Self-healing Multi-Cloud Application Modelling

Erkuden Rios Tecnalia Research & Innovation C/ Geldo, 700 Derio, Spain 48160 erkuden.rios@tecnalia.com

Eider Iturbe Tecnalia Research & Innovation C/ Geldo, 700 Derio, Spain 48160 eider.iturbe@tecnalia.com

Maria Carmen Palacios Tecnalia Research & Innovation C/ Geldo, 700 Derio, Spain 48160 maricarmen.palacios@tecnalia.com

ABSTRACT

Cloud computing market forecasts and technology trends confirm that Cloud is an IT disrupting phenomena and that the number of companies with multi-cloud strategy is continuously growing. Cost optimization and increased competitiveness of companies that exploit multi-cloud will only be possible when they are able to leverage multiple cloud offerings, while mastering both the complexity of multiple cloud provider management and the protection against the higher exposure to attacks that multi-cloud brings.

This paper presents the MUSA Security modelling language for multi-cloud applications which is based on the Cloud Application Modelling and Execution Language (CAMEL) to overcome the lack of expressiveness of state-of-the-art modelling languages towards easing: a) the automation of distributed deployment, b) the computation of composite Service Level Agreements (SLAs) that include security and privacy aspects, and c) the risk analysis and service match-making taking into account not only functionality and business aspects of the cloud services, but also security aspects. The paper includes the description of the MUSA Modeller as the Web tool supporting the modelling with the MUSA modelling language. The paper introduces also the MUSA SecDevOps framework in which the MUSA Modeller is integrated and with which the MUSA Modeller will be validated.

KEYWORDS

Cloud, Multi-cloud, security, modelling, deployment.

Cross-Reference:

Rios, Erkuden, Eider Iturbe, and Maria Carmen Palacios. "Self-Healing Multi-Cloud Application Modelling." Proceedings of the 12th International Conference on Availability, Reliability and Security - ARES '17 (2017). DOI:10.1145/3098954.3104059

1 INTRODUCTION

Cloud computing forecasts and market estimates of the last years confirm that Cloud is an IT disrupting phenomena with a tremendous growth potential [5]. According to Gartner [7], by 2020 the 'Cloud Shift' (shifting from traditional IT offerings to cloud services) will impact more than $1 Trillion in IT spending.

In the last years the percentage of companies that have a strategy to use multiple clouds for running applications or for experimentation is continuously growing [25]. In such landscape, cost optimization and increased competitiveness of companies that exploit multi-cloud will only be possible when they are able to leverage multiple cloud providers offerings, while mastering both the complexity of multiple cloud provider management and the protection against the higher exposure to attacks that multi-cloud brings.

Multi-cloud applications are those that take most out of the cloud by combining multiple cloud offerings, i.e. those which components use or are distributed in heterogeneous cloud resources, thus facing highly complex challenges with regards to both distributed deployment automation as well as component and overall application security assurance.

As opposite to cloud federations, in multi-cloud paradigm the cloud service providers which services are combined by the user do not need necessarily to have previously reached to an agreement about the way or model in which the services will be offered.

Therefore, a number of problems arise on how to model and decide application components distribution in the clouds, as well as issues on how to specify security properties offered and required (from the cloud providers) by the multi-cloud application individual components and the overall application.

In order to solve these problems, we present a novel approach for addressing security in the modelling of multi-cloud applications. The formalisms and supporting tool presented herein have been developed in the context of the EU-funded MUSA project [18]. The solution relies in the extension of the Cloud Application Modelling and Execution Language (CAMEL) [20] to address richer deployment requirements specification as well as fully-fledged security behaviour specification addressing cases when application components require and provide security capabilities.

The current state-of-the-art modelling languages lack expressiveness that eases: a) the automation of distributed deployment, b) the computation of composite Service Level Agreements (SLAs) that include security and privacy aspects, and c) the risk analysis and service match-making taking into account not only functionality and business aspects of the cloud services, but also security aspects, and this paper addresses such gaps.

The rest of the paper is organized as follows: Section 2 explains state-of-the-art and challenges of multi-cloud application security

modelling. In Section 3, we introduce the MUSA SecDevOps framework and workflow for multi-cloud applications, which gives the context of the work developed. Section 4 details the MUSA Security Domain Specific Language and in Section 5 we describe its supporting tool, named MUSA Modeller, included in the MUSA framework. Section 6 describes the validation scenarios of the work presented. Finally, Section 7 presents the conclusions and future work.

2 MULTI-CLOUD APPLICATION SECURITY MODELLING

In the last years the cloud based infrastructure and platform services offering has increased notably as well as the number of providers. One of the main problems when architecting and operating (multi-)cloud applications is that cloud providers support different interfaces for different sets of services. Furthermore, in order to exploit multi-cloud potential, different architectural models can be adopted to enlarge security [1]:

- replication of applications, deploying the same system in more than one provider so as malicious attacks can be easily discovered comparing operation results;
- partition of application system into tiers, which allows separating logic from data, and thus minimizing risks of attacks or incidents in both parts at a time;
- partition of application logic into fragments in order to obfuscate the overall application logic to providers;
- partition of application data into fragments to prevent a single provider reconstructing data, safeguarding confidentiality.

Exploiting multi-cloud for higher availability and security means that DevOps teams in charge of developing and operating (multi-)cloud applications need to learn, systematise and automate the process behind the provisioning of services when deploying the application components. Among other tasks, they need to learn how to create Virtual Machines (VMs) and how to choose the right VM sizes for each application service or component.

To address these challenges, current research stakeholders propose to take advantage of the well-known Model Driven Engineering (MDE) techniques to configure the deployment of cloud applications. According to the work in [22], application models can be specified using general-purpose languages like the Unified Modelling Language (UML) [8]. However, to fully unfold the potential of MDE, models are frequently specified using domain-specific languages (DSLs), which are tailored to a specific domain of concern.

The MDE techniques become really interesting for multi-cloud application specification when the model captures multi-concern information, expressed at high level first and detailed at low level application platform afterwards, and when the model is enacted at runtime. This allows for a seamless alignment of design decisions with actual deployment and application execution. In this line work CloudML [28] and CAMEL [22] (which includes CloudML as Deployment model for expressing deployment needs) languages. CAMEL was developed as part of the research work in PaaSage [19] and CloudSocket [17] EU-funded projects.

Another great exponent in the literature on cloud-based applications modelling is the Topology and Orchestration Specification for Cloud Applications (TOSCA) language developed by OASIS [15] which provides a language for specifying the components comprising the topology of cloud-based applications along with the processes for their orchestration. Another example is the Cloud Application Modelling Language (CAML) defined in the ARTIST project [16] which realised CloudML as a UML internal DSL based on extensions to the deployment meta-model in terms of a library and profiles capturing domain knowledge.

Following a similar approach of that of PaaSage Security DSL [14], in MUSA project [4] that contextualises the work presented in this paper, CAMEL has been adopted in order to cope with the modelling of multi-cloud applications.

CAMEL includes two main security-oriented meta-models [22]: the *Security* meta-model to support the specification of security requirements posed by users and capabilities of cloud providers (in form of security controls and service level objectives) and the *Organisation* meta-model that captures security-oriented information about organisations including organisation security policies, users and roles.

The application requirements in CAMEL are mainly captured by the *Requirements* meta-model. Thus, both the *Security* and the *Requirements* meta-models can complementarily capture security requirements. CAMEL offers support to the following tasks [22]: (i) matching in deployment phase security capabilities and requirements of the application to the security controls offered by the cloud providers; (ii) monitoring and assessing security service level objectives (SLOs) which can be mapped to adaptation rules in order to adapt the structure or behaviour of an application to exhibit the security level required.

Although both PaaSage and MUSA projects offer support to both tasks, they differ in the modelling aspects, mainly because PaaSage relies on CAMEL model enactment and MUSA follows a different approach as explained in Section 3. The rationale for selecting CAMEL in our approach on top of other versions of CloudML and TOSCA is described in Section 4.1.

3 THE MUSA SECDEVOPS FRAMEWORK FOR MULTI-CLOUD APPLICATIONS

3.1 The MUSA framework

Multi-cloud solutions pose new challenges when trying to add value to overall cloud client experience [29].

The MUSA framework [26] introduced in this paper aims at ensuring that the desired security and privacy levels are reached in all types of multi-cloud environments, including those that combine multiple architectural scenarios as described in Section 2.

To this aim, the MUSA framework combines preventive security with reactive security. For prevention, the MUSA framework supports Security by Design practices in the development as well as embedding in the application the required security mechanisms, the so called *MUSA Enforcement Agents*, which will enable the self-healing of the application in operation. For reaction, the MUSA framework includes monitoring of the application at runtime to detect, notify and early mitigate security incidents, so multi-cloud application providers can be informed and promptly react to security problems or attacks.

In order to ensure the preventive security mechanisms to be embedded in the application components and aligned with reactive

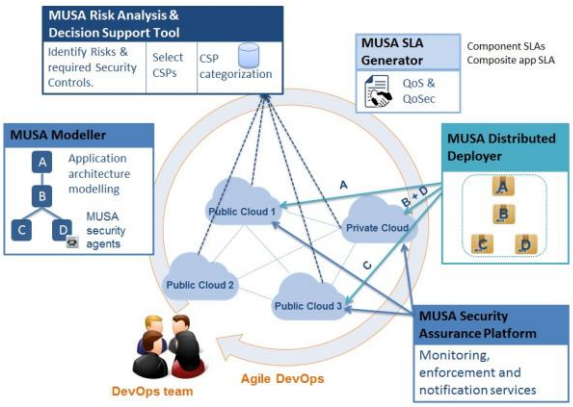

Figure 1: MUSA approach and framework tools.

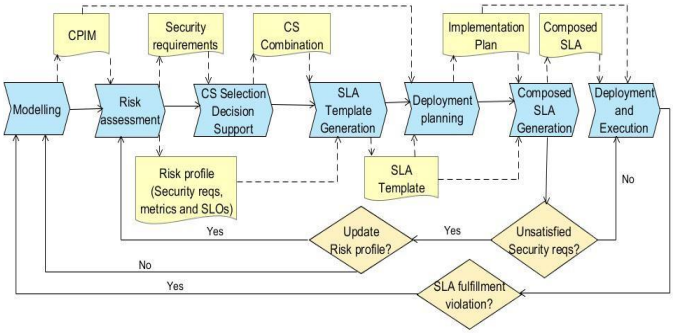

Figure 2: MUSA overall process.

security measures, MUSA holistic framework offers a number of mechanisms and supporting tools that seamlessly work together along all phases of the application lifecycle, as shown in Fig 1.

3.2 The MUSA workflow

The MUSA framework is intended to provide integrated support to addressing security aspects of multi-cloud applications in all the steps of the engineering lifecycle, as illustrated in the process of Fig 2.

During the *design phase*, the DevOps team first models the Cloud Provider Independent Model (CPIM) of the multi-cloud application using the MUSA Modeller. The CPIM is a MUSA extended CAMEL model specification of the multi-cloud application in a level of abstraction independent from specific information about the Cloud Service Providers (CSP) the application components will use or be deployed in.

Once the application CPIM is specified, the DevOps team obtains the security requirements in the risk assessment step. In this step, together with the security requirements, the DevOps team can analyse other criteria as well, such as business criteria, and can search for the cloud services that best match all the criteria, by relying on the use of the MUSA Decision Support Tool (DST) [4].

Modelling, risk analysis and cloud services selection are made following an iterative process that allows identifying which security requirements (if any) were not possible to address with security controls offered by the cloud services under study, and therefore the DevOps team can update the CPIM at modelling step to specify the use of MUSA security agents that offer the pending security controls (if available).

Having selected the combination of cloud services that best match the requirements of the multi-cloud application and having previously defined the security requirements, the DevOps team can generate the *Security SLA templates* for the components of the multi-cloud application. These Security SLA templates will be stored in the SLA Repository and will be retrieved by the MUSA Deployer, so it can generate the Implementation plan for the multi-cloud application. Once the Implementation plan is generated, the MUSA Deployer shares it with the SLA Generator [2], so this tool can generate the *Composite Security SLA* for the whole multi-cloud application.

Afterwards, at *deployment phase*, the MUSA Deployer is invoked by the DevOps team in order to deploy (by following the Implementation plan) both the multi-cloud application components and the corresponding MUSA agents declared in the CPIM.

Finally, at *runtime* or *operation phase*, the MUSA Security Assurance platform [27] starts monitoring the multi-cloud application based on the final SLAs and the Implementation plan.

If the MUSA Security Assurance Platform detects any alert event or violation of the SLAs in place, it notifies to the DevOps team and the appropriate reactive measure is triggered. The reaction to security incidents in MUSA relies on different mechanisms depending on the cause of the incident. Reactive measures include the re-deployment of multi-cloud application component(s) or even the application re-design. In the re-design phase the need of including MUSA Enforcement Agents can be evaluated again in order to try to address the security incident detected. Adaptation of multi-cloud applications at execution is supported in MUSA by the enforcement services in the MUSA Security Assurance Platform, that remotely control the configuration, activation and deactivation of some of the MUSA Enforcement Agents. We refer to future publications of MUSA for detailed explanation of the enforcement services offered.

4 MUSA SECURITY DSL

The MUSA Security Domain Specific Language (DSL) builds on top of the CAMEL language and the main contributions are related to its expressiveness to define and configure multi-cloud applications at design-time. Such a powerful definition can be used later on by other MUSA tools to perform the risk analysis and to generate the individual components' security SLAs and the *Composite Security SLA*. Furthermore, this allows deploying and monitoring the security properties of multi-cloud applications and its components at run-time. In brief, the main innovations achieved in MUSA are the improvements to the CAMEL language, the accompanying model syntax and semantics verification rules, and the development of a Web-based modelling tool (named MUSA Modeller) as explained in the following. The MUSA Security DSL is shown in Fig 3.

4.1 Rationale for CAMEL language selection

The main reasons for selecting CAMEL language as the basis of the MUSA security DSL are explained in the following.

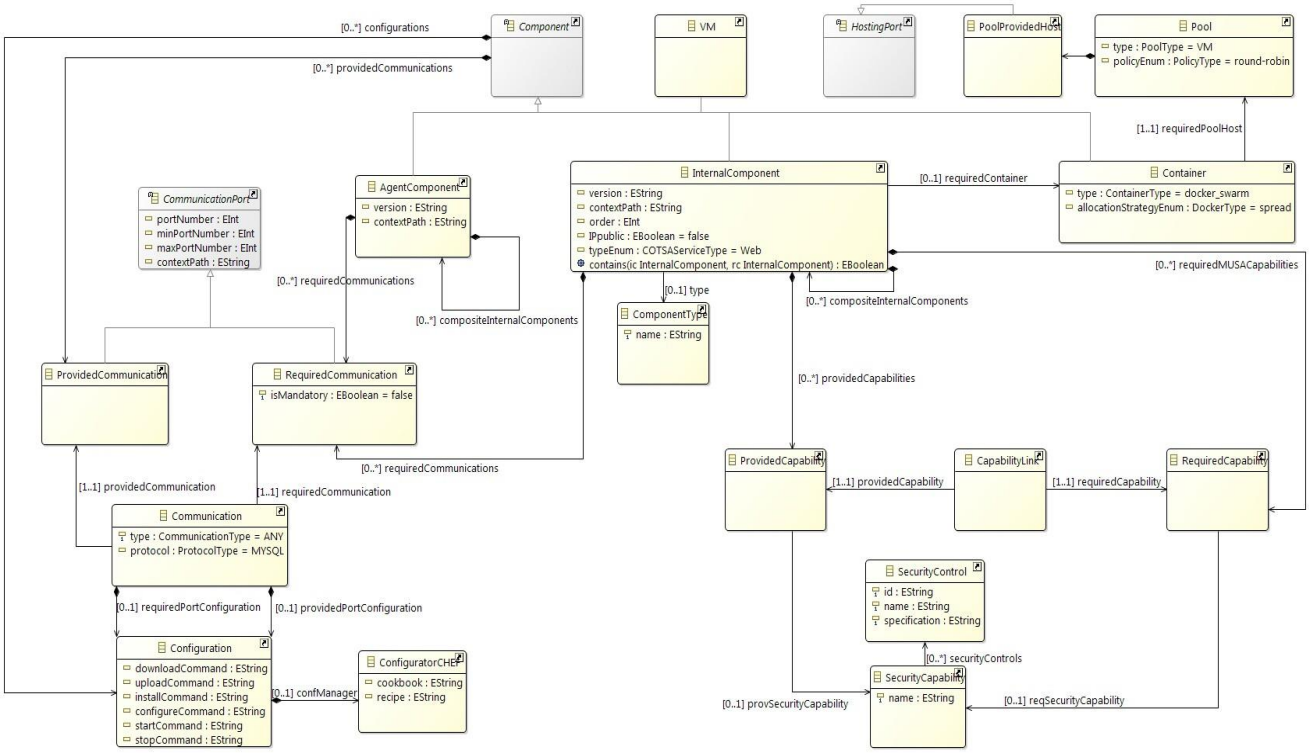

Figure 3: MUSA Security DSL.

First, the enactment of the multi-cloud application model at runtime is not needed in MUSA. This is because in MUSA we do not rely on a single model of the multi-cloud application to capture all the aspects, but on a number of models. In MUSA we use the CAMEL model of the application to express its architecture, security and deployment requirements, but in addition we use: *Security SLA templates* of the components to capture the required security features over the cloud providers, and the *Composite Security SLA* to express the guarantees of the overall application. The *Composite Security SLA* serves in MUSA to control the application behaviour at runtime. Multi-cloud applications re-designs will be driven by actual measurements taken of the Service Level Objectives and metrics stated in the *Composite Security SLA*. In MUSA we also use the Decision Support Tool data model for expressing business and security requirements over the cloud providers, and the deployment Implementation plan as the cloud platform dependent deployment model.

Second, CAMEL includes CloudML as *Deployment* meta-model, therefore, adopting it implies adopting CloudML too. Besides, CAMEL includes also other meta-models such as *Security* and *Requirements* meta-models that are valuable when placing the focus on security. CAMEL follows the same approach that CloudML in relation to the provision of a single set of abstractions and APIs so that developers can define declaratively: (i) the application architecture made of components, (ii) their use/host relationships so that they can be properly configured and deployment orders automatically derived, (iii) constraints on the characteristics of the required types of VMs and (iv) the execution commands to provision application components.

At this point, it is important to note that MUSA project was born with the objective of supporting cloud adoption by addressing current security open issues, since companies are reluctant to adopt cloud computing because of the difficulty in evaluating the trade-off between cloud benefits and the additional security risks and privacy issues it may bring. They have to deal with the security of the individual components as well as with the overall application security including the communications and the data flow between the components. Therefore, it is required to adopt a language that allows end-users to friendly and easily create and deploy security-aware components balancing security with performance properties.

As opposite to PaaSage approach, where the low level security information is captured in the CAMEL based Cloud Provider Specific Model (CPSM), in MUSA it is captured and managed by two other specialised languages, respectively, the MUSA Security SLA model (extended from that of SPECS project [23] to address multi-cloud scenarios) and the MUSA Decision Support Tool's Cloud Service Provider data model that allows defining full flavoured business and security requirements and capabilities for multi-factor service match-making.

Third, opposite to TOSCA, MUSA end-users had already experienced with CAMEL and liked the language expressiveness because it was the only language that already provided rich Security model and Requirements model (for deployment). CAMEL and TOSCA

basically share the same purpose, but CAMEL results in being less complex and easy to use.

Furthermore, TOSCA community did not offer advanced tool support at the time when MUSA solution was defined.

Last but not least, the feedback received from the MUSA project end-users about previous experiences with multi-cloud modelling languages and tools was considered as fundamental. One of the end-user teams had experienced with CAMEL and liked the expressiveness of the language for multi-cloud application description, although security aspects were not considered by them previously. About the modelling tools, they had previously used PaaSage CAMEL Textual Editor [21] and other graphical editors (such as Modelio [3]) and preferred the non-graphical editor that included friendly functionality such as identification of *required* vs. *optional* attributes, auto completion capabilities and model validation among others. Thus, these are exactly the features we aim to maintain in the MUSA Modeller. As conceptually both graphical and textual editors allow for the same degree of detail and completeness, we decided to develop our MUSA Modeller as a text editor.

4.2 Innovations for security behaviour specifications

The MUSA innovations to CAMEL language include the enhanced component security behaviour characterization, which addresses concepts required to support both composition of components' security SLAs and risk analysis.

4.2.1 Classification of components by their nature which allows describing what the component does. This information is required by the MUSA SLA Generator to create the *Composite Security SLA* of the multi-cloud application from individual components' SLAs and by the MUSA Risk Analysis tool in order to identify the security threats and risks at component level. Our CAMEL extension allows specifying the type of the component; in fact, the component can be classified by two features: WHAT and HOW. While WHAT indicates the type of the functionality delivered by the component, HOW indicates the way the component is delivering such functionality. Currently, three types of HOW have been defined:

- COTS, which refers to Commercial off-the-shelf software that the application uses.
- SERVICE, meaning that the component is not a commercial package but developed by the DevOps team responsible for the multi-cloud application engineering.
- AGENT, i.e. a MUSA Agent component provided by the MUSA framework and available in the MUSA Security Agent Catalogue.

The types of WHAT include:

- in case the component is COTS or SERVICE, the possible WHAT values are: Web, Storage, IDM or Firewall. Web refers to any functionality provided through a Web interface, Storage refers to data storage solutions (e.g. MySQL), IDM stays for Identity Management and Firewall for any software solution that protects resources from unauthorised access.
- in case the component is AGENT, the possible values of WHAT come from the list of the agents in the MUSA Security Agent Catalogue.

4.2.2 Security Controls information that properly supports Security Control Framework families. For example, *name* attribute has been updated to <Family>-<Number>(Number) format. In addition, the *subdomain* attribute in the Security Control entity now is optional instead of compulsory.

The CAMEL extension developed in MUSA allows specifying which security capabilities are required and provided by each multi-cloud application component. The security capabilities are defined in the model by selecting and grouping the security controls part of the capability. The security controls of a Security Control Family are predefined and the list is offered to the user by the MUSA Modeller to ease the selection of the ones included in the capability; currently, the security controls from the NIST SP 800-53 rev.4 [6] are supported. In the following example two security capabilities CAP1 and CAP2 are defined, the first with three security controls and the second with only two.

```
security model SEC {
security capability CAP1 {
controls [MUSASEC.AC-11(1), MUSASEC.AU-13(2),
MUSASEC.AC-17(6)]
}
security capability CAP2 {
controls [MUSASEC.AC-11(1), MUSASEC.AC-17(6)]
}
}
```

Once the security capabilities are defined in the CAMEL (in the security model part of the model), the user can specify what security capabilities the components require and/or provide. In the following example, Comp1Cap is a provided capability and Comp1CapReq a requested one.

```
provided security capability Comp1Cap {
security capability SEC.CAP2
}
required security capability Comp1CapReq
```

When a component requires a specific security capability from another component (in the example, Comp1CapReq) then the matching of the capability needs to be modelled as follows.

```
capability match Comp1ToComp2 {
from Comp1.Comp1CapReq to Comp2.Comp2Cap
}
```

4.3 Innovations for multi-cloud deployment

Other MUSA innovations to CAMEL language address improvements for enhancing the expressiveness of the deployment requirements, as follows:

4.3.1 Explicit characterization of the nature of the IP address associated with virtual machines ponents will be deployed. At deployment phase, when acquiring new cloud resources such as VMs,

the system operator needs to indicate whether a public IP address is required. The CAMEL extension allows specifying whether the IP address required for a VM should be public or not by the *IP public* attribute on each component. The possible values for this attribute are: true or false.

4.3.2 Specification of the deployment order of the application components. Dealing with multi-cloud environments, it is critical to identify the order in which each component should be deployed and configured, since there are inter-dependencies among the components that are part of the same application. For example, the start up of a component may require that previously another component is already up and running. The CAMEL extension allows specifying the order in which the components are required to be deployed. This can be done by using the *order* attribute for each component. The expected value for the *order* attribute is an integer number.

4.3.3 Explicit definition of data exchange protocols. In the MUSA extended CAMEL, users can model the communications between the components (e.g. by setting the IP addresses and ports in the configuration of the components) and specify the need to use a specific data exchange protocol (e.g. MySQL, OAuth, Other).

4.3.4 Modelling of dynamic configurations of communications between components. In CAMEL, users model the communications between the components in a static way (i.e. through specific port numbers and operating system configuration variables). However, in MUSA, dynamic characteristics have been introduced such as context paths (instead of IP addresses) and dynamic port ranges. Such new capabilities are useful, for instance, to configure explicitly inbound traffic when users deploy components in Docker containers.

4.3.5 Modelling of deployment handlers. In CAMEL the user can model components and associate deployment instructions for installing, configuring, starting, and stopping the components on virtual machines. However, such deployment instructions are restricted to scripted commands and CAMEL lacks support to the specification of configuration management tools such as Cloudify [12], Puppet [24], or Chef [9]. Therefore, in MUSA this gap between multi-cloud application models and these advanced frameworks has been faced via the new *Configuration* entity and its associated concepts (e.g. *cookbooks* and *recipes* in case of Chef).

4.3.6 Modelling of PaaS layer elements. CAMEL lacks support to the description of architectures where the application components are not directly deployed in Virtual Machines (VMs) but in containers. Our extension allows specifying the container type that will be used in deployment and defining the component allocation strategy it should follow, even in cases when the container uses VM pools. The new elements in our extension include:

- *pool*: is a cluster of VMs. This cluster can be used by a container or directly by a component, for example, a database solution that is capable of managing a cluster.
- *manager*: the VM in a pool that will act as the manager vs. the rest which will be workers.
- *container*:
  - *type*: the container solution to use, for example, Docker Swarm [10].
  - *allocationStrategy*: defines the allocation strategy of the containers on top of the acquired VMs for resource optimization (e.g. automatically scheduling container workloads). It supports the following four values:
    - *spread*: balance containers across the VMs in a pool based on the available CPU and RAM of the VMs.
    - *binpak*: schedule containers to fully use each VM capacity. Once the full VM capacity has been used, the container moves on to the next one in the pool.
    - *random*: choose a VM randomly.
    - *custom*: the user defines the specific VMs in which the containers should run.

4.3.7 Refinement of security aspects in Organisation, User, Credentials and Role entities. A number of enhancements to CAMEL have been made in order to manage the authorisation of different roles in the DevOps team to multi-cloud application deployment execution. For instance, in MUSA the types of credentials available to authenticate a user have been extended. Among other changes, it has been added expiration date to Credentials and additional parameter properties to User entity.

4.4 Innovations for *self-healing* capability of multi-cloud applications

Considering self-healing as the capability of a multi-cloud application of being able to self-control or modify its security behaviour at runtime so as security incidents or attacks are corrected or mitigated, the self-healing is enabled in MUSA by the MUSA Enforcement Agents.

As their name suggests, the MUSA Enforcement Agents enforce multi-cloud application security properties at runtime such as access control, security vulnerability scanning or Denial of service mitigation mechanisms. For these mechanisms to work, they need to be deployed at the same time as the application components are deployed. Some of these mechanisms require to be deployed together with (in the same VM) the component they will enforce the property on. Therefore, the MUSA extended CAMEL allows the definition of MUSA Security Enforcement Agents as Internal Components of the application, similarly to application components themselves, so as they can be included in the deployment plan. Such agents are already pre-defined in the MUSA Security Agent Catalogue so users are able to re-use and configure them in a friendly way. Some of these agents are always on and some will be managed through the enforcement services in the MUSA Security Assurance Platform.

5 MUSA MODELLER

5.1 MUSA Modeller Architecture

The MUSA Modeller is a web editor tool that allows the creation and maintenance of (MUSA extended) CAMEL models. Through these models it is possible to define a complete specification of the requirements needed by an application to be deployed in a secure multi-cloud environment. MUSA Modeller has leveraged one of the new capabilities of the Xtext technology [11]: the Xtext Web

editor support. Therefore, multi-cloud application models can be edited and updated remotely by end-users while they are stored in shared repositories. All in all, without any program installation and using any web browser since its JavaScript-based API allows adding language-specific features such as auto completion and live validation.

As the initial step of the design phase, the DevOps team creates the Cloud Provider Independent Model (CPIM) of the multi-cloud application using the MUSA Modeller tool. The generated CPIM will be used as input to the risk assessment, the SLA generation and the multi-cloud application deployment processes.

At high level, the MUSA Modeller is structured as follows:

- A *Web component*, the GUI from which the end-users access the MUSA Modeller tool and exploit all the internal web-services offered. It also includes the web Xtext libraries that offer syntactic and sematic services for remote management of syntax validation and auto completion.
- A *Server component*, which is the core of the MUSA Modeller since it implements all the business functionality. Moreover, it offers a series of web services that are consumed by the Web component.
- A *Database component*, which manages all the data stored in the central database. It uses the Hibernate framework [13] that allows the abstraction of relational database engines.

5.2 MUSA Modeller API

As explained in the previous chapter, the Server component of the MUSA Modeller offers a series of REST API interfaces that support the following functionality:

- Creation of a new multi-cloud application model in (MUSA extended) CAMEL language. End-users can instantiate previously defined templates for particular component types or applications.
- Definition and storage of multi-cloud application component templates for reusing CAMEL models of the components.
- Edition and storage of multi-cloud application models, where application models can be defined by multiple end-users.
- Model checking for syntax and semantic correctness and integrity of the created models. The tool provides messages of warnings and errors whenever a non-conformity is identified in the model.
- the selection of Security Controls previously identified and stored in the MUSA Security Service Level Agreement Catalogue.
- the selection of MUSA Security Enforcement Agents previously identified and stored in the MUSA Security Agent Catalogue. See Fig 4.

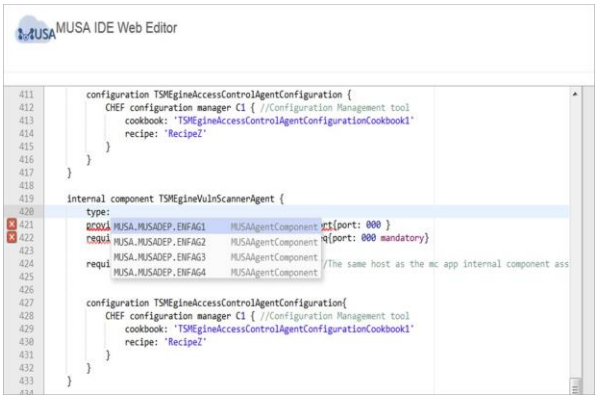

Figure 4: MUSA Modeller support for Security Agent selection.

6 VALIDATION IN USE CASES

During the first period of the MUSA project, the evaluation of the MUSA solution has been performed using generic usage scenarios along with two specific use cases, one led by Lufthansa Systems (LHS) and the other by Tampere University of Technology (TUT), both partners of the project. These case studies have been designed to ensure that MUSA framework evaluation takes place in different contexts of multi-cloud deployment as well as diverse security and privacy requirements.

The first use case is related to the flight scheduling application prototype by Lufthansa Systems that is intended to be used by multiple airlines around the world to plan airplane flight schedules. Today's airlines need to permanently revise their schedule plans in response to competitor actions or to follow updated sales and marketing plans, while constantly maintaining operational integrity. This makes schedule management a very complex process. The MUSA Modeller enabled LHS to specify the breakdown of the multi-cloud application into its components as well as components' security capabilities. This includes also the integration of the MUSA security agents into the application and the design of the provisioning and deployment. LHS has validated the MUSA Modeller to help architects and developers of the multi-cloud application components in the security design to gain more focus on the complex field of security requirements analysis and enforcing the security by design principles even for the less experienced colleagues.

In the second use case, the Smart Mobility application developed by Tampere University of Technology provides Tampere citizens context-aware energy efficient smart mobility services and recommendations for transportation means. This case study represents the scenario of SMEs which create applications that exploit services hosted in a number of clouds. As many SMEs, the workgroup that develops this case study does not have a specialized division for cyber security. Because of this, the use of MUSA framework and its tools has allowed easily addressing security aspects at different stages of the application engineering project: design, deployment and runtime.

Specifically, in this case study the MUSA Modeller tool has been evaluated to model the multi-cloud application components and MUSA Enforcement Agents to integrate security controls for high availability, confidentiality and integrity.

The added value of the MUSA modelling language and tool has been already identified in the initial evaluation performed in both case studies. The usefulness and acceptance of the approach was

confirmed not only in the usage of the MUSA Modeller as a stand-alone tool, but more significantly in its integration with the rest of the tools in the MUSA Security DevOps (SecDevOps) framework, particularly in collaboration with Risk Analysis, SLA Generator and Deployer tools. It is expected that the final evaluation of the MUSA solution will enable the validation of all the enhancements defined in the MUSA Security DSL.

7 CONCLUSIONS AND FUTURE WORK

As the number of cloud offerings grows and cloud environments become more and more complex, cloud consumers will need to face multi-cloud challenges related to multiple cloud service combination and holistic protection of application components deployed in distributed heterogeneous clouds.

This paper presents a novel modelling language and supporting tool that address the special needs of multi-cloud applications modelling. The solution overcomes the lack of expressiveness of state-of-the-art modelling by easing both: a) the computation of *Composite Security SLAs* that include security and privacy aspects, and b) the risk analysis and service match-making taking into account not only functionality and business aspects of the cloud services, but also security aspects.

The language and tool presented were developed in the context of the MUSA EU-funded project on the basis of enhancements to the CAMEL language which already provided rich *Requirements*, *Deployment*, *Scalability*, and *Security* meta-models that cover many of the requirements for multi-cloud applications specification. Nevertheless, additional requirements were identified in MUSA in order to address richer deployment and security specification, risk analysis and Security SLA composition, for which extensions to CAMEL have been developed. The modelling tool supporting this extended CAMEL meta-model, the MUSA Modeller, is already integrated with the MUSA framework and available in the MUSA website at www.musa-project.eu

Consequently, the contributions of this paper pave the way to develop security and privacy-aware multi-cloud applications by mastering the expressiveness of security aspects in the CPIM so as they enable integrated SecDevOps support.

In the future, we have identified the need to research on how to leverage the *Scalability* meta-model of CAMEL to define scalability rules as pre-defined configuration of high availability enforcement agents aimed at scaling up (and down) internal components of multi-cloud applications when needed (similar to HA Proxy [30]).

Another important aspect is the research on how to support composability of CAMEL models required to easily combine and refer to CAMEL models of individual components in the creation of multi-cloud applications' models. Furthermore, this would largely increase the models readability and dramatically improve the tool usability.

ACKNOWLEDGMENTS

The MUSA project leading to this paper has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644429. We would also like to acknowledge all the members of the MUSA Consortium for their valuable help.

REFERENCES

[1] Jens-Matthias Bohli, Nils Gruschka, Meiko Jensen, Luigi Lo Iacono, and Ninja Marnau. 2013. Security and privacy-enhancing multicloud architectures. *IEEE Transactions on Dependable and Secure Computing* 10, 4 (2013), 212–224.

[2] Valentina Casola, Alessandra De Benedictis, Massimiliano Rak, and Erkuden Rios. 2016. Security-by-design in clouds: a security-SLA driven methodology to build secure cloud applications. *Procedia Computer Science* 97 (2016), 53–62.

[3] Inc. Eclipse Foundation. 2011. Modelio, The open source modelling environment. (2011). Retrieved June 26, 2017 from https://www.modelio.org/

[4] Jaume Ferrarons, Smrati Gupta, Victor Muntés-Mulero, Josep-Lluis Larriba-Pey, and Peter Matthews. 2016. Scoring cloud services through digital ecosystem community analysis. In *International Conference on Electronic Commerce and Web Technologies*. Springer, 142–153.

[5] Forbes. 2016. Roundup Of Cloud Computing Forecasts And Market Estimates. (2016). Retrieved June 26, 2017 from https://www.forbes.com/sites/louiscolumbus/2016/03/13/roundup-of-cloud-computing-forecasts-and-market-estimates-2016/#7971c3de2187

[6] JOINT TASK FORCE and TRANSFORMATION INITIATIVE. 2013. Security and privacy controls for federal information systems and organizations. *NIST Special Publication* 800, 53 (2013), 8–13.

[7] Gartner. 2016. Gartner Says by 2020 "Cloud Shift" Will Affect More Than $1 Trillion in IT Spending. (2016). Retrieved June 26, 2017 from http://www.gartner.com/newsroom/id/3384720

[8] Object Management Group. 2015. Unified Modeling Language Specification v2.5. (2015). Retrieved June 26, 2017 from http://www.omg.org/spec/UML/2.5/

[9] Chef Software Inc. 2017. Chef technology. (2017). Retrieved June 26, 2017 from https://www.chef.io/chef/

[10] Docker Inc. 2017. Docker Swarm. (2017). Retrieved June 26, 2017 from https://docs.docker.com/swarm/

[11] Eclipse Foundation Inc. 2017. Xtext framework. (2017). Retrieved June 26, 2017 from http://www.eclipse.org/Xtext/documentation/330_web_support.html

[12] GigaSpaces Technologies Inc. 2017. Cloudify. (2017). Retrieved June 26, 2017 from http://cloudify.co/

[13] Red Hat Inc. 2017. Hibernate framework. (2017). Retrieved June 26, 2017 from http://hibernate.org/

[14] Kyriakos Kritikos and Philippe Massonet. 2016. An integrated meta-model for cloud application security modelling. *Procedia Computer Science* 97 (2016), 84–93.

[15] OASIS. 2013. TOSCA 1.0 (Topology and Orchestration Specification for Cloud Applications), Version 1.0. (2013). Retrieved June 26, 2017 from http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html

[16] ARTIST EU project consortium. 2013. Advanced software-based service provisioning and migration of legacy software. (2013). Retrieved June 26, 2017 from http://www.artist-project.eu

[17] CloudSocket EU project consortium. 2015. Business and IT-Cloud Alignment using a Smart Socket. (2015). Retrieved June 26, 2017 from https://site.cloudsocket.eu

[18] MUSA EU project consortium. 2015. Multi-cloud Secure Applications. (2015). Retrieved June 26, 2017 from http://www.musa-project.eu

[19] PaaSage EU project consortium. 2013. A Model-based Cross-Cloud development and deployment platform. (2013). Retrieved June 26, 2017 from http://www.paasage.eu

[20] PaaSage EU project consortium. 2014. Deliverable D2.1.2 CloudML Implementation Documentation. (2014). Retrieved June 26, 2017 from http://www.paasage.eu/images/documents/paasage_d2.1.2_final.pdf

[21] PaaSage EU project consortium. 2015. Deliverable D2.1.3 CAMEL Documentation. (2015). Retrieved June 26, 2017 from https://www.paasage.eu/images/documents/docs/D2.1.3_CAMEL_Documentation.pdf

[22] PaaSage EU project consortium. 2016. CAMEL Documentation v2015.9. (2016). Retrieved June 26, 2017 from http://camel-dsl.org/documentation/

[23] SPECS EU project consortium. 2015. Secure Provisioning of Cloud Services based on SLA Management. (2015). Retrieved June 26, 2017 from http://www.specs-project.eu/

[24] Puppet. 2017. Puppet documentation. (2017). Retrieved June 26, 2017 from https://docs.puppet.com/puppet/

[25] Rightscale. 2017. Cloud Computing Trends: 2017 State of the Cloud Survey. (2017). http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2017-state-cloud-survey

[26] Erkuden Rios, Eider Iturbe, Leire Orue-Echevarria, Massimiliano Rak, Valentina Casola, and others. 2015. Towards Self-Protective Multi-Cloud Applications: MUSA–a Holistic Framework to Support the Security-Intelligent Lifecycle Management of Multi-Cloud Applications. (2015).

[27] Erkuden Rios, Wissam Mallouli, Massimiliano Rak, Valentina Casola, and Antonio M Ortiz. 2016. SLA-driven monitoring of multi-cloud application components using the MUSA framework. In *Distributed Computing Systems Workshops (ICDCSW), 2016 IEEE 36th International Conference on*. IEEE, 55–60.

[28] SINTEF. 2013. Model-based provisioning and deployment of cloud-based systems. (2013). Retrieved June 26, 2017 from http://cloudml.org
[29] Marko Vukolić. 2010. The Byzantine empire in the intercloud. *ACM SIGACT News* 41, 3 (2010), 105–111.
[30] Q HAProxy Wu. 2017. The Reliable, High Performance TCP/HTTP Load Balancer. (2017). Retrieved June 26, 2017 from http://www.haproxy.org/