Supporting Cloud Service Operation Management for Elasticity*

Georgiana Copil, Hong-Linh Truong, Schahram Dustdar

{e.copil,truong,dustdar}@dsg.tuwien.ac.at
Distributed Systems Group, Vienna University of Technology

Abstract. Complex cloud services rely on various IaaS, PaaS, and SaaS cloud offerings. Fast (re)deployment and testing cycles, and the rapidity of changes of various dependent infrastructures and services, imply a need for continuous adaptation. Although software-based elasticity control solutions can automate various decisions through intelligent decision-making processes, in many cases, such adaptation requires interactions among different cloud service provider employees and among different providers. However, decisions from stakeholders and elasticity software controllers should be seamlessly integrated.
In this paper, we analyze the needs of service providers and the possible interactions in elasticity operations management that should be supported. We focus on interactions between service provider employees and elasticity controllers, and propose novel interaction protocols considering various organization roles and their concerns from the elasticity control point of view. We introduce the elasticity Operations Management Platform (eOMP) which supports seamless interactions among service provider employees and software controllers. eOMP provides elasticity directives to enable notifications for complex elasticity issues to be solved by service provider employees, and the necessary mechanisms for managing cloud service elasticity. Our experiments show that service provider employees can easily interact with elasticity controllers, and, according to their responsibilities, take part in the elasticity control to address issues which may arise at runtime for complex software services.

1 Introduction

A service deployed in the cloud can make use of various resources and services offered by cloud providers, and can be very dynamic at run-time. The cloud is one of the most dynamic environments: providers can change their cost schemas, and the offered service characteristics, from one day to another. Although in this environment automated controllers are necessary, given this high dynamism, it might be necessary for service stakeholders to re-examine the desired behavior of a cloud service, and their interactions with other stakeholders (e.g., cloud providers, or data providers). For instance, whenever the load of the cloud service

*This work was supported by the European Commission in terms of the CELAR FP7 project (FP7-ICT-2011-8 #317790).

dramatically changes, the normal "safety requirements" (e.g., do not exceed a specific cost value) might not hold. For these situations, service providers have employed analysts to oversee the control process, and detect when such a case is encountered by a controller. A much better solution would be that the controller itself notifies the responsible person with the encountered situation (e.g., unexpected behavior or service health issues). In such situations, the employees responsible for the detected cases, once notified, should be able to easily interact with the controller to solve the detected issues. This type of "human-in-the-loop" based control not only improves the runtime elasticity customization capabilities, but also empowers service providers with more control over their services and automated elasticity controllers. Thus, for obtaining service elasticity at runtime, the service provider needs this kind of support in its operation phase (i.e., from service management lifecycle), in order to carry out the service operation processes in the cloud environment. Although currently several solutions provide elasticity control of cloud services (e.g., [1], [2]), service provider employees are not included in the elasticity control process.

For addressing the challenges above, in this paper we propose adding *roles* (i.e., service provider employees) as first class entities in cloud service elasticity control loops. Based on the roles, we define necessary interaction protocols for managing service elasticity operation phase. We focus on interactions between roles and elasticity controllers, but we also support simple interaction among employees, for notifying each other of updates or for delegating responsibility for incidents. We extend SYBL [3], a language for expressing elasticity requirements, to support roles and role-based communication between stakeholders. Based on this, we develop an elasticity Operations Management Platform (eOMP) for cloud services, and we validate its usefulness showing various events encountered for a complex service. eOMP can adapt to various organization structures, and enables service provider employees to interact easily with the elasticity controller, for obtaining a more complex elasticity control. These interactions are real-time, reason for which great care needs to be given to the controller, which should be able to perform normally without human intervention. eOMPs supports managing unexpected situations, by facilitating the collaboration between elasticity controllers and service employees, identifying various types of events occurring during operation phase, and providing mechanisms for solving them.

The paper is organized as follows: Section 2 presents the motivation of our work. Section 3 discusses the interactions and roles necessary for elasticity operations. Section 4 describes the design of our platform and the interactions surrounding which the platform is designed. Section 5 presents eOMP case studies, Section 6 describes related work, and Section 7 concludes the paper and outlines our future work.

2 Motivation

For managing cloud services, a service provider needs to prepare its employees for issues (e.g., errors appearing at system, application or infrastructure level, or change in cloud provider offerings) that could appear in cloud environments,

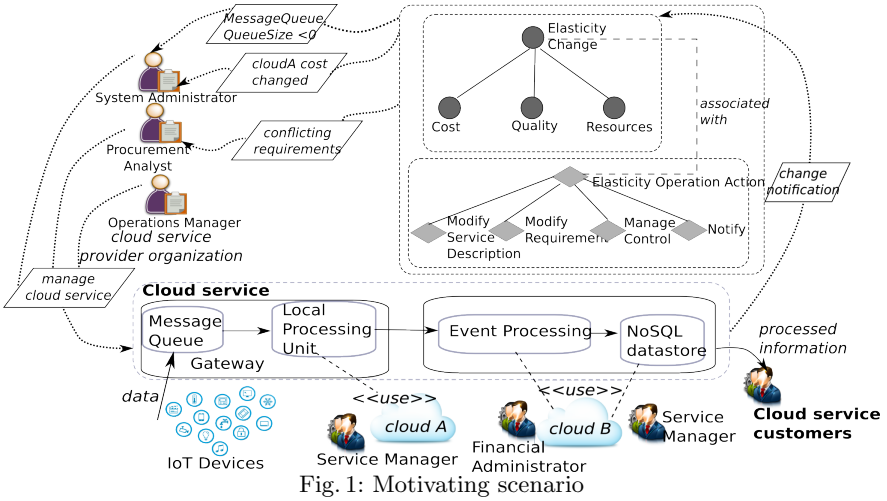
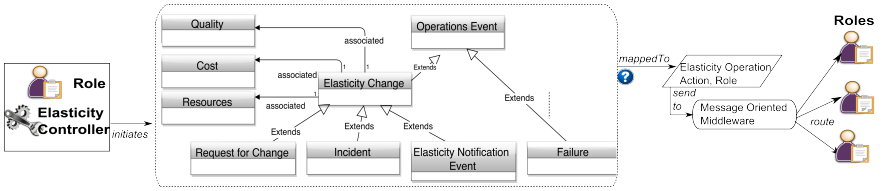


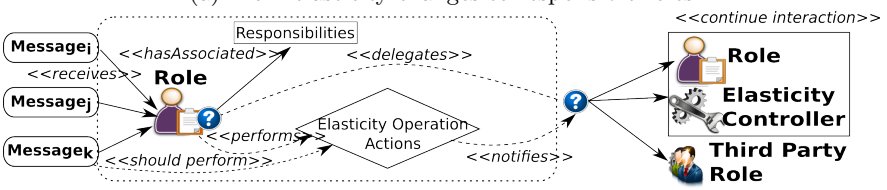
Fig. 1: Motivating scenario

and to adapt their internal processes. Moreover, given cloud environment's dynamism, it might be necessary to constantly analyze the service at runtime, to ensure that the service customers receive the expected quality of service. For this, it would be necessary to use an automated elasticity controller (e.g., rSYBL [4]), rapidly reacting to environment changes. The elasticity controller can release from the employees responsibilities, but might cause change in other responsibilities, by including the elasticity controller in the service operation management. Fig. 1 shows a complex environment ranging from IoT-specific resources (i.e., sensors, actuators) up to the cloud environment, where information gathered by sensors is stored and processed. In cloud infrastructures, we consider a complex cloud service using various types of software (e.g., NoSQL databases, RabbitMQ), processing data coming from sensors and exposing it to various service customers (e.g., fire department, or building management application). In this paper, we consider cloud service elasticity as being controlled at multiple levels of abstraction, following the cloud service model presented in [4], where the cloud service is composed of units (i.e., elementary parts of the service), which can be grouped into topologies (i.e., semantically connected units).

Focusing on the service provider, we can see that it needs to fulfil various types of requirements (e.g., data placement for IoT device users, or providing expected quality for service customers). While it is obvious that nowadays decision-making solutions should be as much software-based as possible, given the complexity of the setting, there are situations in which it is necessary for the decisions to be taken by real persons playing various *roles* in the organization. Depending on the type of change which has appeared in the service (i.e., following elasticity dimensions: resources, cost, quality [5]), various roles can have different interests or responsibilities in the cloud service control. For instance, whenever the cost of running the platform in the cloud gets high, a *financial administrator* might analyze the overall evolution, and decide whether as a strategic decision it makes sense to use more virtual resources than initially estimated. For this, s/he could



(a) From elasticity changes to responsible roles



(b) From roles to elasticity operation actions

Fig. 2: Role interaction flow

either invest more in the platform, or negotiate with *cloud providers* for better prices. Although most times service elasticity is achieved simply by allocating more/less virtual resources, similar effects might be achieved by changing configurations (e.g., changing the load balancing mechanism). In this case, employees in charge with *configuration management* should know which configuration is being used. The service provider's goals can also evolve in time, due to varying number or types of users. In our scenario, when adding further data providers for the service, the control requirements need to be modified by the responsible employees, e.g., ensure better data transfer, even though the cost would go over what before was specified as the maximum admissible cost.

Therefore, although an automated solution is necessary for this case, the Level of Automation (LOA) [6] of the elasticity controller should not be of *Full Automation*, but the human (service provider employee) should be included in the control loop, for supporting the cases discussed above, in a *Supervisory Control* mode. Moreover, for achieving this kind of interaction between the elasticity controller and the different types of service provider employees, clear interaction protocols are needed. Since existing solutions mainly focus on full automation ([2], [1]), we propose using supervisory control for cloud services elasticity, and define interaction protocols for elasticity control. Thus, we introduce a platform easing service provider's interaction with the elasticity controller, at multiple levels of authority and for multiple elasticity concerns.

3 Analyzing Interactions in Elasticity Operations Management

3.1 Role interactions

As described in our motivation, the main focus of our work is supporting service providers to achieve better supervised elasticity control. Different service provider employees are normally in charge with different operations, thus being associated with various roles as part of the organization. We use the term *roles* instead of stakeholders or employee types, to indicate attributions associated to employees/stakeholders, an employee or stakeholder possibly having multiple roles at

a time. Moreover, while the roles present in a company are rarely dynamic, the stakeholders/employee types and persons in a company are both volatile and dynamic. Following our motivation scenario, the roles and elasticity controller need to collaborate in order to manage service operation during runtime. As shown in Fig. 2a, roles should be notified by other roles or by the elasticity controller concerning the operation events which occur in relation to the current service. From the operation events, we focus on elasticity changes. We characterize elasticity changes according to the elasticity dimensions (i.e., resources, cost, and quality), and focus on: (i) request for change (RFC), (ii) incident, and (iii) elasticity notification event. The request for change event can be initiated by either an elasticity controller or a role, requesting for changes in properties of the service. The elasticity capabilities (i.e., changes which can be enforced at runtime for modifying service behavior) can incur unexpected behavior in quality, cost or resources, which can indicate an issue in the service configuration or the deployed artifacts. Moreover, service providers are interested in failure events, in order to be able to learn from service behavior and environment changes which produce failures. As shown in Fig. 2b, a role can receive a multitude of messages from other roles or from elasticity controllers. Analyzing them, the role decides whether it can perform needed actions, or if it should delegate to other roles. After analyzing the interaction flow, the next section is focused on analyzing the roles and their responsibilities and interests in elasticity operations management.

3.2 Elasticity operations and roles

In the case of elastic cloud services the elasticity controllers play a big role in management, as opposed to ordinary service management roles [7]. Lower level authority roles (i.e., system administrator) have limited responsibilities, supervising and delegating work to the elasticity controller. We focus on designing interaction between elasticity controllers and several roles[1], excluding the roles whose functionalities are replaced by the elasticity controller(e.g., Performance/-Capacity Analyst, Systems Operator, or Capacity Manager).

Table 1 shows the possible elasticity-driven behavior modifications which can be triggered by the elasticity controller, and the roles which might be interested in these types of modifications. Depending on the frequency of modifications, the roles are interested of receiving events more or less often, events being aggregated from a number of modifications, or containing a single modification. For instance, cost-related modifications need to be viewed by finance-related roles, like IT Financial Manager, Procurement Analysis, or Service Manager. Quality-related events (e.g., service part is not healthy, requirements not fulfilled) are of interest for Operations Manager and Service Manager.

4 Elasticity Operations Management Platform

We design our platform (Fig. 3) following the flow described in the previous section, for supporting interactions between roles and elasticity controllers, and even third party roles. The controller sends messages through an *Embedded Queue*, for

[1] http://www.itsmcommunity.org/downloads/ITIL_Role_Descriptions.pdf

Modification type	Roles interested
Changes in cost due to scaling/changing the infrastructure/software services which are being used	IT Financial Manager, Service Manager
Changes in cost due to providers change in cost, without change in performance	Procurement Analysis, Service Manager, Operations Manager
Changes in quality due to providers change in quality, without change in cost	Service Manager, Operations Manager
Requirement on cost inflicts degradation of performance	Operations Manager
Configuration change due to change of the workload	Configuration Librarian, System Administrator, Operations Manager
Service part which is not healthy (erroneous, not behaving as expected)	Test Manager, Operations Manager, System Administrator, Incident Analyst
Changes in quality w/o change in workload	Operations Manager, Service Manager
Requirements which are not fulfilled/ are on danger of not being fulfilled by the cloud service elasticity controller	Operations Manager
Data compliance requirements changed by the data provider	Service Manager, Operations Manager

Table 1: Examples of elasticity modifications and roles interested

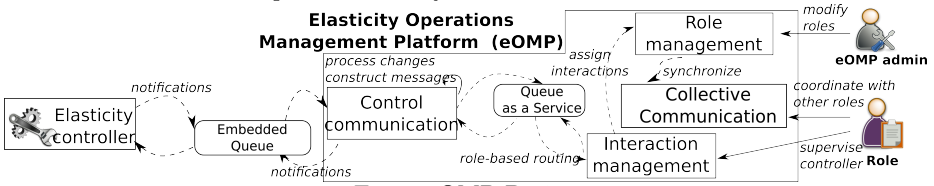


Fig. 3: eOMP Design

message locality reasons. From our elasitcity Operations Management Platform (eOMP), the *Control Communication* component processes the received notifications. Here we provide a plugin-based mechanism for ensuring that eOMP can be adapted to different elasticity controllers, the only constraint being to map controller notifications to the eOMP model with operation events. The *Control Communication* component processes the operation events, and depending on their types, and depending on the responsibilities of the roles in the *Role Management Component*, maps the controller messages to correct interactions with current roles, and adds them to the queue. For this, we use a *Queue as a Service*, since depending on the number of roles, and possibly in the future, organizations, that we want to support, the scale of the queue and routing complexity can increase. Queue interactions are consumed from the roles' side by the *Interaction processing* component, which directly interacts with the role (e.g., user interface, command line, API-driven communication). For communication between roles, we use a component for collaborative interactions in human-based computing systems (e.g., SmartCOM[2]).

[2] https://github.com/tuwiendsg/SmartCom/wiki

We designed the platform in such a way that the elasticity controller is agnostic to IT service management processes, or role types. This is beneficial since it enables service providers to choose other controllers, or, for the case of very small company (i.e., one employee), to use the controller without the rest of our platform. The roles and responsibilities can be modified by the eOMP administrator, e.g., by using roles from a different standard.

4.1 Entities of the Interaction

For defining the interactions which may occur for the cloud service elasticity control, we focus on the participants in interactions. We use *organization* to describe an association with a functional structure (e.g., service provider organization). We focus firstly on modeling interactions between the service provider organization and the elasticity controller, and secondly on modeling limited interactions among organizations.

Roles (Eq. 2) are entities which are associated with responsibilities R and authority levels Aut, and which can be assigned to different employees at various points in the organization lifetime. Each organization is defined by a set of role types, which change rarely throughout organization lifetime (e.g., when adopting CMMI[3], roles change). *Third party roles* are roles from partner organizations, which are known to the current organization, and are accessible only through an internal role for various actions (e.g., notifications, or contract re-negotiations). As we are interested in elasticity control operations management, we focus on *Elasticity Responsibilities* (Eq. 2) related to elasticity dimensions [5].

$$Roles = \{(R, Aut) | R \in Responsabilities, Aut \in [min, max]\} \quad (1)$$

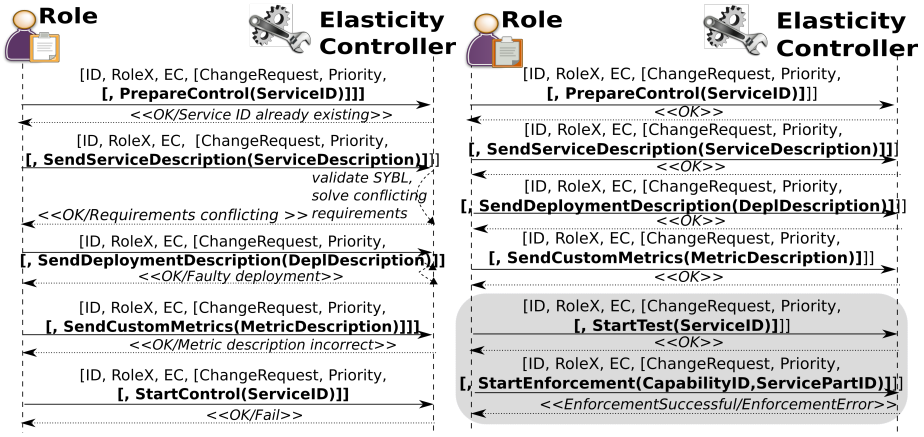
$$Responsabilities = \{x \vee Relations(x, y) | x, y \in \{Cost, Quality, Resources, Error, Analytics\}\} \quad (2)$$

We define a *Message* as being composed of a `header`, and a `body`, the header containing `initiator` and `receiver` related information, while the body contains the message `type`, its `priority` and the `content` of the message. The message content (Eq. 3) can contain suggested interactions, which are nested interactions suggested by the initiator for the receiver(e.g., an elasticity controller can suggest the Procurement Analyst to re-negotiate the contract with the cloud provider due to cost increase). Using these entities, an *interaction* can be defined as a tuple of *Initiator-Receiver-Message* (Eq. 4), where each of the Initiator and Receiver can be a set of Roles.

$$Content = \{[Cause, SuggestedMeasure] | Cause \in Requirements \cup ExpectedBehavior, SuggestedMeasure \in Interactions \cup Actions\} \quad (3)$$

$$Interaction = \{[InteractionID, Initiator, Receiver, Message] | Initiator, Receiver \in Roles, Message \in Messages\} \quad (4)$$

[3] http://cmmiinstitute.com/



(a) Interactions for Starting Elasticity Control (b) Interactions for Testing Elasticity Capability

Fig. 4: Interaction dialogs

Interaction type	Interaction details
Undeploy the service	`[ID, Role, EC, [RFC, Priority, [Cause, UndeployService(ServiceID)]]]`
Replace metric composition rules	`[ID, Role, EC, [RFC, Priority, [Cause, ReplaceRules(ServiceID, CompositionRules)]]`
Replace deployment	`[ID, Role, EC, [RFC, Priority, [Cause, ReplaceDeployment(ServiceID, DeplDescription)]]`
Replace elasticity requirements	`[ID, Role, EC, [RFC, Priority, [Cause, ReplaceRequirements (ServiceID, Requirements)]]`
Pause/Resume control	`[ID, Role, EC, [RFC, Priority, [Cause, PauseControl(ServiceID)]]`

Table 2: Interactions for requesting modification in control

4.2 Interaction protocols for supervisory control of elasticity

As discussed previously, elasticity depends on a large set of variables, both from the IoT, cloud and business world. Elasticity behavior of a service is subject to the business strategy of the service provider, and this can vary with the economic perspectives, and with the market evolution (e.g., the financial manager should decide the strategy in case of financial crisis, and not the elasticity controller). We propose using a *supervisory control mechanism* [6, 8], in which any decision of the human overrides any decision of the elasticity controller (i.e., the roles are the outer control loop). For this, we define a set of interaction protocols, based on the entities defined above, for facilitating the communication between roles and elasticity controllers.

Role as initiator - Bootstrapping Dialogs: The goal of this interaction is to enable roles to initiate dialogs with the elasticity controllers for bootstrapping the elasticity controller (Fig. 4a and 4b). For *starting elasticity control*, the elas-

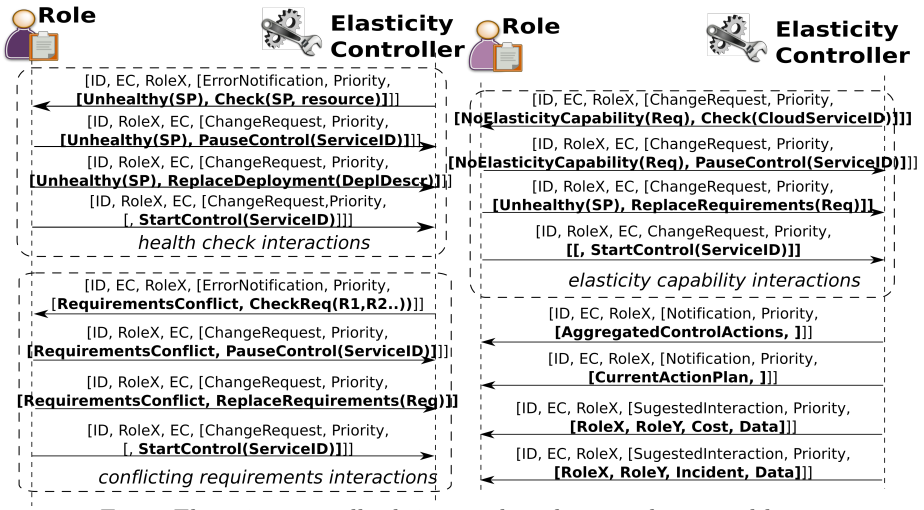


Fig. 5: Elasticity controller bringing the roles into the control loop

ticity controller is sent a `prepare` message, followed by information describing the cloud service is sent one at a time (e.g., service description, custom metrics description), and if each step is successfully achieved, a `Start Control` message is sent. Each call from the role to the dialog starts a complex process on the controller side (e.g., possible elasticity requirement conflicts are solved). For *testing an elasticity capability*, a `Start Test` message is sent for starting the test mode. For instance, the `System Administrator` is able through this dialog to set all needed information for elasticity control (e.g., structure, resources used), and then wait and ensure that each step is successfully completed.

Role as initiator - Request for change : The goal of this interaction is to enable the roles to modify expected service behavior during runtime (Table 2). Whenever a role decides that an update is necessary, e.g., due to events signaled by the elasticity controller, the role can modify elasticity requirements, or deployment description (e.g., after a manual re-deployment). For instance, the `Service Manager` can decide to undeploy the service from the cloud environment, and, e.g., keep only an on-premise deployment.

Elasticity controller as initiator: The goal of this interaction is to enable the elasticity controller to notify appropriate roles on changes in elasticity behavior (Fig. 5). Whenever abnormal changes are observed in the cloud service behavior, the elasticity controller notifies roles, depending on the *Responsibilities*, and the *Authority* which they have associated. For instance, in the case of conflicting requirements which cannot be automatically solved (e.g., response time is expected to be low, while the cloud provider is running in degraded mode), the controller notifies the roles causing the conflict (i.e., $Role_i \dots Rolej$), as well as a *higher authority role* having the responsibilities $\cup_{x=i..j} Responsabilities(Role_x)$.

4.3 Elasticity directives-driven interactions

For creating custom interactions, we have extended the SYBL [3] elasticity requirements definition language with the new `NOTIFY` directive, with BNF form described in the Listing 1.1, to be triggered when certain conditions hold. A call of the `notify()` method of the `NOTIFY` directive maps to the initiation of a new interaction between the elasticity controller and the role mentioned in the directive. An example of such a directive can be `No1: NOTIFY OperationsManager WHEN responseTime > 1.2 s : notify( WARNING, "Response time exceeds 1.2 s")`. Whenever a condition for a notification directive is true, the *Controller Communication* starts an interaction (i.e., translating the aforementioned directive into interaction `[No1, EC, Operations Manager, Notification, "Response time exceeds 1.2 s"]`. However, the frequency of interactions initiated by the *Controller Communication* is adjusted with the interaction aggregation presented in 4.4

Listing 1.1: SYBL Notification in Backus Naur Form (BNF)

```
Notification := notificationID:NOTIFY Role WHEN ComplexCondition
                        : notify(NotificationType, message)
Role := ROLE(Responsability1, Responsability2), Role |
        ROLE (Responsability1, Responsability2)|
        RoleX, Role | RoleX
NotificationType := NOTIFICATION | ERROR | WARNING
```

4.4 Interaction Aggregation

For mapping messages, we provide a generic processing mechanism which searches metric patterns associated with responsibilities in the message from the controller, and creates a new message with the structure described in Section 4.1, initiating interactions for the appropriate roles, considering roles' responsibilities. Depending on roles authorities (Equation 2), interactions are either aggregated or immediately sent to the *Interaction Management* component.

Each role, depending on its responsibilities, receives a different number of messages, or only emergency messages, the amount of messages being inversely proportional with the authority and directly proportional with the responsibilities (e.g., for a maximum Authority of 10, a Service Manager role with an authority of 10 should receive less often messages than the System Administrator with authority 5). Moreover, the nature of the messages should reflect the responsibilities and interests. For this, the *Controller Communication Module* examines messages and identifies metrics of interest for the responsibilities associated with each role. For filtering the interactions initiated, an aggregation function can be defined, selecting the amount of messages to be sent to the roles. The more complex the filtering of the messages, the easier it would be for roles to interact with the elasticity controller. We define in 5 a simple logarithmic *filtering function*, deciding if the aggregation of messages so far should be sent.

$$\begin{aligned} f(Role, QInteractions) = (\log_{auth\_max}(Role.Authority) * \\ THRESHOLD\_NOTIFICATION <= QInteractions.size()) \\ \vee(\log_{auth\_max}(Role.Authority) * THRESHOLD\_ERROR \\ <= MaxPriority(QInteractions)) \end{aligned} \tag{5}$$

5 Prototype and Experiments

5.1 Prototype

The elasticity Operations Management Platform (eOMP) is implemented as a Java enterprise application, which can be deployed either in the cloud or under service provider's premises. eOMP is open-source and available together with further experiments, details and user guides[4]. The current version integrates with the rSYBL elasticity controller, making use of the notification queue (i.e., embedded queue in the eOMP design, implemented using ActiveMQ[5]) exposing events during runtime. This can be easily extended to other elasticity controllers, by implementing an adapter for receiving and processing events. The service queue is using CloudAMQP[6], which is managed RabbitMQ[7] offered as a cloud service. Our Primefaces[8]-based frontend includes dynamically generating diagrams and charts for the cloud service provider employees.

5.2 Elasticity Operations Management Features

To illustrate eOMP features, we used our pilot application[9] which consists of: (i) an event processing topology composed of an event processing unit and a load balancer, and (2) a data end topology composed of a data node unit and a data controller unit. We used recordings from a previous run, to which we injected events (i.e., by modifying monitored data for a limited amount of time). We chose this approach instead of real-time injecting faults, since it is more reliable, and our focus is showcasing the eOMP platform, and not the service versatility.

Implicit vs. explicit interactions For understanding current service behavior, roles need elasticity controller interactions executed regularly (e.g., every 10 minutes, or each time the role logs into the platform). We distinguish between two types of interactions from the user/employee perspective: (1) implicit interactions, for getting the necessary data to be displayed to the employee (e.g., dialogs for getting the service description), and (2) explicit interactions, initiated by the eOMP user. Fig. 6 shows a dialog from the first type, with the system administrator (one role of the current logged in employee) requesting for initial description information. While without eOMP, the employee would need to manually call them, with eOMP the implicit interactions are already managed when the employee logs on in the platform.

Solving conflicting requirements The controller can encounter a case where no actions are suitable for solving the discovered issues. Fig. 7a shows a situation where constraints `Co1`("Co1:CONSTRAINT cost<10$;") and `Co3`("Co3:CON STRAINT responseTime<400 ms;") are conflicting, because of the high workload and the limit on the cost. Since the employee receiving this interaction has more roles associated, s/he decides to replace the requirement from the `Procurement`

[4] http://tuwiendsg.github.io/rSYBL/eOMP

[5] http://activemq.apache.org

[6] http://cloudamqp.com

[7] http://rabbitmq.com

[8] http://www.primefaces.org/

[9] https://github.com/tuwiendsg/DaaSM2M

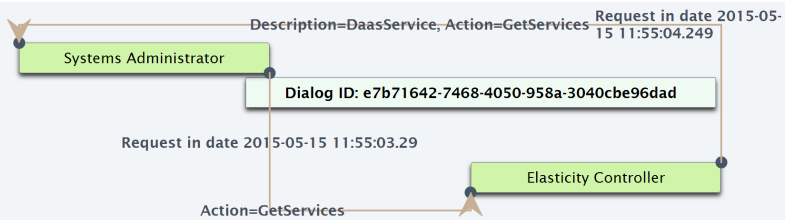
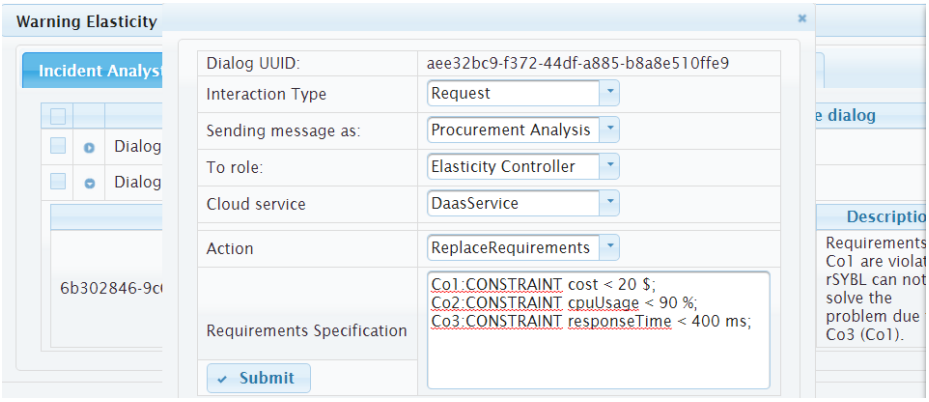
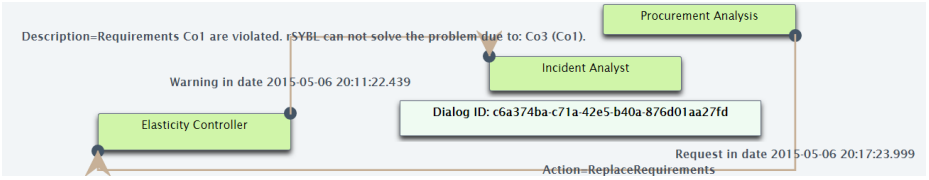


Fig. 6: eOMP snapshot: implicit initial dialog requesting services information



(a) eOMP snapshot: replace requirements



(b) eOMP snapshot: dialog for clarifying requirements

Fig. 7: Conflicting requirements resolution

`Analyst` role, for being able to increase the limit for the service cost. The interaction dialog from the three roles is shown in Fig. 7b, and consists of two steps: (1) the controller notifies the `Incident Analyst` role that no action is available due to the requirements conflict, (2) the employee uses the `Procurement Analyst` role to modify the cost requirement. Moreover, eOMP uses knowledge on role types and their authorities, avoiding modification conflicts (e.g., with no eOMP, two roles can fix observed issues at the same time). In our case, the role interaction with the elasticity controller is managed by eOMP, the employee being able to choose even a different role from which the issue can be solved better.

Service health incidents Another issue which may occur during service operation is that a service part might be unhealthy (i.e., monitoring metrics have error-like values for an amount of time). With eOMP, all the roles which have incidents as responsibilities receive notifications, but the timing and the amount of notifications is inversely proportional with their authority. Fig. 8 shows interactions which are due to `Event Processing Topology` being unhealthy. When the Operations Manager (high authority) gets the interaction from the controller, it means that the lower level authority roles have ignored or weren't

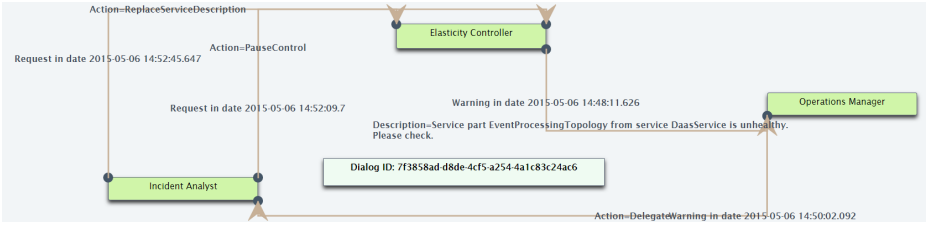
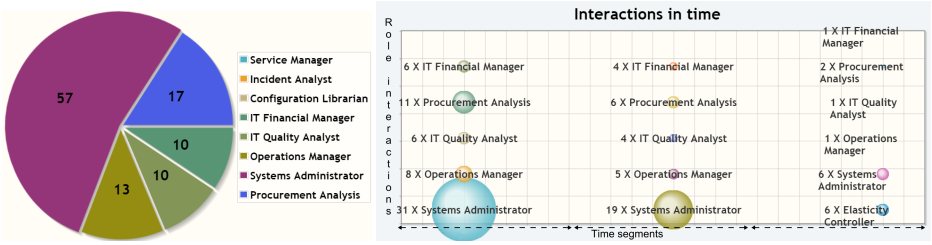


Fig. 8: eOMP snapshot: unhealthy service part dialog



(a) Total role interactions (b) Role interactions in time

Fig. 9: eOMP snapshots: Statistical information regarding interactions

able to address the situation. Therefore, it delegates the interaction to the Incident Analyst. The Incident Analyst can try to fix the issue, or can report on the difficulty of the issue. Left side of Fig. 8 shows the actions performed by the Incident Analyst (i.e., pause-fix-replace service description). All can be followed by the Operations Manager, to make sure the incident is being solved, since these interactions are part of the initial dialog.

Dealing with roles authorities Fig. 9 shows the number of interactions which occured over time, undertaken by each role. The amount of interactions varies with the events which occur, with role's responsibilities in relation with these events, and with role's authority. We can see that higher level authority roles have less interactions (e.g., Service Manager has no interactions), following the interaction aggregation function in Section 4.4.

Thus, eOMP facilitates interactions between service providers and elasticity controllers, and among service provider roles, automating operation tasks and providing support for interaction management based on role's authority and responsibilities. With eOMP the roles can easily follow the evolution of their elastic service, and the evolution, in time, of incidents, requests for change, or measures taken by the elasticity controller in order to control their service behavior.

6 Related Work

Various standards and processes have been proposed over the years for IT service management. Sallé [9] provides an analysis of the evolution of IT service management over the years, and its evolution towards IT governance. Starting from Information Systems Management Architecture [10], IT management methodologies have evolved towards well-defined standards/best practices of IT service management (e.g., ITIL® [11], BSI ISO 20000 [12], FitSM [13]). Although the focus of this paper is not the management processes adopted by organizations, understanding their internal processes is necessary for being able to support them

in their quest for cloud service elasticity. Since the latest reference models used nowadays in organizations (e.g., [12], [14]) are in alignment with ITIL® service management practices, we used these processes and organization roles. However, our design (see Section 4) is such that, roles and processes can be modified for accommodating future service management models.

Operation management in cloud computing has been approached mostly from the cloud provider's perspective [15, 16], and little from service provider (i.e., cloud customer) perspective. Bleizeffer et al. [7] propose a set of user roles in cloud systems, having as core roles not only the cloud service provider, but also the cloud service consumer and cloud service creator. The three core roles are expanded into a taxonomy of interconnected user roles, which communicate with each other for delegating responsibilities or gathering information. Demont et al. [17] present an initial proposal of integrating the TOSCA cloud service description standard with ITIL elements. Several commercial solutions enable cloud infrastructure management support, but do not support service operations management at cloud customer's service level (e.g., Oracle Enterprise Manager[10], BMC Cloud Operations Management[11]). Liu et al. [18] propose an incident diagnosis approach based on incident relationships, using co-occurring and re-occurring incidents for performing root cause analysis. Munteanu et al. [19] propose an architectural approach for cloud incident management, including incident lifecycle management, event and incident detection, incident classification and recovery and root cause analysis.

In contrast with above presented work, we focus on the elasticity aspect of service operations management in the cloud, characterizing the relevant properties and interactions. Moreover, we emphasize the importance of supervisory control for the cloud, and introduce service provider employees as first-class entities in the control loops.

7 Conclusions and Future Work

Operation management for services elasticity becomes increasingly more complex when the services rely on several other third-party services deployed in multiple cloud environments. We have proposed a set of interaction protocols for managing elasticity operations of cloud services, which take into consideration service provider roles as first class entities in the service elasticity control. We introduced the eOMP platform, which allows service provider employees to manage the cloud service operations related with elasticity, interacting with the elasticity controller and other employees of the service provider.

As future work, we are studying multi-organizational interactions for cloud services. For this case, dialogs are much more complex, since we need to model the various types of information necessary in the communication, and we need mechanisms to support and track inter-organization interactions. Moreover, organizations could follow different standards, and offer different access points (e.g., a cloud provider might expose IT Financial Administrator role for renegotiating contracts, and Operations Management for handling QoS issues).

[10] http://www.oracle.com/technetwork/oem/enterprise-manager

[11] http://www.bmc.com/it-solutions/cloud-operations-management.html

References

1. Mao, M., Humphrey, M.: Scaling and scheduling to maximize application performance within budget constraints in cloud workflows. In: IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS). (2013) 67–78
2. Jiang, J., Lu, J., Zhang, G., Long, G.: Optimal cloud resource auto-scaling for web applications. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. (2013) 58–65
3. Copil, G., Moldovan, D., Truong, H.L., Dustdar, S.: Sybl: An extensible language for controlling elasticity in cloud applications. In: International Symposium on Cluster, Cloud and Grid Computing (CCGrid), IEEE/ACM (2013) 112–119
4. Copil, G., Moldovan, D., Truong, H.L., Dustdar, S.: Multi-level Elasticity Control of Cloud Services. In Basu, S., Pautasso, C., Zhang, L., Fu, X., eds.: Service-Oriented Computing. Lecture Notes in Computer Science. Springer Heidelberg
5. Dustdar, S., Guo, Y., Satzger, B., Truong, H.L.: Principles of elastic processes. IEEE Internet Computing **15**(5) (2011) 66–71
6. Endsley, M.R., Kaber, D.B.: Level of automation effects on performance, situation awareness and workload in a dynamic control task. Ergonomics **42** (1999) 462–492
7. Bleizeffer, T., Calcaterra, J., Nair, D., Rendahl, R., Schmidt-Wesche, B., Sohn, P.: Description and application of core cloud user roles. In: Proceedings of the 5th ACM Symposium on Computer Human Interaction for Management of Information Technology. CHIMIT '11, New York, NY, USA, ACM (2011) 2:1–2:9
8. Sheridan, T.B.: Adaptive automation, level of automation, allocation authority, supervisory control, and adaptive control: Distinctions and modes of adaptation. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on **41**(4) (2011) 662–667
9. Sallé, M.: It service management and it governance: review, comparative analysis and their impact on utility computing. Hewlett-Packard Company (2004) 8–17
10. Van Schaik, E.A.: A Management System for the Information Business: Organizational Analysis. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1985)
11. Arraj, V.: Itil®: the basics. Buckinghampshire, UK (2010)
12. BSI Group: ISO/IEC 20000-Information technology-Service management. http://www.iso.org/iso/publication_item.htm?pid=PUB200013
13. FedSM: FitSM: Standards for IT Service Management. http://www.fedsm.eu
14. HP: The HP IT Service Management (ITSM) Reference Model. ftp://ftp.hp.com/pub/services/itsm/info/itsm_rmwp.pdf
15. Zhan, H., Zhang, W.: The Operation and Maintenance Management System of the Cloud Computing Data Center Based on ITIL. In: Advances in Computer Science and its Applications. Volume 279. Springer (2014) 1103–1108
16. IBM: Integrated service management and cloud computing:More than just technology best friends. https://www.ibm.com/ibm/files/E955200R99025N70/5Integrated_service_management_and_cloud_computing_644KB.pdf
17. Demont, C., Breitenbücher, U., Kopp, O., Leymann, F., Wettinger, J.: Towards integrating tosca and itil. In: ZEUS, Citeseer (2013) 28–31
18. Liu, R., Lee, J.: It incident management by analyzing incident relations. In Liu, C., Ludwig, H., Toumani, F., Yu, Q., eds.: Service-Oriented Computing. Volume 7636 of Lecture Notes in Computer Science. Springer (2012) 631–638
19. Munteanu, V., Edmonds, A., Bohnert, T., Fortis, T.F.: Cloud Incident Management, Challenges, Research Directions, and Architectural Approach. In: International Conference on Utility and Cloud Computing, IEEE/ACM (2014) 786–791