An Event Driven Multi-Agent Architecture for Enabling Cloud Governance

 Victor Ion Munteanu*,†, Teodor-Florin Fortiș*† and Viorel Negru*†

*Institute e-Austria, Timișoara, Romania
†Faculty of Mathematics and Informatics
Department of Computer Science
West University of Timișoara, Timișoara, Romania
e-mail: {vmunteanu,fortis,vnegru}@info.uvt.ro

Abstract—Cloud adoption is consistent within IT-based industries at different maturity levels. While cloud migration is an ongoing process, its base characteristics are not yet fully exploited.

As Cloud Governance is built on top of requirements like security, reliability, trust, portability, interoperability, or failover, a highly distributed and concurrent, and fault tolerant solution is required in order to achieve above specified requirements.

This paper describes the core of an event-driven multi-agent architecture for supporting Cloud Governance activities, built around Akka/Clojure actor characteristics and fully exploiting the Enterprise Integration Patterns. The modeling process follows the Gaia methodology for agent-oriented analysis and design.

Keywords—cloud governance, cloud computing, Gaia methodology, multi-agent system, cloud management

I. INTRODUCTION

With the advent of Cloud Computing, more and more enterprises seek to take advantage from its pay-per-use business model as well as its core characteristics [1], [2]. Of these, small and medium-sized enterprises (SMEs) can benefit the most by eliminating the need for maintaining IT infrastructure and personnel.

By using Cloud Computing, “developers with innovative ideas for new Internet services no longer require the large capital outlays in hardware to deploy their service or the human expense to operate it” [3].

Unfortunately, as cloud adoption drives forward and more and more services become available due to the fact that “an increasing number of SMEs [...] are thinking of migrating some aspects of their operations to the cloud” [4], the demand for service integration remains unsatisfied [5], [6] and most of the services are being run in isolation.

Cloud migration is not easy to achieve as it relies on additional solutions being available like platform abstraction (PaaS) or cloud management which both enable and enhance ease of use in utilizing and managing of cloud resources.

Even if there are many platform-as-a-service (PaaS) solutions, most of them only cover part of the issues related to Cloud Computing and require an additional set of services that are complementary to the existing, sometimes built into PaaS, cloud management solutions [7], [8], [9].

Cloud governance comes as the next step in cloud development, after cloud management. Built on top of SOA governance, and around the ISO/IEC 38500 principles for IT governance, cloud governance provides the ability to set policies within the environment in order to ensure system wide security, privacy and compliance, as well as provide the business level and integration with decision support systems, which cloud monitoring and PaaS solutions clearly lack. When cloud governance is coupled with cloud management and PaaS solutions it can drive cloud adoption up.

Thus “creating simple marketplaces with common rules, that enable the dynamic selection and consumption of functionality, is the missing link to allow small businesses to enter the cloud [...] as vendors” [10]. Furthermore, by facilitating vendors to group and work together as virtual enterprises enables the development of complex tailored or targeted solutions, based on services available from the different members of a virtual enterprise.

The focus of this paper is to model the agents that make up the core of an event driven multi-agent architecture for cloud governance using the Gaia methodology.

The remainder of the paper is organized as follows. Section II covers related work in agent-based software engineering and cloud multi-agent systems. Section III offer basic information on cloud management and governance, while Section IV establish the principles for modeling a multi-agent cloud governance solution. Conclusions and future work are detailed in Section V.

II. RELATED WORK

A. Agent-based software engineering

The design of multi-agent architectures is a complex process which can be enhanced through the use of agent-based software engineering. These tools play and important role as they “assist in all the phases of the life cycle of an agent-based application, including its management” [11]. There are several approaches for agent-based software engi-
neering like agent-oriented methodologies, frameworks and conceptual modeling.

The MESSAGE methodology [12] is built on top of unified modeling language (UML) by integrating agent-related concepts in order to support agent-oriented software engineering.

Tropos offers one of the most important methodologies for developing agent-oriented software [13]. This methodology takes advantage of agent oriented programming but also follows the belief, desire, intention (BDI) agent architecture from the early stage of requirements to the later stage of implementation.

Gaia is a high level methodology for agent-oriented design and analysis that allows the description of small-scale, closed multi-agent systems both at micro and macro levels [14], [15]. It focuses on roles which come with responsibilities, permissions, activities, and protocols.

B. The mOSAIC Project and mOSAIC’s Cloud Agency

The Open source API and Platform for multiple Clouds (mOSAIC) is an FP7-ICT project, whose main goal is to provide a unified Cloud programming interface enabling the flexibility needed to build interoperable applications across different Cloud providers [16], and it is currently comprised of the mOSAIC PaaS (API) and the Cloud Agency (CA).

While the mOSAIC API is a Cloud-based language and platform-independent API which extends the existing language or platform-dependent API capabilities with composite features based on patterns [17], the Cloud Agency is a multi-agent system that has been designed to handle resource provisioning, monitoring, and reconfiguration of resources [18].

C. Cloud multi-agent systems

There are several approaches that aim to provide multi-agent solutions for various cloud related issues.

D. Talia argues that “the convergence of interests between MAs that need reliable distributed infrastructures and cloud computing systems that need intelligent software with dynamic, flexible, and autonomous behavior will result in future intelligent services and applications”, covering the issues needed to be addressed in order for this to take place [19].

One particular approach focuses on the analysis of a multi-agent system security techniques in relation to cloud distributed storage [20]. The authors propose a two layer security framework, one layer consisting of multiple agents and another layer being at the cloud data storage level.

Another interesting approach aims to provide an agent based solution for automated cloud service composition [21]. Their solutions tackles problems like the dynamical contracting of service providers, the change in service fees and the incomplete information about cloud resources while using self-organizing agents.

### Table I

<table>
<thead>
<tr>
<th>Role</th>
<th>ServiceProviderProxy (SPP)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>Receives requests from service providers, authenticates them and, based on the request, ensures appropriate actions are taken.</td>
</tr>
<tr>
<td><strong>Protocols &amp; Activities:</strong></td>
<td>AwaitRequest, Authenticate, DelegateRequest, InformServiceProvider</td>
</tr>
<tr>
<td><strong>Permissions:</strong></td>
<td>reads supplied serviceProviderIdentity, supplied serviceProviderCredentials, supplied operationInformation, supplied operationOutput, supplied securityToken</td>
</tr>
<tr>
<td><strong>Responsibilities</strong></td>
<td>ServiceProviderProxy = (AwaitRequest, ProcessRequest) Ω ProcessRequest = (Authenticate, [DelegateOperation], InformServiceProvider)</td>
</tr>
<tr>
<td><strong>Liveness:</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>Safety:</strong></td>
<td>true</td>
</tr>
</tbody>
</table>

### III. Cloud Management and Governance

In Distributed Management Task Force (DMTF) white papers [8], [9], the different Cloud Management issues and concerns are identified in relation with the core aspects of the lifecycle of a cloud service. Various components in the architecture for managing clouds are identified, including monitoring/evaluation, key management, policy/security management, notification management.

However, DMTF’s documents are rather interested in prescriptions for management interfaces than in detailing cloud management concerns. Being built on the basis of a security-oriented architecture, the described architecture touch not only management requirements, but also cloud governance requirements, and these are identified in close relation one with the others.

While in [8] the place of cloud governance components is clearly specified in relation with the generic cloud management architecture, the white paper [7] offers detailed information related with Service Level Agreements (SLA), security patterns and security controls, as part of their vision for cloud governance.

1) **Cloud Governance Architecture:** A cloud governance architecture can be built in close relation with mOSAIC’s Cloud Agency and designed to offer a variety of services which complement it. This architecture will have to follow closely DMTF’s document [8] and can be built as a multi-agent system. mOSAIC’s Cloud Agency exposes itself within the ecosystem as services.

2) **Cloud Service Lifecycle:** Cloud governance’s ability to manage the lifecycle of services is fundamental for achieving success within the cloud environment. Service lifecycle in the cloud environment must successfully deal with the heterogeneity and dynamics of this environment. Unlike SOA service lifecycle, cloud service lifecycle has to
take into account the distribution of services across different cloud providers and the ability of services to scale.

IV. MODELING THE CLOUD GOVERNANCE ARCHITECTURE

In order to model the proposed architecture, we will be using the Gaia methodology [14], [15] for agent-oriented software engineering. This methodology enables us to have a high-level approach to analysis and design phases through which our architecture passes. We are not deterred by the fact that Gaia is not compliant with any international agent standard like FIPA, since the implementation is going to be done under Akka, which is a runtime built on top of the Scala programming language, for building distributed, highly-concurrent and fault tolerant event-driven applications.

The Gaia methodology is composed of two stages: analysis and design. The analysis stage tries to understand the structure and organization of the system. The design stage involves the transformation of existing high level models described during the analysis stage into low level abstractions that can be then used to implement the agents.

The remainder of the section will describe the cloud governance behavior we are trying to analyze and design while closely following the above described models, focusing our discussion only on the agents which play a major role in the service lifecycle and their most important responsibilities.

A. General description

While a complete cloud governance system can encompass a large number of processes, some related to security and privacy, others to service lifecycle, the particular one that is going to be modeled refers to the ability of the system to serve external requests coming from service providers, as well as the one that enables an adequate number of service instances to be running. These activities involve both governance and management functionality, and cover tasks ranging from providing security to providing brokering, deployment, and others.

The first process is initiated when a service provider contacts the system in order to execute a certain service related operation. Because of security related reasons, the provider does not have access to the whole system, instead all its requests must go through a proxy, which will act on its behalf in the system. This proxy will make sure that the provider is properly authenticated by contacting the security subsystem before proceeding with the execution of the desired operations.

All service related operations (publishing, updating, removing, searching) are handled by a service subsystem. It executes the operations on behalf of the service provider and returns the output.

The second process is initiated when the service system
detects that there are not enough service instances and more need to be created. The first step to create a new service instance is to make sure that all its dependencies are covered. In order to do that, a broker must search, select and contract them. Some of the dependencies can be other services, some can be cloud resources. In order to provision the cloud resources, a specific system must be able to communicate with cloud providers. Once all dependencies have been met, the service is deployed and commissioned and the service system is notified with that the instance is ready and running.

B. Roles model

Based on the description, several distinct roles have been identified. Of these, two act as interfaces to the service provider (ServiceProviderProxy, Table I) and to the cloud providers (CloudVendorProxy, Table VI). The service lifecycle is primarily centered around the ServiceManager (Table III) which makes use of ServiceProviderProxy (Table IV) and ServiceBroker (Table V) in order to complement its functionality. Last but not least, the SecurityManager (Table II) plays a central role by providing security throughout the system.

C. Interaction model

The focus will be on several interaction models associated with the roles of ServiceProviderProxy and ServiceManager.

The ServiceProviderProxy role interacts with the SecurityManager role in order to authenticate the provider and receive the securityToken (Authorize protocol, Figure 1(a)) and with the ServiceManager role in order to forward the operation for execution (DelegateOperation, Figure 1(b)).

The ServiceManager role interacts with the SecurityManager role in order to authorize the operation that must be executed (Authorize protocol, Figure 3(a)) and also with the ServiceDeployer when it needs to manage instances (ManageInstances protocol, Figure 3(b)).

D. Agents model

There is direct correspondence between existing roles and agent types as show in Table VII. As can be seen, some agents like ServiceAgent, SecurityManagementAgent and ServiceManagementAgent are needed at run-time while the DeploymentAgent, BrokerAgent and VendorAgent are only needed when a service needs instantiation. ServiceAgent needs to scale as more service providers connect. The SecurityManagementAgent and ServiceManagementAgent...
Table V
SCHEMA FOR SERVICEBROKER ROLE

<table>
<thead>
<tr>
<th>Role Schema:</th>
<th>ServiceBroker (SB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Handles the service brokering. Automatically filters, selects and contracts services.</td>
</tr>
<tr>
<td>Permissions:</td>
<td>reads supplied serviceProperties, supplied securityToken, generates serviceInformation</td>
</tr>
<tr>
<td>Liveness:</td>
<td>true</td>
</tr>
</tbody>
</table>

Table VI
SCHEMA FOR CLOUDVENDORPROXY ROLE

<table>
<thead>
<tr>
<th>Role Schema:</th>
<th>CloudVendorProxy (CVP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Handles the management of resources from cloud providers.</td>
</tr>
<tr>
<td>Protocols &amp; Activities:</td>
<td>AwaitRequest, ProcessRequest, Authorize, PerformOperation, SendResponse</td>
</tr>
<tr>
<td>Permissions:</td>
<td>reads supplied cloudProviderInformation, supplied operationInformation, supplied securityToken, generates operationOutput</td>
</tr>
<tr>
<td>Liveness:</td>
<td>true</td>
</tr>
</tbody>
</table>

Table VII
AGENTS MODEL

<table>
<thead>
<tr>
<th>ServiceProviderProxy</th>
<th>ServiceAgent</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecurityManager</td>
<td>SecurityManagementAgent</td>
</tr>
<tr>
<td>ServiceManager</td>
<td>ServiceManagementAgent</td>
</tr>
<tr>
<td>ServiceDeployer</td>
<td>DeploymentAgent</td>
</tr>
<tr>
<td>ServiceBroker</td>
<td>BrokerAgent</td>
</tr>
<tr>
<td>CloudVendorProxy</td>
<td>VendorAgent</td>
</tr>
</tbody>
</table>

are single instances because they hold information about security and services.

E. Services model

The focus is again on the ServiceProviderProxy role and ServiceManager role, the available services being described in Table VIII.

From the Authenticate protocol we have the “authentication” service. Its inputs are the provider identification and credentials and its output is the security token. The service has no pre-conditions or post-conditions.

The service associated with the DelegateOperation protocol is operation delegation. This service requires the security token as well as operation information as input and will return the operation output. The pre-condition is that a valid security token is given; the post-condition requires operation output to exist.

Another service is coming from the Authorize protocol: authorization. Authorization involves sending the security token to be validated and receiving a validation. The validation must not be null.

The last service, instance management, is derived from the ManageInstances protocol. In has as input the security token and information and outputs instance information. It requires that a valid security token is provided and as post-condition, instance information must be generated.

Table VIII
THE SERVICES MODEL

<table>
<thead>
<tr>
<th>Service</th>
<th>authentication</th>
<th>operation delegation</th>
<th>authorization</th>
<th>instance management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>providerIdentifier</td>
<td>securityToken, operationInformation</td>
<td>securityToken</td>
<td>securityToken, serviceInformation</td>
<td></td>
</tr>
<tr>
<td>Pre-condition</td>
<td>securityToken ≠ bad</td>
<td>true</td>
<td>securityToken ≠ bad</td>
<td></td>
</tr>
<tr>
<td>Post-condition</td>
<td>operationOutput ≠ nil</td>
<td>validation ≠ nil</td>
<td>instanceInformation ≠ nil</td>
<td></td>
</tr>
</tbody>
</table>

F. Acquaintance model

Finally, the acquaintance model describes existing communication between agents as shown in Figure 2. As can be seen, both the ServiceManagementAgent and the SecurityManagementAgent are the central point for all other agents.

V. CONCLUSIONS AND FUTURE WORK

This paper offers a partial analysis of a multi-agent system for cloud governance using the Gaia methodology. The analysis covers several roles in the system (ServiceProviderProxy, CloudVendorProxy, ServiceManager, SecurityManager, ServiceDeployer and ServiceBroker) along with the associated interaction, agent, services and acquaintance models. While the paper covers the analysis stage, as described in the Gaia methodology, the design stage will follow, and coupled with agent implementations.
The possibility of creating marketplaces with common rules which facilitate the dynamic selection and consumption of services and resources becomes available only when governance is coupled with existing PaaS and management solutions [10].

Cloud Governance is a perfect fit for highly concurrent, distributed, and fault tolerant developments due to the fact that it is built on requirements like security, reliability, trust, portability, interoperability, or failover and it covering activities in diverse domains ranging from security providers, acquisitions, contracts, SLAs, provisioning, management to monitoring and audit.

ACKNOWLEDGMENTS

This work was partially supported by the grant of the European Commission FP7-ICT-2009-5-256910 (mOSAIC), FP7-REGPOT-CT-2011-284595 (HOST), and Romanian national grant PN-II-ID-PCE-2011-3-0260 (AMICAS). The views expressed in this paper do not necessarily reflect those of the corresponding projects’ consortium members.

REFERENCES


