unit (2010). 4CaaSt: Building the PaaS Cloud of the Future. Project Objectives Document.

Ferrer, A. J., Hernndez, F., Tordsson, J., Elmroth, E., Ali-Eldin, A., Zsigri, C., Sirvent, R., Guitart, J., Badia, R. M., Djemame, K., Ziegler, W., Dimitrakos, T., Nair, S. K., Kousiouris, G., Konstanteli, K., Varvarigou, T., Hudzia, B., Kipp, A., Wesner, S., Corrales, M., Forg, N., Sharif, T., and Sheridan, C. (2012). Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66 – 77.

Fu, W. and Huang, Q. (2006). Grideye: A service-oriented grid monitoring system with improved forecasting algorithm. In *GCC Workshops*, pages 5–12.

Hammadi, A. M. and Hussain, O. (2012). A framework for sla assurance in cloud computing. In *AINA Workshops*, pages 393–398.

Koller, B. and Schubert, L. (2007). Towards autonomous sla management using a proxy-like approach. *Multiagent and Grid Systems*, 3(3):313–325.

Nguyen, D. K., Lelli, F., Papazoglou, M. P., and van den Heuvel, W.-J. (2012a). Blueprinting approach in support of cloud computing. *Future Internet*, 4(1):322–346.

Nguyen, D. K., Lelli, F., Papazoglou, M. P., and van den Heuvel, W.-J. (2012b). Issue in automatic combination of cloud services. In *ISPA*, pages 487–493.

Sukwong, O., Sangpetch, A., and Kim, H. S. (2012). Sageshift: Managing slas for highly consolidated cloud. In *INFOCOM*, pages 208–216.

Wood, T., Shenoy, P. J., Venkataramani, A., and Yousif, M. S. (2009). Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17):2923–2938.

Wu, L. and Buyya, R. (2010). Service level agreement (sla) in utility computing systems. *CoRR*, abs/1010.2881.