Active Mechanisms for Cloud Environments

Irina Astrova
Institute of Cybernetics
Tallinn University of Technology
Tallinn, Estonia
irina@cs.ioc.ee

Stella Gatziu Grivas, Marc Schaaf
Institute for Information Systems
University of Applied Sciences Northwestern Switzerland
Olten, Switzerland
{stella.gatziugrivas, marc.schaaf}@fhnw.ch

Arne Koschel
Faculty IV, Department for Computer Science
University of Applied Sciences and Arts Hannover
Hannover, Germany
arne.koschel@fh-hannover.de

Ilja Hellwich, Sven Kasten, Nedim Vaizovic, Christoph Wiens
Faculty IV, Department for Computer Science
University of Applied Sciences and Arts Hannover
Hannover, Germany
arne.koschel@fh-hannover.de

***Abstract*—Active mechanisms are used for the coordination (e.g., scalability) of IT resources in clouds. In this paper, we give an overview of existing technologies and products – viz., OM4SPACE Activity Service, RESERVOIR, Amazon SNS, IBM Tivoli Live Monitoring Service, Zimory and PESA – that can be used for providing active mechanisms in cloud environments. Our overview showed that these technologies and products mainly differ in the architectures they support and the cloud layers they provide.**

***Keywords*—Cloud computing; events; active mechanisms.**

I. Introduction

Cloud computing has become more and more popular. Many companies (viz., cloud providers) are outsourcing their IT resources into clouds so that users can hire those resources only if they really need the resources and give the resources back when they do not need the resources any longer. This creates a new challenge for cloud providers – they need to provide users with systems, which can automatically assign IT resources on the fly. These systems should give the possibility to evaluate events from different event sources at one or more external coordination points. These points can coordinate the usage of the IT resources in clouds. Thus, the systems should use active mechanisms for the coordination (e.g., scalability) of IT resources in cloud environments.

The purpose of this paper is to give an overview of existing technologies and products that can be used for providing active mechanisms in cloud environments. Technologies like OM4SPACE Activity Service, RESERVOIR and PESA are mostly theoretical concepts and not end products. There are also (commercial) end products like Amazon SNS, IBM Tivoli Live Monitoring Service and Zimory.

II. OM4SPACE Activity Service

OM4SPACE [2] provides software-as-a-service (SaaS), platform-as-a-service (PaaS) and infrastructure-as-a-service (IaaS). The crucial part of OM4SPACE is the Activity Service.

In cloud environments, often a large number of services occur on different layers. The Activity Service offers an approach for managing a large number of events from different event sources, processing these events and triggering appropriate actions on the events, e.g., starting new virtual machine instances when a specified threshold for the CPU load has been exceeded.

Figure 1 shows the architecture of the Activity Service, which consists of the following components:

- **Event Source:** This component can be an arbitrary part of a cloud environment; it generates different types of events (both simple and complex). Every event is sent to the Event Service for further processing. The Event Source can be on any layer of a cloud environment: SaaS, PaaS and IaaS.
- **Event Service:** This component receives events from an arbitrary number of Event Sources and performs the first step of processing, which is divided into two phases. The first phase dispatches the received events to Event Consumers that are registered for this type of events. The second phase consists of performing complex event detection (CED) on the incoming event stream. This CED can cause new complex and enriched events to be created by the Event Service and dispatched to the registered Event Consumers. Thus, the Event Service controls a granularity shift of the incoming event stream and helps to scale down the number of events, which enables complex event and rule processing by the Rule Execution Service.
- **Event Consumer:** This component receives a particular event type from the Event Service. Events can be of two types: simple events that the Event Service receives and complex events that the Event Service detects and generates. To receive events from the Event Service, an Event Consumer has to implement an appropriate event handler service, which needs to be published to the service registry.

The Event Service discovers event handler services by looking for the service registry. To inform the Event Service about the events an event handler service is interested in, filtering criteria have to be added to the WSDL description, which will be extracted by the Event Service.

- **Rule Execution Service:** This component receives events from the Event Service to match them against rules. Thus, it acts as an Event Consumer of the Event Service by registering an event handler service. Matching of the rules results in the execution of action handlers. An action handler needs to be implemented by each of the components that are to be called from within the rules. The rules are stored in the Rule Base, which is managed by the Rule Management Service.
- **Event Monitor (EM):** Not all components are built for active notification by the Event Service. For those components, a monitor capsule mechanism is used. As a result, a small application that acts as a monitor capsule around the Event Source can be implemented in such a way that it obtains events from the Event Source and transfers them to the Event Service. Furthermore, the monitor capsule can provide conversion between different types of events.

OM4SPACE adapts an activity service embedded in active database management systems (ADBMSs) to cloud environments. Active mechanisms are divided into different components that are put into the cloud. Each of these components has one or more well-defined interfaces. So the implementation of the components is interchangeable. The communication between the components is implemented using a service-oriented architecture (SOA). In addition to processing a large number of events, the Activity Service can be used for monitoring and scaling applications in the cloud.

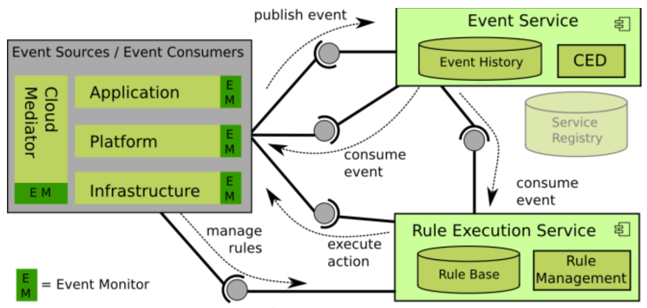

Figure 1. Architecture of OM4SPACE Activity Service [2].

III. RESERVOIR

RESERVOIR [3][4] provides IaaS. It requires the usage of manifests. A manifest serves as a contract between service and infrastructure providers.

Figure 2 shows the architecture of RESERVOIR, which consists of the following components:

- **Hypervisor:** This is a layer of abstraction, which runs on top of physical hardware. It allocates (physical) resources to virtual machines and manages and controls the execution of them by booting, suspending and shutting down resources as required. It can even provide the replication and migration of virtual machines. Examples of a Hypervisor include Xen [5] and VMWare [6].
- **Virtual Execution Environment Host (VEE Host):** This is the lowest layer in the architecture and provides plug-ins for different hypervisors. It enables upper layers to interact with heterogeneous virtualized products.
- **Virtual Execution Environment Manager (VEE Manager):** This layer implements the key abstractions needed for cloud computing and provides the functionality to control multiple VEE Hosts. Because of cross-site interactions between multiple different VEE Managers, the architecture offers the possibility to deal with and federate different sites, which implement heterogeneous virtualized products. Examples of a VEE Manager include OpenNebula [7].
- **Service Manager:** This layer is an interface to build the connection to the Service Providers. It ensures that the requirements of them are correctly enforced.
- **Service Provider:** This is the highest layer in the architecture and offers services to provide operations of specified businesses and uses the Service Managers to connect to the cloud.

RESERVOIR brings active mechanisms and the usage of events into cloud environments. The scalability of a service is enabled through an application description language (which introduces a monitoring framework along with Monitoring Agents) and key performance indicators (which describe the state of the service). The Monitoring Agents send Monitoring Events to the service management infrastructure, where these events are processed and rules are executed. The execution of these rules is additionally monitored by OCL operations to insure the correctness.

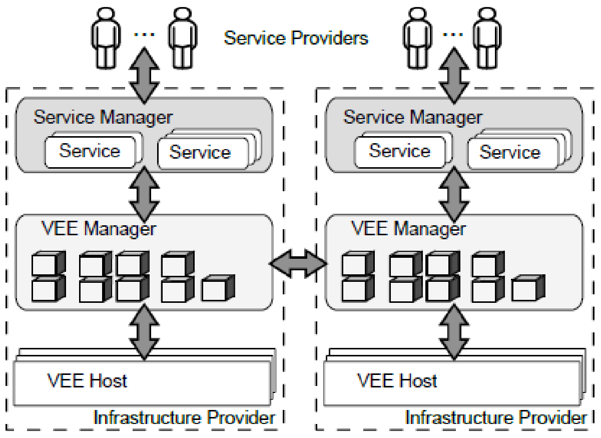

Figure 2. Architecture of RESERVOIR [3].

IV. AMAZON SNS

Amazon SNS (Simple Notification Service) [9] is a middleware product that offers a service for managing and sending notifications over cloud environments. The crucial part of Amazon SNS is topics.

Before sending notifications a topic has to be created. This is done by providing the topic name, which is used by Amazon SNS to generate a unique identifier called Amazon Resource Name. After the topic has been created, notifications can be sent to it. A notification consists of a message, the Amazon Resource Name and optionally a subject. Amazon SNS also adds meta-data like a signature and a timestamp to the notification.

To receive notifications, a subscriber needs to be registered for a topic. Every notification that arrives at a topic is delivered to all subscribers of this topic. Subscription contains endpoint information, which defines how the notification is delivered to the subscriber. Amazon provides the following types of endpoints:

- **Email:** The notification is sent via an email by using the SMTP protocol. The notification subject is mapped to the email subject and the notification message is mapped to the email message. Amazon SNS adds additional information to every outgoing notification, which contains an HTTP link for unsubscribing.
- **Email JSON:** The notification is also sent via an email but by using the human and machine-readable format called JSON [10]. All notification properties (e.g., the timestamp, the message and the subject) are stored in a list of key-value pairs.
- **HTTPS:** The notification is delivered by using the HTTPS protocol. In case of the incoming notification, Amazon SNS performs an HTTP-Post on a specified URL. The body of the HTTP-Post contains all notification information in the JSON format. All notification properties are encrypted and stored in a list of key-value pairs [11].
- **HTTP:** Like HTTPS but without using encryption.
- **Amazon SQS:** The notification is delivered to a queue of the Amazon SQS (Simple Queue Service). This is another Amazon service, which provides the functionality of sending text messages to the cloud. These messages are cached for a limited period of time while clients can request and receive them.

Every subscription needs to be confirmed by the receiver. For this purpose, Amazon SNS sends a confirmation request, which contains a specific token, to another endpoint. By transmitting this token back to Amazon SNS, the subscriber guarantees that it gets access to the endpoint and can receive notifications. Otherwise, it would be possible to enter arbitrary endpoints and flood them with notifications, which they do not want to receive. Amazon SNS supports an event-driven architecture (EDA) to decouple the notification sender from the receiver (i.e., the subscriber). Thereby it propagates this decoupling into the cloud.

Amazon allows for a detailed access control on Amazon SNS and its topics, e.g., by defining who is allowed to access a topic, who is allowed to add subscriptions to this topic and what types of subscriptions are permitted. For this purpose, Amazon introduces syntax for defining policies. These policies contain all information, which is important for security and access-control configuration.

Amazon SNS provides easy-to-use active mechanisms. But these mechanisms do not enable performing CED or condition-based rule execution. Another big problem with Amazon SNS is the small size of a message (8 kilobytes per notification).

V. Zimory

Zimory [13] provides IaaS. It is a product, where highly distributed components cooperate with each other using active mechanisms.

Monitoring and management are used to assure the scalability of virtual machine instances. The monitoring and management are done via a web interface through which users can specify rules that trigger one or more of the following actions when an event occurs [8]:

- **Storeback:** During this action, the virtual machine will be restored from a specified backup or set to a specified state.
- **Snapshot:** During this action, a snapshot of the current running state of the virtual machine will be created.
- **Clone:** During this action, the current running virtual machine will be duplicated (i.e., cloned). A clone can then be started immediately.

Zimory uses active mechanisms to distribute the CPU load to more than one virtual machine instance. In such a scenario, the cloud component (not named in the Zimory documentation), which monitors the instances, is acting like an event producing service that evaluates on what state the event should be produced. After this, the event is published to a component inside the cloud, which is able to receive events regarding the instances, a kind of event receiving service. This service then processes events and performs one or more of the actions listed above.

VI. IBM Tivoli Live Monitoring Service

IBM Tivoli Live provides SaaS. The crucial part of IBM Tivoli Live is the Monitoring Service [12].

The Monitoring Service is used to monitor network components and manage them in an online web portal. For this, a monitoring server is installed in the cloud. This server can monitor the cloud components (both active and passive) and send the collected data to the cloud. Here the data are stored and can be accessed via an online web portal. This portal gives users the possibility to set thresholds for the different monitored parameters of a component. When a threshold exceeds, an event is generated and an action for the event is performed. Unfortunately, the IBM documentation does not further specify how such an event or action can be used. But in theory it should be possible to send alarms via email to the users to notify them. Also, it should be possible to automatically perform an action on components, which use the Monitoring Service with an active agent. Such an action could be the execution of a script to automatically repair the state of a failed cloud component.

Figure 3 shows the architecture of the Monitoring Service. The following types of monitoring occur in the Monitoring Services:

- **Touchless monitoring:** A component, which is monitored, needs no further software installed, except a simple standard SNMP service. The Monitoring Service uses SNMP to retrieve current information about the component. This is only passive monitoring without any chance of interaction or controlling of the component. Also, the monitored data can be different across several components because SNMP does not standardize the monitored data but the messaging protocol.
- **Distributed monitoring:** This type of monitoring is based on an extra agent, which is deployed to the component being monitored. The monitoring server connects to this agent to retrieve information on the component. With the agent solution, it is also possible to control and manage the component.
- **Performance services:** These services are used to monitor data for long-time performance analysis and bottleneck indication. It should be noted that the performance services are used for manual analysis only.

The Monitoring Service uses multi-agent systems for active mechanisms. So it is more aligned to the "old-fashioned" IT systems. But the Monitoring Service can also be used in cloud environments because such environments are nothing else than virtualized IT systems.

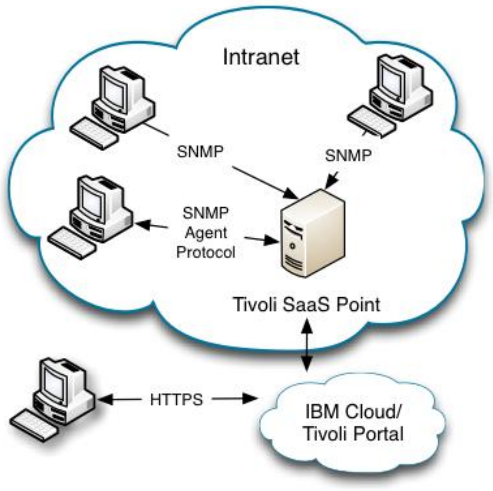

Figure 3. Architecture of the IBM Tivoli Live Monitoring Service [12].

VII. PESA

PESA [11] is a technology that adds policies to an event-driven service-oriented architecture (ESA). A policy is a set of rules that control behaviors of services.

A large number of services that are deployed into different cloud environments can generate a large number of events. Because of that, it is not practical to set up a centralized management service that collects, stores, correlates and processes events. A better approach would be to build management services responsible for a small number of services, thereby reducing bottlenecks and speeding up the reaction of the whole system. PESA offers this approach. The management services are used for:

- Matching a pattern that consists of different single events, which can indicate some failure.
- Creating high-level business events out of simple events, which can indicate that some reaction of the business layer is needed to assure the execution of the business workflow.
- Adding data to events to give more information about the reason for the occurrence of an event.
- Triggering conditions for policies or rules to react on occurrence of events without human interaction.
- Invoking services or business workflows when their conditions are satisfied to assure that a business workflow can be executed.

As the services they are managing, the management services should be loosely coupled and highly distributed so that they can be set up in different cloud environments near the managed services. This will improve the performance while managing the services, reduces the complexity and eliminates drawbacks of a centralized management service (e.g., single points of failure). In such a scenario, the managed services can automatically incorporate service components for monitoring and management from their "local" management services. Furthermore, it becomes possible to compose the "small" management services into a "bigger" management service that provides coherent management for all services used in the business workflow.

Figure 4 shows the architecture of the management service. The components of the architecture can be assigned to one of the following layers:

- **Monitors and sensors, extraction and transformation tools:** Events are generated through monitoring services and sensors. Then every occurring event is transformed to a standard data format by a tool and the resulting events are published to an enterprise service bus (ESB). Both layers correspond to the event generation layer of an EDA.
- **Classification and categorization:** This is the layer where events are classified and assigned to a class of events in order to provide a coherent scope on the events for the layer above.
- **Analysis engines:** This layer correlates, associates and links events together to generate more complex events that have relevance to business. It corresponds to the event processing layer of an EDA.
- **Operational actions, policy actions and conflict resolution actions:** These are the layers where policies trigger actions based on events as their conditions. So the layers are responsible for performing the actions that assure the execution of a business workflow. The layers correspond to the event handling layer of an EDA.

All the layers communicate with each other through the ESB. Layers are built so that higher layers have broader scope that enables more complex analysis and management.

But it does not mean that the high and low-level management services are built with more or less complex components. Rather every management service is able to handle events and perform actions based on policies. Such a system is highly extensible, by adding more management services responsible for managing more services.

The intelligence that PESA adds to active mechanisms is the possibility to build business workflows that are very agile and can be easily adapted to changes, without human interaction during the execution of a business workflow. This becomes possible only by adding the events to signal changes and by adding the policies to react on those changes.

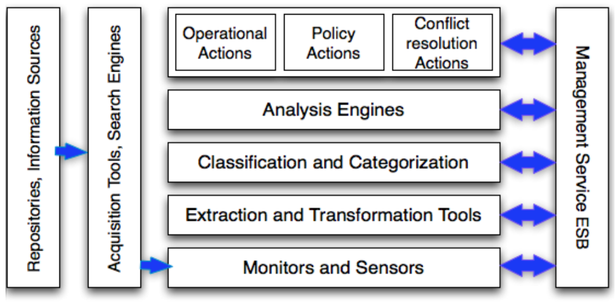
Repositories, Information Sources
Acquisition Tools, Search Engines
Operational Actions
Policy Actions
Conflict resolution Actions
Analysis Engines
Classification and Categorization
Extraction and Transformation Tools
Monitors and Sensors
Management Service ESB

Figure 4. Architecture of the PESA management service [11].

VIII. Conclusion

This paper gave an overview of existing technologies and products – viz., OM4SPACE Activity Service, RESERVOIR, Amazon SNS, IBM Tivoli Live Monitoring Service, Zimory and PESA – that can be used for providing active mechanisms in cloud environments. Table I summarizes this overview.

TABLE I. Summary of overview of technologies and products for providing active mechanisms in cloud environments

Technology / Product	Architecture	Cloud layer
OM4SPACE Activity Service	SOA	SaaS, PaaS, IaaS
RESERVOIR	SOA	IaaS
Amazon SNS	EDA	PaaS
IBM Tivoli Live Monitoring Service	SOA	SaaS
Zimory	EDA	IaaS
PESA	ESA	PaaS

The overviewed technologies and products mainly differ in the architectures they support: EDA, SOA or ESA.

An EDA [1] enables event producers to publish their events and event consumers to subscribe to and consume those events. In an EDA, events know nothing about their consumers. Events can also remain unconsumed because none of the consumers is interested in them. There are no direct relationships between the event producers and consumers. So the services built on top of an EDA are loosely coupled. This helps an EDA fit into a scenario where the services deployed into the cloud are managed by management services. Support of an EDA can be found in Amazon SNS and Zimory.

A SOA enables the composition of loosely coupled highly distributed services. These services can be deployed into different cloud environments where the clouds themselves take care of the services. Support of a SOA can be found in OM4SPACE Activity Service, RESERVOIR and IBM Tivoli Live Monitoring Service. The Activity Service in its initial version follows a cloud-native approach, by using an ESB to build a SOA and Web services to provide communication between loosely coupled highly distributed components. RESERVOIR also supports a SOA but it does not actually specify such communication. The Monitoring Service follows a more old-fashioned approach, by using multi-agent systems for active mechanisms. One possible reason for this is the growing structure of the whole IBM Tivoli Live.

An ESA [11] is the result of combining an EDA with a SOA. Such a combination is needed because a SOA typically composes services to business workflows. It does not account for events that occur across or outside of business workflows or complex events. Being combined with an EDA, a SOA can react on events. For example, a high-level business event can cause the execution of a single service or a set of services that can handle the problem occurred in a business workflow. Such a SOA enriched by events through an EDA can be used to build agile business workflows that adapt to changes, which occur during the execution of the business workflow. In such a scenario, the changes will be signaled by events. An EDA can also take advantage of the combination with a SOA because of the flexibility that a SOA provides through composing of services on different layers. As a result, it becomes possible to integrate an EDA on every layer. So an EDA can become responsible for publishing, subscribing and consuming events on both simple low service layer and high complex business layer. Because of these advantages, an ESA fits well into a scenario where events should be monitored, enriched and connected with each other on different levels. The connection between events is important because it can be used to connect multiple low-level system events to create a high-level business event. In such a scenario, events can occur everywhere, e.g., they can be created from applications, databases or services that are involved in a business workflow. Support of an ESA can be found in PESA. One possible reason for this is that cloud environments are typically environments for loosely coupled highly distributed services that can be orchestrated to a business workflow.

The overviewed technologies and products also differ in the cloud layers on which they provide their services: SaaS, PaaS or IaaS.

SaaS is a model of software deployment whereby a cloud provider licenses an application (i.e., software) to users for the usage as a service on demand. OM4SPACE Activity Service and IBM Tivoli Live Monitoring Service provide SaaS. One possible reason for this is the popularity of SaaS. (Currently, SaaS is the most popular type of cloud computing because of its simplicity, flexibility and scalability.)

PaaS is a model of application development and delivery. In particular, PaaS offers a development platform for users. OM4SPACE Activity Service, Amazon SNS and PESA provide PaaS.

Whereas SaaS allows for the usage of applications in cloud environments and PaaS offers the ability to develop and deliver these applications, IaaS provides users with the infrastructure for developing, running and storing the applications. RESERVOIR and Zimory provide IaaS. OM4SPACE Activity Service could also be used as an event processing component within IaaS.

IX. Future Work

There are advantages and disadvantages with all the overviewed technologies and products. An advantage of one is often a disadvantage of another. Therefore, the most promising approach would be to combine them all. This approach could be based on OM4SPACE Activity Service because it could possibly operate or be utilized on all different layers of a cloud environment and performs the complete event roundtrip by: generating events at an Event Source, sending events to the Event Service, performing CED at the Event Service and generating new complex events, sending events to an Event Consumer and the Rule Execution Service, performing rule processing and rule execution, and performing action invocations on action handlers in case of matching rules.

The Activity Service could benefit from using RESERVOIR. RESERVOIR defines a standard way to monitor a cloud component in order to read the parameters out of that component using a manifest, which assures that the Service Providers can deploy their services into the cloud. The Activity Service should also be able to monitor cloud components (both active and passive), but without concretely defining what parameters would be monitored. This may first not be seen as a real problem. But when rules for the events should be generated, it may become a big issue because the rules use the attributes of the events, which are set at a cloud component. Also, for the content enrichment of events at the Event Service, a concrete set of attributes should be defined. Thus, the usage of RESERVOIR could solve the problem of defining rules and getting the parameter dependencies for the rules.

Another improvement could be done in the action performing part of the Activity Service, which is currently implemented as a simple call to an action handler. This could be improved if the action handler and the Rule Execution Service used Amazon SNS as a transport mechanism. For example, the action handler could subscribe to a topic filled by the Rule Execution Service via Amazon SNS. But this problem cannot be solved by using Amazon SNS solely because this product is mostly not standardized and thus, can cause a vendor lock-in. Another problem with the usage of Amazon SNS is the small message size (8 kilobytes only). Therefore, a better solution would be if the Activity Service itself could implement a-la Amazon SNS transport mechanism. Such independence from a transport mechanism is currently implemented in the Activity Service to allow for better integration with existing cloud communication services.

Acknowledgment

Irina Astrova’s work was supported by the Estonian Centre of Excellence in Computer Science (EXCS) funded mainly by the European Regional Development Fund (ERDF).

References

[1] J. Dunkel and R. Bruns. Event-Driven Architecture. Springer, 2010.

[2] A. Koschel, M. Schaaf, S. Gatziu Grivas, and I. Astrova. An Active DBMS Style Activity Service for Cloud Environments. 1st Intl. Conf. Cloud Computing 2010, pages 80–85, IARIA, Portugal, November 2010.

[3] C. Chapman,W. Emmerich, F. G.M´arquez, S. Clayman, and A. Galis. Software architecture definition for on-demand cloud provisioning. 19th ACM International Symposium on High Performance Distributed Computing, HPDC’10, pages 61–72, New York, NY, USA, 2010. ACM.

[4] S. Eliot. Reservoir homepage. http://www.reservoir-fp7.eu/ Accessed: November 2011.

[5] Citrix Systems Inc. Xen. http://www.xen.org/ Accessed: November 2011.

[6] VMware Inc. Vmware. http://www.vmware.com/ Accessed: November 2011.

[7] OpenNebula. Opennebula. http://www.opennebula.org/ Accessed: November 2011.

[8] F. Galan, A. Sampaio, L. Rodero-Merino, I. Loy, V. Gil, and L. M. Vaquero. Service specification in cloud environments based on extensions to open standards. 4th Intl. ICST Conference on Communication System software and middleware, COMSWARE’09, pages 19:1–19:12, New York, NY, USA, 2009. ACM.

[9] Amazon Simple Notification Service (Amazon SNS). http://aws.amazon.com/de/sns/ Accessed: November 2011.

[10] Json (javascript object notation). http://www.json.org/ Accessed: November 2011.

[11] P. Goyal and R. Mikkilineni. Policy-based event-driven services-oriented architecture for cloud services operation and management. IEEE Intl. Conference on Cloud Computing, pages 135–138, 2009. IEEE.

[12] IBM Tivoli foundations and IBM Tivoli Live Monitoring Services. http://www-01.ibm.com/software/tivoli/products/monitor/ Accessed: November 2011.

[13] Z. GmbH. Zimory Enterprise Cloud Anwendungsbeispiel. http://www.zimory.de/index.php?id=75 Accessed: November 2011.