Future Generation Computer Systems 67 (2017) 206–226

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Towards a security-enhanced PaaS platform for multi-cloud applications

Kyriakos Kritikos [a,*], Tom Kirkham [b], Bartosz Kryza [c], Philippe Massonet [d]

[a] ICS-FORTH, Heraklion, Crete, Greece
[b] STFC, United Kingdom
[c] AGH, Krakow, Poland
[d] CETIC, Belgium

ARTICLE INFO

Article history:
Received 4 April 2016
Received in revised form
10 August 2016
Accepted 6 October 2016
Available online 12 October 2016

Keywords:
Cloud
Security
Meta-model
Policies
Roles
SAML

ABSTRACT

Multi-cloud adaptive application provisioning can solve the vendor lock-in problem and allows optimising user requirements by selecting the best from the multitude of services offered by different cloud providers. To this end, such provisioning type is increasingly supported by new or existing research prototypes and platforms. One major concern, actually preventing users from moving to the cloud, comes with respect to security, which becomes more complex in multi-cloud settings. Such a concern spans two main aspects: (a) suitable access control on user personal data, VMs and platform services and (b) planning and adapting application deployments based on security requirements. As such, this paper addresses both security aspects by proposing a novel model-driven approach and architecture which secures multi-cloud platforms, enables users to have their own private space and guarantees that application deployments are not only constructed based on but can also maintain a certain user-required security level. Such a solution exploits state-of-the-art security standards, security software and secure model management technology. Moreover, it covers different access control scenarios involving external, web-based and programmatic user authentication.

1. Introduction

Cloud computing delivers various advantages, such as cost reduction and adaptive application provisioning to address increasing customer load via exploiting the infinite resources available. As such, an constantly increasing number of organisations makes the step to move to the cloud. To assist in this move, various proprietary and free platforms have been developed supporting cloud-based application management.

However, most of current platforms cannot address the following main issues: (a) application deployments derived are not optimal; (b) vendor lock-in [1]; (c) security aspect is neglected or not completely addressed. The last two issues, in particular, have been extensively reported to constitute the main factors [2] which prevent migration to the cloud.

The aforementioned issues are addressed via multi-cloud application management which enables avoiding vendor lock-in, and optimising as well as adapting application provisioning in multi-clouds by smartly and dynamically selecting the best from the variety of services offered by the current cloud providers. This management type is supported by a respective research trend revealed via recent publications and the acceptance of European research projects activated in this research area. At the industry level, there is a small move towards this type but most of the existing platforms seem to only support hybrid clouds.

In most of the cases, a model-driven approach is followed to partially automate the multi-cloud application lifecycle. This is especially true in the case of the PaaSage project[1] [3]. This project has also developed a rich and extensive domain specific language (DSL) called CAMEL [4] covering various aspects, including deployment, provider, organisation, metric and scalability. Based on this DSL, different modules have been developed which focus

* Corresponding author. Fax: +30 2810 391638.
E-mail addresses: kritikos@ics.forth.gr (K. Kritikos), tom.kirkham@stfc.ac.uk (T. Kirkham), bkryza@agh.edu.pl (B. Kryza), philippe.massonet@cetic.be (P. Massonet).

http://dx.doi.org/10.1016/j.future.2016.10.008
0167-739X/

[1] www.paasage.eu.

on multi-cloud application deployment, monitoring and adaptive provisioning.

However, as in case of most available research prototypes and proprietary platforms, user security requirements and security capabilities of existing cloud services are accounted in the (multi-)cloud application lifecycle. As such, these prototypes can become not practically exploitable. To equip the PaaSage platform with a distinguishing feature, it has been decided to extend CAMEL with a security sub-DSL [4,5] covering most of the main security notions, such as security controls, requirements and SLOs. In addition, the deployment planning component has been extended [6] to derive security-aware deployment plans satisfying the security requirements posed. To also support adaptive provisioning of multi-cloud applications, this article presents an extension the CAMEL Scalability Rule Language (SRL) [7] sub-DSL focusing on the ability to express (security-aware) adaptation rules. Via such extension and the enhancement of the PaaSage platform adaptation capabilities, such rules can now be inspected at runtime and respective security adaptation actions are fired to remedy for any security SLO violation or still satisfy the security SLOs posed.

Another distinguishing PaaSage feature is the capability to store user information in terms of application models and scalability rules, as a side-effect of a model-based approach, as well as share such information among all platform users via exploiting the facilities of a Social Network. Via this knowledge sharing, the cloud can be better utilised and multi-cloud applications can be better managed. To also secure the access to platform facilities, this paper proposes a security architecture and solution enforcing a specific authentication and authorisation level over user models and platform services. This solution relies on existing security standards, like SAML, and model-based technology, thus not having to develop everything from scratch.

The proposed solution is suited from multi-tenancy by enabling constructing personal information spaces for organisations and their users. It also caters for different access control scenarios, including both external, programmatic and web-based user authentication. It also relies on an administration API allowing each organisation administrator to manage its organisation security-oriented information, in terms of access control policies indicating which organisation roles can access which platform services and which parts of the organisation information space. Such an API can be definitely exploited to build administration UIs enabling administrators to edit organisational policies without the explicit knowledge of a certain policy language.

By covering the above security aspects, the PaaSage prototype becomes a platform that takes security seriously by considering different security requirement types and controlling the access to its resources and services. This is a very distinguishing feature that cannot be observed in other related PaaS platforms.

The overall PaaSage architecture which covers both security aspects works perfectly for the case of one PaaSage platform instance. However, this article also shows that such an architecture can be utilised in the case of multiple PaaSage platform instances with a quite small number of extensions. Such extensions will surely further promote the sharing of knowledge related to the cloud as they would enable to connect disparate organisations that are members of different instances in the distributed platform system.

Compared to our previous work, this article presents novel contributions related to: (a) the coverage of the security-aware multi-cloud application provisioning aspect; (b) the proposal of an extensive architecture able to cover both security aspects; (c) the proposal of extensions on existing CAMEL meta-models; (d) the coverage of additional but important details with respect to our access control solution.

The rest of the article is structured as follows. Section 2 provides background information about PaaSage and its security-sub-DSL. Section 3 analyses our extended PaaSage architecture that now covers both security aspects. Section 4 exemplifies how the extended PaaSage architecture is applied on a certain use case. Section 5 analyses the main extensions on two CAMEL meta-models. Section 6 analyses important details regarding our access control solution for the platform resources and services. Section 7 explicates how the proposed architecture can evolve into a multi-instance platform setting. Section 8 reviews related work. Finally, Section 9 concludes the article and draws directions for further research.

2. Background

2.1. PaaSage architecture

PaaSage follows a model-driven approach that relies on CAMEL [8] which extensively captures various aspects in multi-cloud application lifecycle. This approach is based on a particular architecture depicted in Fig. 1 comprising three main modules: (a) *Upperware*, (b) *MetaDataDataBase* (*MDDB*) and (c) *ExecutionWare*. The functionality of these modules along with details about other components important for the success of our security solution is now analysed.

The *Upperware* obtains user application and requirement models, constructs the respective application profiles and then maps user requirements to specific deployment plans by using the *Reasoner*. It is also responsible for performing global adaptation for the application deployment to close the design-time adaptation loop. The *Executionware* retrieves and enforces the application deployment plans in a multi-cloud manner, by being able to interact with and manage those individual heterogeneous (private or private) cloud infrastructures involved in the deployment plan thus being able to raise the abstraction level and deal with the respective heterogeneity, while also monitors respective resource and user-specific metrics. Via monitoring, it is also able to evaluate scalability rules and adapt the application deployment, if needed, in a cloud-specific manner, thus closing the local adaptation loop. *MDDB* provides a model repository enabling the management of (user) models, including their persistency, sophisticated querying, and event publication mapping to their updating, and can also be used as a communication medium between PaaSage components and modules. It is also able to derive new, added-value facts by exploiting respective rules fired by a *KnowledgeBase*. Such facts can support tasks of other components. For instance, they can assist in speeding-up the reasoning process if they indicate best deployments for certain application components or the whole application based on past execution histories of other similar applications.

The CDO[2] technology was used to construct the MDDB *CDORepository* which enables EMF model management and persistence. This repository exhibits various features, including transactionality, lazy loading and fault-tolerance. It can also exploit a multitude of different underlying database management systems for model persistence. *CDOServer* is a service on top of *CDORepository* accepting requests from *CDOClient*s to perform model-based management actions. As such, the *CDOClient* can be an internal component of any PaaSage module/component that needs to manage models over the *CDORepository*.

There are two main entry points in the PaaSage platform to enable users exploiting its facilities. The Service Point (SP) is a

[2] http://wiki.eclipse.org/CDO.

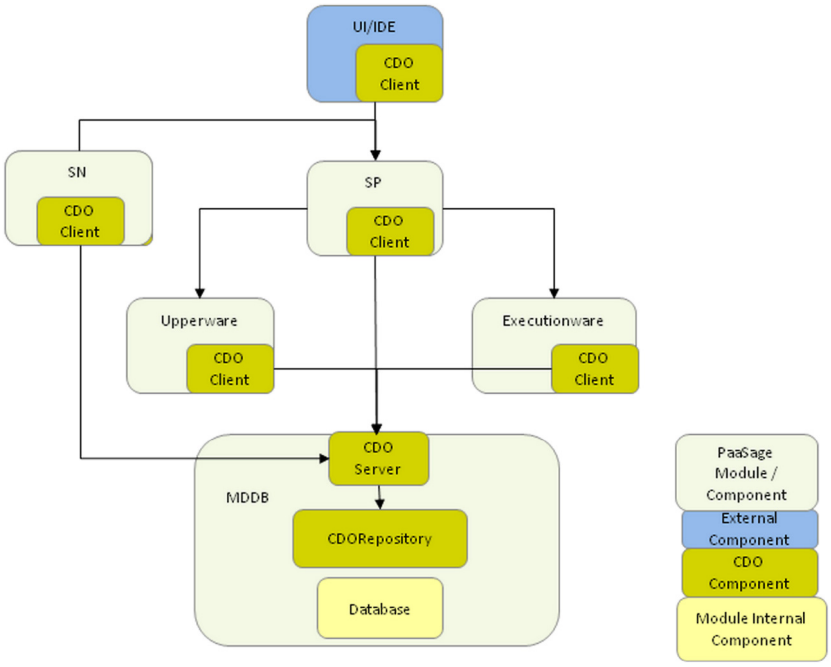

Fig. 1. The PaaSage platform architecture.

REST service which offers platform services enabling users to either run standalone lifecycle tasks or the whole adaptive application provisioning workflow. In this respect, the SP is the orchestrator for the underlying platform modules and components to support this provisioning workflow. This service is not intended to be used directly by users but is suited for programmatic interactions.

The next entry point builds on the SP to enable a more interactive experience with the platform users. This is realised via the Social Network (SN), while other types of external UIs or IDEs can be used for similar purposes or even for providing added-value user support to the user (e.g., billing, analysis or model editing services). The SN, apart from providing usual social network services, it allows users to initiate application deployment workflows, check the status of deployment tasks and browse application execution histories. It also enables the sharing of knowledge in terms of application models, metric specifications, scalability rule models or interesting deployment patterns for applications similar or equivalent to the application at hand. As part of the knowledge sharing experience, the SN allows users to load or export models from MDDB. In this sense, it can enable users to edit models based on, e.g., deployment pattern findings to optimise the service level exhibited by the respective application.

2.2. Security-oriented & adaptation-based meta-models

CAMEL DSL comprises two security-oriented meta-models covering different security aspects. The security meta-model [5] covers security requirements and capabilities specification while the organisation one [4] covers generic organisation details and security-oriented information in terms of roles, users and policies. In the following, we shortly analyse only the security meta-model while the organisation one along with its proposed extensions is analysed in Section 5.2.

2.2.1. Security meta-model

The security meta-model (depicted in Fig. 2 with external SRL concepts coloured in orange and internal concepts coloured in white) aims at covering the specification of different types of security constraints (requirements and capabilities) at different levels of abstraction. On the one hand, it is able to specify higher-level security constraints in terms of security controls. On the other hand, it is able to connect these high-level security constructs to lower level ones which are modelled in terms of constraints on security terms (metrics and properties). Such a linkage enables inspecting those conditions where their satisfaction clarifies whether the high-level security capabilities promised are met.

To allow specifying constraints on security metrics and properties, these two term types are expressed as sub-classes of the respective terms in *SRL* [7]. As such, conditions over these security terms can be specified by exploiting the respective *SRL* constructs. Both security term types are also linked to security domains. This enables grouping or partitioning security terms into different categories. For instance, the *mean-time-between-incidents* (*mtbi*) metric is associated to the domain of *Vulnerability Management*. Security properties can be further classified as *certifiable* when they are measured by security metrics. The sub-classing of security metrics follows the one of the non-functional metrics (as shown in Section 5.1).

3. Solution architecture

3.1. Introduction

The extended architecture for the PaaSage platform prototype relied on satisfying two main goals: (a) enable the security-based adaptation of multi-cloud applications when security incidents occur and (b) secure the access to all platform facility types, i.e., information and service resources. Its primary aim is to support single-site platform deployments based on the rationale that from an exploitation perspective such a platform can be exploited either privately by organisations desiring to move their business into the cloud or by existing but possibly small cloud providers desiring to enhance their single-cloud capabilities.

This architecture relied on certain principles and requirements, detailed now based on the goal concerned.

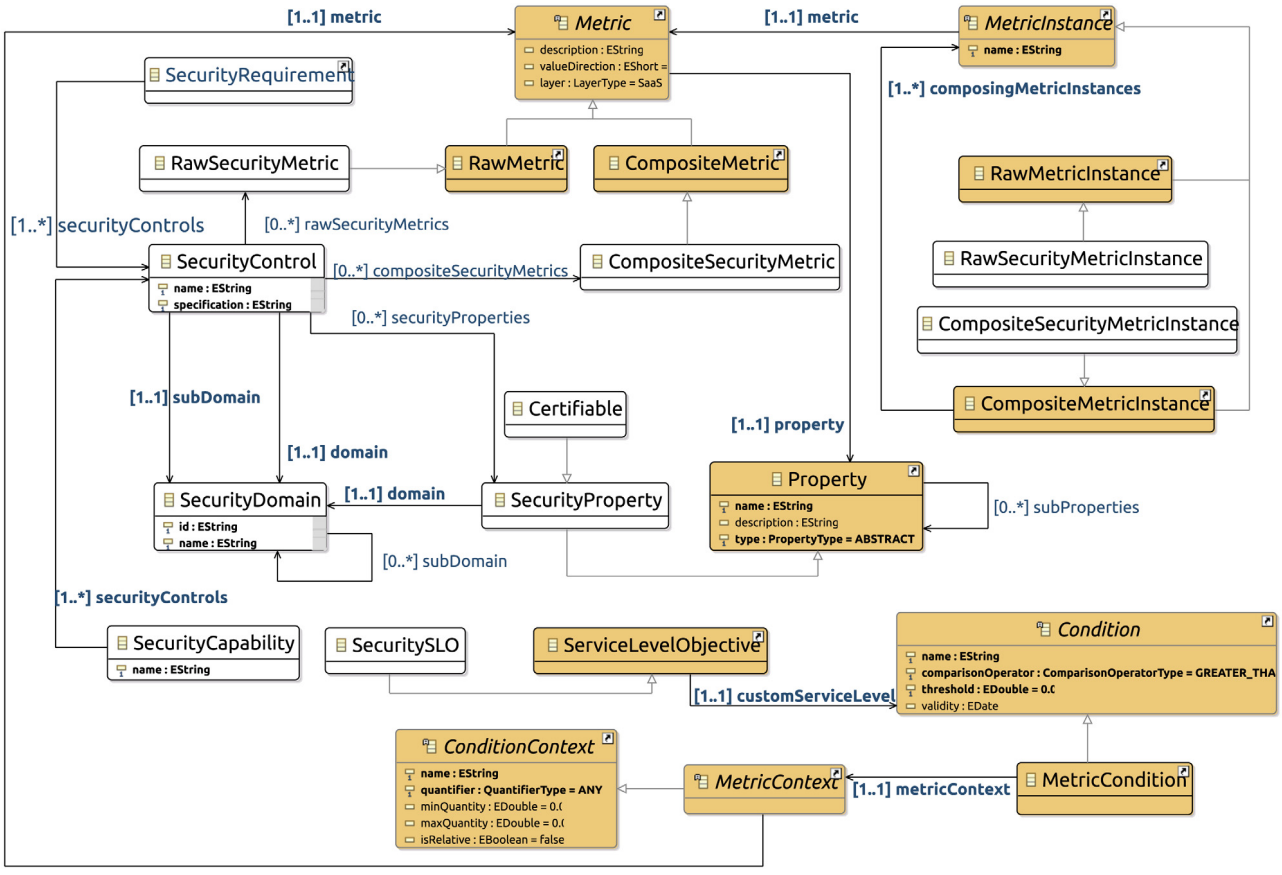

Fig. 2. The CAMEL security meta-model.

3.1.1. Access control requirements

The requirements and principles for controlled access to platform facilities are the following:

- *IC1* — unique credentials per user for single sign-on (SSO) access on all resource types across all platform entry points, components and modules.
- *IC2* — a default permission set must be associated to each basic role of an organisation. An organisation's administrator can then update this set based on organisation needs and policies. This accelerates specifying permissions as a common basic role subset exists across different organisations and can be mapped approximately to the same permission sets.
- *IC3* — an organisation must control the access that other organisations' users can have, indicated from now on as *external users*, over information owned by this organisation. By fulfilling this requirement, the PaaSage prototype will offer multi-tenancy facilities to its users and MDDB will be transformed into a multi-tenant store with private information spaces controlled by the organisations owning them.
- *IC4* — a super administrator is needed to deal with unforeseen security issues by having full write access to the whole MDDB space. The client organisations should trust the PaaSage platform instance operator for this as it represents the most suitable measure to address potential vulnerabilities, especially in case where the access to a customer organisation's resources is completely taken out of its control.
- *IC5* — user identification can be performed via both internal and external identity providers. For external authentication, this creates the need to be able to process and exploit externally certified information so as to properly identify users and map them to the roles assigned to them.
- *IC6* — adopt security standards, such as SAML[3] and XACML[4], such that state-of-the-art software available for them can be exploited to realise the required authentication and authorisation functionality
- *IC7* — re-use existing technology to speed up the security functionality development. At MDDB-side, this translates to exploiting the CDO security feature to secure the access to the CDO repository.

3.1.2. Adaptive security-based provisioning requirements

The requirements to support the adaptive security-based application provisioning are as follows:

- *PR1* — ability to express both high-level and low-level security requirements
- *PR2* — ability to match all types of security requirements and apply them to the deployment reasoning process
- *PR3* — exploitation of state-of-the-art software libraries for security-based monitoring and adaptation
- *PR4* — ability to model security-based rules to drive the adaptive provisioning behaviour of the application
- *PR5* — ability to enforce adaptation actions on the infrastructure on which the application resides.

3.2. Architecture

The solution developed led to extending the current PaaSage prototype architecture by including extra components or enhancing existing ones. The extra components added are mostly related

[3] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security.
[4] https://www.oasis-open.org/committees/xacml/.

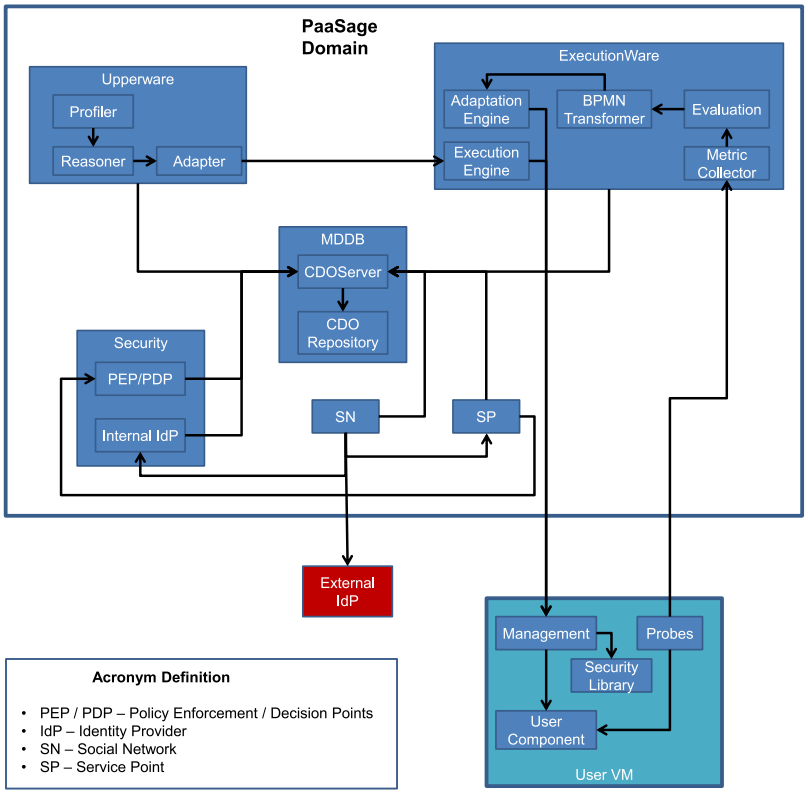

Fig. 3. The overall security solution architecture.

to access control enforcement while the components enhanced are related to both security aspects addressed in this article. Fig. 3 depicts the extended solution architecture. The analysis now continues by explicating the functionality of the added or extended components for each security aspect. The analysis is complemented with the explanation of which respective aforementioned requirement from the previous sub-section is satisfied.

3.2.1. Security-aware provisioning

To explain the adaptations performed for enabling the adaptive provisioning of multi-cloud applications, we need to indicate that there are two architecture levels. The first one maps to the PaaSage platform and is the one controlling in an global way the security-based optimisation while the other maps to the VMs deployed on behalf of the user application at the cloud provider domain and is mainly dedicated to the security-oriented and cloud-specific monitoring and adaptation of these VMs and their hosted components. We also go down into additional details for the *Upperware* and *Executionware* modules, as many of their internal components are influenced or new components have been introduced.

In the PaaSage operator level and starting with the *Upperware* module, three components have been updated. The *Profiler* has been enhanced to process security requirements posed in user models and imprint them in the application profile as well as modify the way the deployment plan problem is constructed by filtering out those cloud providers not conforming to high-level security requirements. The latter maps well to the existing component's functionality as it has to perform a pre-filtering of the provider space to facilitate accelerating the subsequent deployment reasoning process.

The *Reasoner* was enhanced to extend the deployment plan problem by including low-level security requirements to be used for filtering (in case of hard global constraints) the cloud provider space and formulating the optimisation function (for security-based optimisation requirements such as maximise *mtbi*).

The extensions to the *Profiler* and *Reasoner* components actually lead to the fulfilment of the *PR2* requirement as actually high- and low-level requirements are enforced before and during the reasoning process for multi-cloud application deployment, respectively.

The *Adapter* takes care of global application adaptation by considering new possible plans proposed by the *Reasoner* for the current application deployment and selecting one for execution if the benefits for the respective transition are appropriate. In case of security, the *Adapter*'s selection criteria were extended to account for security metrics such that the usual trade-off between performance, cost and security is considered. For instance, the selection criteria could regard that a new deployment is possible as new more secure and less costly cloud provider offerings have been discovered so as to increase the levels of security metrics like *mtbi*. Once a new deployment is selected, the *Adapter* calculates the minimum number of adaptation actions to be performed in order to transit the application to the new deployment state and then sends this action set for execution to the *Executionware* module.

In the case of the *Executionware* module, only the *Execution Engine* has been updated, responsible for managing VM instances and application components, by enforcing the installation of: (a) a security library in each application VM and (b) the respective security probes for security monitoring. As the way security properties and metrics are specified is indifferent to the one for other non-functional terms, the module's *Metrics Collector* and *Evaluation* components were not modified and still server the purpose of collecting metric measurements and evaluating them against any SLO type. To this end, /emphPR3 requirement is partially fulfilled as security-based libraries can actually be exploited for security-based monitoring and adaptation.

The following new components were also added to *Executionware*. The *BPMN Transformer* is responsible for transforming an

adaptation strategy into a BPMN process specification. This strategy will have to be executed in case that a security-oriented adaptation rule is fired. To enable the specification of such rules, SRL, as shown in Section 5.1, was modified to become more generic and incorporate security-oriented adaptation actions (fulfilment of *PR4* requirement). Once the BPMN file is produced, it is relayed to the *Adaptation Engine* for execution.

The *Adaptation Engine* is responsible for executing (security or scaling) adaptation strategy workflows specified in BPMN. Each security adaptation action is executed by calling a *Management* component which runs the respective software library functionality at the VM on which the problematic application component resides. Scaling actions are executed via calling the *Execution Engine*. As such, scaling and security actions can be mixed in an adaptation workflow and we foresee complex application provisioning adaptation scenarios requiring this mixture. For instance, one application VM can be both overloaded and have a security SLO violated. In this case, we could have a composite adaptation rule indicating that the VM will be scaled-out and that both VM instances (old and new) should be enhanced through executing an additional security software to increase the security (protection) level on them. The open-source Activiti Engine[5] was used to realise the *Adaptation Engine*.

At the cloud provider domain, each user VM comprises three specialised components apart from the normal application ones which are: (a) the *Management* component responsible for managing the installation of any other component in the hosting VM. This component was extended to be able to install the security library component and call it in case a specific security functionality must be run; (b) the *security library* component comprising state-of-the-art open-source security adaptation software, focusing on intrusion detection and prevention as well as on encryption, including: (i) fail2ban,[6] (ii) Snort,[7] (iii) OSSEC,[8] (iv) Suricata[9] and (v) AESCrypt[10]; (c) the (monitoring) *probes* that can sense any kind of non-functional metric, either resource-based or user-specific (mapping to higher levels).

Fig. 3 also depicts the interactions involved among the respective components for security-based local/cloud adaptation, where the prefix in the interaction labels indicates the adaptation type involved (i.e.,"L" means local and "G" global). As it can be seen, the security probes sent measurements to the *MetricsCollector* which aggregates them and reports them to the *Evaluation* for local adaptation as well as to the *Reasoner* for global. The *Evaluation* assesses the respective event conditions and checks which adaptation rules are enabled. In case of reaching scalability limits or local adaptation cycles detected, the *Adapter* is informed in order to select the best deployment plan from those derived by the *Reasoner* via the measurements received. Then, global adaptation is performed by calling the *Executionware* with the respective action set to execute.

For each adaptation rule triggered, local adaptation continues with *Evaluation* calling *BPMN Transformer* with the rule adaptation strategy as input and then sending the resulting BPMN file to the *Adaptation Engine* for execution. This engine runs the BPMN file and calls the *Execution Engine* for scaling actions and the *Management* component for managing the respective underlying application or security components (e.g., to start/stop a security component or re-configure an application component).

Based on the above analysis, we can definitely indicate that finally requirements *PR3* and *PR5* are completely fulfilled as the extended PaaSage architecture has the capability to both install state-of-the-art security libraries but also exploit them in a appropriate manner for security-aware multi-cloud application monitoring and adaptation. Previously, in this sub-section, we have explicated how requirements *PR1* and *PR2* are fulfilled, while the next section (Section 5.1) explains how the *PR4* requirement is satisfied. As such, we actually validate that all security-oriented provisioning requirements are fulfilled by the proposed architecture extension and the enhancements to existing CAMEL meta-models/sub-DSLs.

3.2.2. Access control

Access control enforcement mainly regards existing or new components at the PaaSage operator level. The new security components realised interact with the CDO repository to fulfil their security duties as it has decided that this is the sole place where security-information (users, roles and permissions) is stored. Moreover, access control to information sources is mainly realised by exploiting the CDO security feature (see *IC7* requirement).

Our architecture analysis starts with the *CDOClient*, enhanced to obtain the PaaSage platform user credentials in the following two forms: (a) a username and password or (b) a user SAML token (see *IC6* requirement) produced and signed by an *Identity Provider* (*IdP*). Such credentials are exploited to establish an access-controlled session with *CDOServer*, thus securing access to both normal as well as security-oriented information. The first credential form is suitable in programmatic interactions when the respective component desires to immediately access information resources on user behalf. The second form is suitable in web-based interactions where the user first authenticates via *IdP* before attempting to access any kind of PaaSage prototype facility.

The *CDOServer* has also been enhanced in the following ways: (a) it activates the CDO security feature enabling access control session generation and management; (b) the latter feature has been modified and extended to address some particular missing or incomplete functionality and also support token-based user authentication. Section 6.2 highlights what is the missing/incomplete functionality covered.

The communication between the CDO client and server can be additionally secured via using SSL-based sessions in case any security component, like *IdP*, requiring to communicate with the CDO to fetch security-oriented information, is not situated inside the same VM with that hosting the CDO Repository.

To support user authentication and cater for UI-based access in the platform enabling SSO for users, another component has been accounted called *IdP*. We envisage two *IdP* types to be exploited by the platform: (a) an external *IdP* like facebook which authenticates users against its own use base and returns back a token that can assist in user identification such that the user is mapped to those roles assigned to him/her; (b) an internal *IdP* authenticating users against the CDO user base and returning back a token which clearly identifies the user and his/her roles and thus can be immediately used for subsequent user authorisation.

The Policy Enforcement and Policy Decision Points (PEP/PDP) are security components responsible for user authorisation. They take as input the user request and token sent by the SP and communicate to the latter the respective authorisation decision, that is to deny or grant the access to the respective resources designated by the request. The authorisation decision relies on respective permissions granted to the roles assigned to the respective user. If such permissions do allow the accessing of the respective resource according to the corresponding access type, an access grant decision is relayed. As such, these components have

5 activiti.org.
6 www.fail2ban.org.
7 www.snort.org.
8 www.ossec.net.
9 suricata-ids.org.
10 https://www.aescrypt.com/.

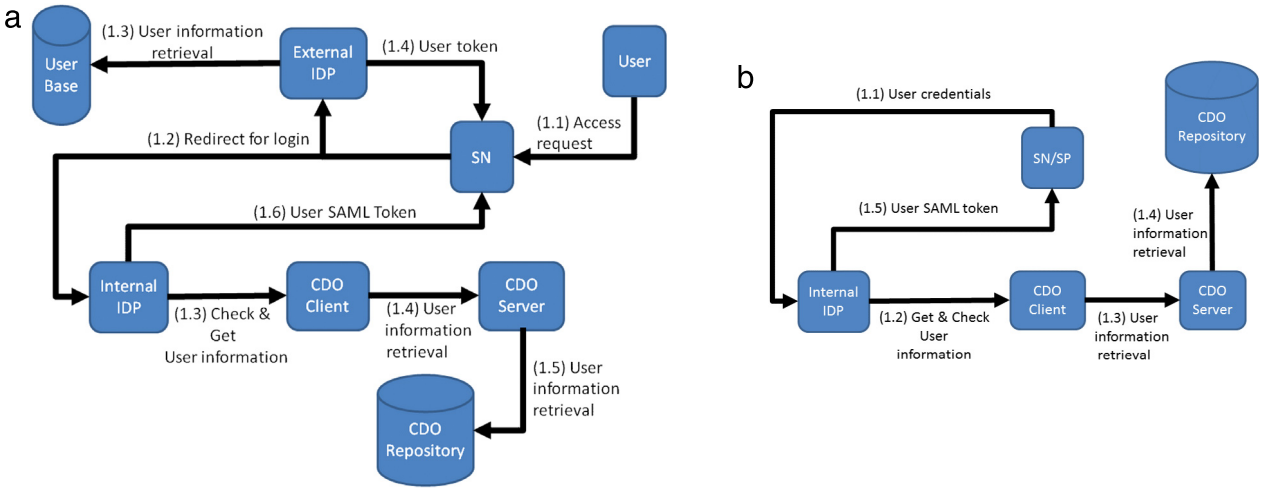

Fig. 4. (a) Web-based authentication interactions. (b) Programmatic authentication interactions.

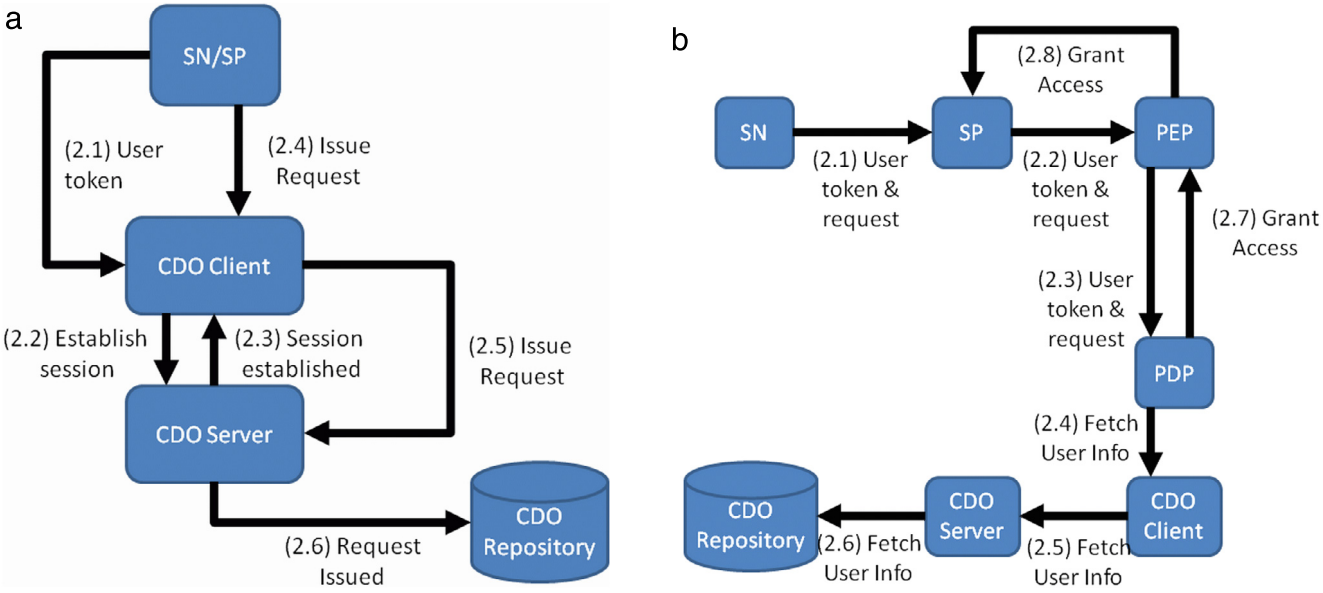

Fig. 5. (a) CDO authorisation interactions. (b) Service authorisation interactions.

access to the underlying CDO repository to obtain the respective permissions granted to the user.

To enable the access control to the platform services, SP was enhanced to perform the following: (a) retrieve user tokens and pass them along with the request to *PEP/PDP*; (b) redirect users to the most suitable *IdP* in case users must be authenticated; (c) initiate access-controlled *CDOSessions* to manage models stored in the *CDOServer* as part of the user request by exploiting a *CDOClient* constructed via the user token.

Finally, similarly to the *SP* case, the SN was also enhanced with the following capabilities: (a) selection of suitable internal or external *IdP* and user re-direction; (b) exploitation of CDOClients constructed via user tokens for accessing CDO models; (c) forwarding user request and token to the *PEP/PDP* for authorisation purposes in cases of service-based resources.

To realise the access control on the platform facilities, the aforementioned components participate in specific interactions catering for different user authentication and authorisation types. Figs. 4(a)–(b) and 5(a)–(b) visualise these interactions for these two processes, respectively. The numbering of interactions in the figures reflects their order where the number prefix indicates the type of security task involved (1 — authentication, 2 — authorisation). To reduce interaction complexity as well as achieve better clarity and comprehensiveness, these figures only show successful interactions and omit some obvious steps, like subsequent service execution in case of a service-access request. Web-based and programmatic authentication are visualised in Fig. 4(a)–(b), while service or model-based authorisation is visualised in Fig. 5(a)–(b), respectively.

As each couple of cases maps to a different security task, we can have different combinations of cases per security task. This leads to catering for 4 main security scenarios, providing an added-value to our solution, which are: (a) web-based authentication & CDO authorisation; (b) web-based authentication & service authorisation; (c) programmatic authentication & CDO authorisation; (d) programmatic authentication & service authorisation. In the following, we analyse the interactions for each security case.

During web-based authentication, the SN redirects the user to the suitable IdP after it is selected. Once user credentials are checked and user authentication succeeds, the respective user token is returned to be exploited for user authorisation. In case of unsuccessful authorisation, the IdP returns an error message which is finally relayed to the user. The same procedure is followed in case of direct access to *SP* services.

Different user checking procedures are involved depending on the *IdP* type. An internal *IdP* authenticates the user by attempting to establish a session with *CDOServer* by using the user credentials. In case the session is established, the user is valid. Then, the *IdP* initiates a more secure session with the *CDOServer* to obtain the extra user information to be asserted which is used to construct the user SAML token.

External *IdP*s perform this checking via their own user bases. Upon successful user authentication, they construct a user token that may not contain the same user information as that stored in CDO. However, we realistically assume that the user email is asserted which can be used to identify the user subsequently.

In case of programmatic authentication, the user program or the SN have direct control over the authentication process. This means that there is no redirection but direct access to the *IdP*'s authentication services. Only the internal *IdP* is exploited in this case as it does support programmatic authentication. Of course, it is in the respective program/SN discretion to exploit also external IdPs if they also support this authentication type. The checking of user credentials follows the same procedure as the one outlined above.

Different types of components can be involved in CDO- or service-based authorisation: (a) a user program; (b) the *SN* when attempting to manage models on user behalf; (c) the *SP* when attempting to access models, e.g., in case of user-initiated deployments. All these components follow the same interaction process for CDO-based authorisation. They first exploit a *CDOClient* to establish a controlled-access session with the *CDOServer* by exploiting the user credentials/SAML token. Upon successful session establishment, the user is authenticated against the *CDOServer* and the respective user request is checked against the permissions attributed to this user in the CDO security model. In case the user is allowed to access the respective model resource, the request is granted and enforced while the respective result is sent back to the user. Otherwise, an error message is relayed back to the initiating component to be displayed to the user.

In case of service-based authorisation, the component sends the user token and request to the *SP* which redirects this information to *PEP*/*PDP*. *PDP* checks the information relayed and especially the authentication token. If the token was produced by the internal *IdP*, it immediately obtains the user roles and fetches the respective permissions for them by securely exploiting the *CDOServer*. In case the token is generated by an external *IdP*, it securely communicates with the *CDOServer* to perform a more complex query which returns the permissions that can be attributed to the respective user identified via his/her email. The fetched user permissions are checked against the user request to reach an authorisation decision (i.e., an access grant or deny) which is communicated to the *PEP* and then to *SP*. Depending on the authorisation outcome, the *SP* either allows the service call or sends an error message to the authorisation initiating component.

Based on the above analysis, we can definitely see that requirements *IC1* and *IC5-7* have been fulfilled. *IC1* is satisfied as now each entry point as well as PaaSage platform component exploits CDO in order to perform user authentication while the *PEP*/*PDP* as well as the basic security mechanisms in CDO are used to perform user authorisation. As CDO relies on the CAMEL organisation model as the single place for storing user credential per organisation, we are now able to enable the exploitation of one credential pair per platform user. More details about how organisation models are managed in the context of the CDO security feature with respect to the proposed access control solution are provided in Section 6. Architecturally, it can be easily seen that we cater for the *IC5* requirement. Moreover, we have also included respective functionality enabling the exploitation of both internal and external identity providers. As we mainly exploit SAML for web-based user authentication and we also include mechanisms for exploiting SAML tokens for user authorisation, *IC6* requirement is also satisfied. Furthermore, as already explained various times in this section, an existing security solution was exploited in terms of enforcing access control in model repositories via enabling the CDO security feature. The way we have been able to exploit this feature is additionally analysed in Section 6 where particular functionality was realised in order to enable this feature's smooth integration in the extended PaaSage architecture. This latter section also explains how the *IC2-3* access control requirements have been fulfilled.

Requirement *IC4* is already fulfilled by including a role in the platform which has full access on the whole MDDB space. The permission set for this role is automatically created on platform initialisation. In this respect, the PaaSage platform operator has just the opportunity to map this role to a respective administration user in order to enable the respective capability to be available in case serious access control incidents occur at the platform.

4. Use case

The extended PaaSage prototype platform, with its architecture analysed in Section 3, is now exploited in order to handle a certain use case so as to highlight its main benefits. The application of the use-case is split into two parts, which are analysed in separate sub-sections, dealing with the two main security aspects related to our main article contributions. For each part, respective visualisations of CAMEL models or parts of them are also supplied. The conformance of these models (parts) to the respective CAMEL meta-models can be checked by examining the content of Sections 2.2.1 and 5.

4.1. Security-aware multi-cloud application deployment & provisioning

This use case concerns a traffic monitoring application executed in certain city areas in order to regulate traffic as well as sustain certain noise and pollution levels. This application consists of 3 main components: (a) a monitoring component able to sense the current noise, pollution levels, traffic size and patterns in the designated area; (b) a traffic analysis (*Anal*) component able to analyse traffic and propose particular traffic reconfiguration plans; (c) a traffic configuration (TC) component able to execute the produced plan. The city areas are charged to the respective municipality (named as *MUNIC_HER*) which has to deploy and run the application in these areas and guarantee that a certain security level is sustained to prevent adversary users from monitoring any traffic configuration decisions or even modifying them to create a traffic chaos. As such, as we are dealing with a hybrid cloud scenario here where the first and last component are deployed in the municipal cloud, the analysis component has to be deployed in a public cloud and be as secure as possible.

Table 1 shows the requirements modelled for *Anal* spanning the deployment, security, cost and performance aspects. These requirements map to the global level as only one component is actually considered and the rest are ignored. The security controls mentioned originate from the Cloud Control Matrix (CCM) [9] of Cloud Security Alliance (CSA)[11] and have been already embraced by the research community and industry. The meaning of each security control is as follows: AAC-02 relates to conducting independent reviews and annual provider assessments, DSI-01 maps to the ability to classify data and services according to various

[11] www.cloudsecurityalliance.org.

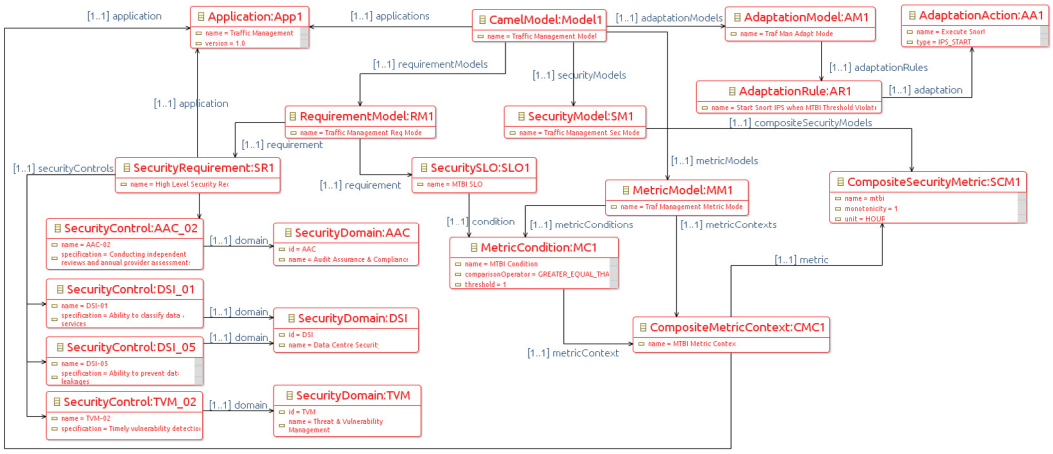

Fig. 6. Snapshot of the user's CAMEL model.

Table 1
Application requirements for deployment, security, cost and performance.

Component requirements	VM	Security controls	Security SLOs	Performance requirements	Cost
Analysis	4 cores	AAC-02, DSI-01,	*mtbi* $\geq$ 1 h	*ex.time* $\leq$ 35 s	$\leq$350$ per month
	4 GBs of memory 40 GBs of hard disk	DSI-05, TVM-02	*avail* $\geq$ 99.99% min(*ex. time*):2.0 max(avail):2.0	min(*cost*): 3.0	

criteria, DSI-05 relates to preventing data leakages, and TVM-02 maps to the timely vulnerability detection. Via this security control selection, the municipality is interested in choosing a cloud provider which is constantly assessed, can prevent data leakages and timely detect security vulnerabilities. Such high-level security requirements are rationale as the selection of a provider should minimise the various vulnerability types which could be involved in the VM hosting *Anal*. The security SLO, concerning the mean-time-between-incidents (*mtbi* metric), posed indicates that security incidents occur as rare as possible. The deployment requirements are associated to increased needs for *Anal* leading to "high" VM requirements. As *Anal* is a critical component, extra requirements map to appropriate performance and availability levels to address unaccepted traffic situations. Depending also on the respective metric monotonicity, different priorities have been given to different requirement types. In particular, cost minimisation is considered the topmost requirement followed by the minimisation of response time and the maximisation of availability that are equally important.

Table 2 depicts the VM and security offerings as well as security capabilities of three cloud providers (based on real information drawn from existing cloud providers), where each offering comes with a particular cost per hour. These offerings and capabilities are inserted into the PaaSage prototype system via the respective cloud provider who uploads a CAMEL model that includes a organisation and a provider (sub-)model. As also explained in Section 5.2, the first sub-model provides generic information about the cloud provider organisation. On the other hand, the provider model describes the actual capabilities offered by the provider via exploiting the feature meta-model [10] in CAMEL.

By comparing Tables 1 and 2, it can be inferred that the third cloud provider, while offering a suitable VM offering, has not realised the DSI-01 security control and violates the security SLO posed. By considering the remaining VM offerings, the *Reasoner* will finally conclude with the selection of VM offering 1 of provider A for deploying the *Anal* component.

The above selection relies both on the offering cost and all optimisation objectives provided as VM offerings affect also the performance level to be exhibited by an application component. In fact, quality metrics, like availability, can be expressed as a linear resource metric combination by following an approach, such as the one proposed in [11]. As such, a certain correlation is introduced to completely understand the benefits of selecting a VM with better characteristics than those indicated by the user baseline VM requirements. In this way, by producing an optimisation function derived from the optimisation requirements posed, VM offering 1 of A has the best utility as it is much cheaper than offering 2 of the same provider and offering 1 of B, although it is not memory optimised, and leads to a performance level similar to the one mapping to the rest of the offerings that could exploited.

The user requirements are specified in a CAMEL model which includes a deployment, a requirement, a metric, a security and an adaptation(sub-)model. A snapshot of the respective CAMEL model, focusing mainly on security and adaptation aspects, can be seen in Fig. 6. The adaptation sub-model comprises of the following three adaptation rules designed to regulate application adaptation behaviour:

$$mtbi < 1 \wedge (size\,(sec\_sw) == 0) \rightarrow \text{execute(Snort)} \quad (1)$$

$$mtbi < 1 \wedge (size\,(sec\_sw) == 1) \rightarrow \text{execute(OSSEC)} \quad (2)$$

$$mtbi < 1 \wedge (size\,(sec\_sw) == 2) \rightarrow \text{migrate(Anal)}. \quad (3)$$

The first rule signifies that when the security SLO posed is violated and no security software runs at the VM hosting *Anal*, the Snort[12] network-based Internet Protection Software (IPS) must be started. The second rule enforces the start of the OSSEC IPS software[13], which has complementary functionality to Snort, when the SLO is still violated and a security software already runs. If the security SLO is still violated with the running of the two IPS software, *Anal* has to be migrated to a new VM.

While the constraint on *mtbi* is about 1 h, its evaluation occurs more frequently to react as quickly as possible to an undesired

12 www.snort.org.
13 www.ossec.net.

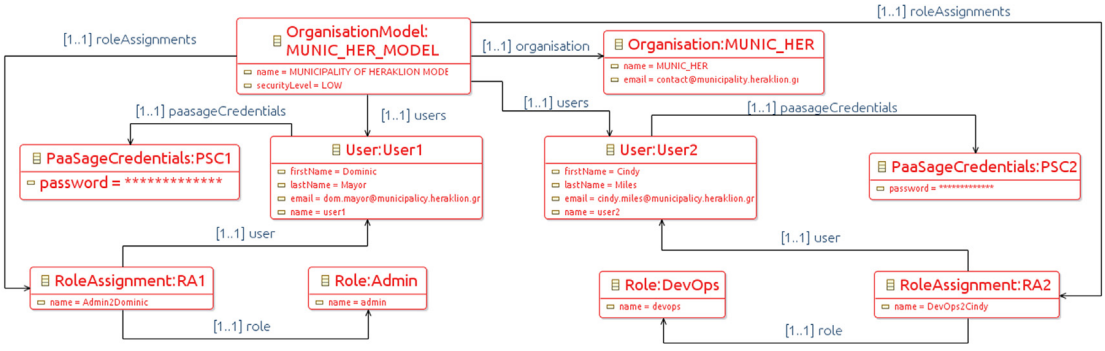

Fig. 7. The CAMEL organisation model of the municipal organisation.

Table 2
VM, security offerings, security capabilities of three cloud providers.

Provider	VM offering	Security controls	Security capabilities	Security offering
A	(1) 4 cores, 7.5 GB, 80 GB → 0.210$ per hour (2) 4 cores, 15 GB, 80 GB → 0.280$ per hour	AAC-02, AAC-03, DSI-01, DSI-05, EKM-03, TVM-02, SEF-05	*mtbi* ≥ 1 h	IPS_A → 0.300$ per hour
B	(1) 4 cores, 4 GB, 130 GB → 0.22$ per hour (2) 4 cores, 4 GB, 40 GB → 0.250$ per hour	AAC-02, AAC-03, DSI-01, DSI-05, EKM-03, TVM-02, SEF-05	*mtbi* ≥ 2 h	IPS_B → 0.250$ per hour
C	(1) 4 cores, 4 GB, 40 GB → 0.1$ per hour	AAC-02, AAC-03, DSI-01, EKM-03, TVM-02,	*mtbi* ≥ 0.9 months	

security situation. Otherwise, it is not rationale to execute one security adaptation action and then wait for 1 h to check whether no vulnerability has occurred.

The designed adaptation resolution behaviour can be checked for correctness by using the PaaSage platform's monitoring dashboard (e.g., through involving a system administrator of the municipality). In case the same security problems occur even after migrating to a new VM, the user requirements must be modified to include using security services as well as utilising *mtbi* in deployment optimisation. In particular, the new requirements involve exploiting an IPS cloud-specific service and requiring that *mtbi* is maximised with a priority equal to 4, i.e., the highest one. Based on this requirement modification, cloud provider B will be selected as it offers a higher security level and a cheaper IPS service. Compared to the first solution, cost is higher, which is quite natural as there is usually a trade-off between security and cost. However, a specific security service is used which guarantees the maximisation of *mtbi*.

Based on the above analysis, it can be understood that PaaSage offers modelling facilities at a high-level of abstraction which can enable application owners/modellers to specify various aspects with respect to a particular application including deployment, requirement, security and adaptation. The resulting models are then exploited by the PaaSage platform in order to derive the most optimal deployment plan, to enforce it as well as re-configure the application when needed. The respective application responsible (e.g., administrator) has then the ability to interfere when problematic situations occur by changing the requirements in order to drive the application reconfiguration in a global manner.

4.2. Access control on models

The municipal organisation (*MUNIC_HER*) that desires to manage the traffic management application via the PaaSage platform has to initially provide another CAMEL model, an organisation one (see Section 5.2), which explicates that two users are mainly involved, assigned to the *devops* and *admin* roles, respectively, while a low security level is required. The platform will exploit this model in order to adapt the security model of the CDO model repository such that the respective policies are derived and enforced. The organisation does not have to provide fine-grained policies for each role exploited as the platform maps each role to a default policy set. In addition, as it will be shown in Section 6.1, the low security level indicates that external users will be able to see all CAMEL models specified by this organisation but not the organisation one as it includes sensitive organisation information. The respective organisation model is shown in Fig. 7.

An organisation model, as indicated in the previous sub-section, needs to be supplied by each cloud provider. Suppose that provider A needs to supply such a model. This model will indicate that the organisation model is partially visible as it contains some generic provider information that can be viewed by all other organisations in the platform but, of course, policy information should not be visible. It will also indicate that the provider model of this organisation can be fully viewed by all platform users enabling them to inspect the respective cloud capabilities offered. Furthermore, an administration user is modelled such that it can enable it to modify both the organisation and provider models, e.g., in case some provider capabilities are modified. In this case, we can actually see a customisation of the access control policies that need to be enforced for this provider organisation. The platform enables the supply of such customised information via enabling the cloud provider to provide specific policies and do not rely on the default ones. The respective organisation model of provider A is shown in Fig. 8.

Suppose, now, that the *devops* user of the municipal organisation attempts to login in the *SN*. The *SN* will redirect the user to

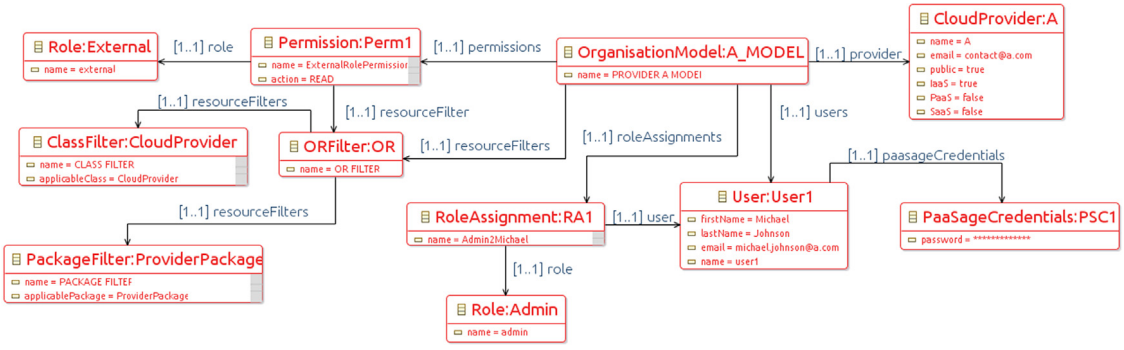

Fig. 8. The CAMEL organisation model of the cloud provider A.

the (internal) *IdP* site where he/she can provide his/her credentials. Upon successful authentication, the SAML token is returned to the *SN*. The user then requires from the *SN* to obtain the respective information and capabilities of the cloud provider A, as this provider and its first offering was selected to host the *Anal* component. This request is internally mapped to establishing a *CDOClient* session by relying on the SAML token and running a query over the models uploaded by provider A to the platform. The first model is the provider one and the *SN* can obtain the whole information for it and display it as this does not conflict the policies supplied by provider A. The second model is the organisation one. *SN* will attempt to obtain all information from this model but the access to any information different from the actual generic organisation description will raise an exception. As such, only this generic information can be displayed for this provider. To this end, we have seen here a use case where web-based authentication and CDO-based authorisation was involved enabling application providing organisations to have access to generic and cloud capabilities information supplied by cloud providers that are in accordance to the access control policies of these providers.

5. Meta-model enhancements

To enable the aforementioned solution to work, two CAMEL meta-models have been extended, namely the scalability rule language and the organisation meta-model. In this section, we analyse each extended meta-model in separate sub-sections. Graphical elements depicting the extended elements over the existing ones in these meta-models are also supplied.

5.1. Scalability rule language — SRL

5.1.1. Original meta-model content

SRL is a DSL focusing on the specification of scalability rules in the form of complex event patterns which when occurring map to specific sets of scaling actions that must be performed. Event patterns are logical or time-based event combination, while events are actually conditions on non-functional terms, like properties (e.g., *availability*) and metrics (e.g., *average response time*). Scaling actions can be horizontal or vertical and cover both possible directions (e.g., scale-up and scale-down).

Event conditions include the respective bound value and the comparison operator (e.g., 20 and *LESS_THAN* for *average response time*). They are also associated to the respective non-functional term and its context. A context explicates the actual object to be measured (e.g., a specific virtual machine (VM)) and the way the evaluation can be performed by determining the measurement schedule and window. It also involves an evaluation pattern signifying whether the conditions on all or some of the instances of the respective object have to be satisfied so that the event can be deemed as triggered. *SRL* specifies the semantics of the non-functional terms considered. Two types of terms can be specified, properties and metrics. *Properties* represent abstract or measurable properties. Abstract properties are at a higher-level and cannot be directly measured by one or more metrics as is the case of measurable properties. However, they can be composed of other more concrete and possibly measurable properties.

A metric explicates all suitable details needed to measure a property, including the measurement formula or sensor and unit. Metrics can be classified as *composite* or *raw*. Composite metrics are derived from simpler metrics via a measurement formula (e.g., *average response time* is measured by taking the *average* over *raw response time* measurements). Raw metrics (e.g., raw CPU usability) can be computed directly from sensors, either attached in the respective object's hostor or being part of the object's code itself.

5.1.2. Meta-model extensions

To address the specification of security-oriented adaptation rules, *SRL* was extended to move from the scaling to a more generic adaptation level (see Fig. 9 with new concepts coloured in grey and old ones in white while orange denotes extended concepts/enumerations). As such, SRL can now specify adaptation rules that can include events on different non-functional term types as well as the performance of actions not necessarily bound to the IaaS level. To this end, the *ScalabilityRule* class was renamed as *AdaptationRule* to signify this change of focus.

The coverage of different action types was complemented with the capability to specify complex adaptation plans [12], not restrained, as in the original version of SRL, in sequential sets of (scaling) actions. In particular, an *Adaptation Task* was introduced, further distinguished into an *AdaptationPlan* and an *AdaptationAction*. An *AdaptationPlan* in turn comprises a control flow construct of a certain type and recursively includes a set of adaptation tasks. This type of modelling enables to construct an adaptation plan tree that can be quite complex and capture sophisticated adaptation cases. In such a tree, we go from quite generic plans to more concrete ones while moving down the tree levels until we research (atomic) adaptation actions.

Each adaptation action is related to a specific type modelled via the *AdaptationType* enumeration which includes scaling, security-based and other actions neglected in the original SRL version, like *VM_MIGRATION*. To also handle erroneous situations in adaptation action performance, an action is related to an *ErrorHandling* which represents an error handling case of a certain type also associated to a specific adaptation action to be performed to alleviate this error. Any kind of action is also related to the component instance on which it should be applied. This completes all information required by the component responsible to execute the respective adaptation action. For instance, the *Execution Engine* will need to know which VM instance to scale out for an adaptation action

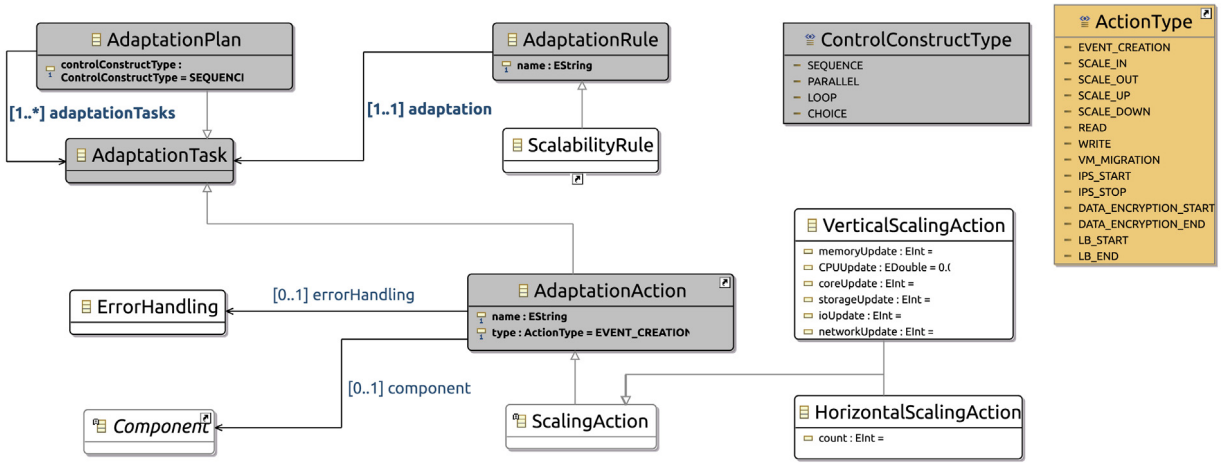

Fig. 9. The CAMEL's SRL meta-model.

as this will provide the knowledge about which VM offering to instantiate.

Based on the aforementioned extension, requirement *PR4* is now fulfilled as security-oriented adaptation rules can be expressed in SRL in conjunction to its previous ability to express scalability-oriented adaptation rules. In fact, this extension now enables SRL to specify adaptation rules even at higher levels of abstraction, provided that the respective adaptation actions are covered by the *ActionType* enumeration.

5.2. Organisation meta-model

5.2.1. Original meta-model content

The organisation meta-model covers the specification of various organisation-specific information to be exploited by the PaaSage platform for performing various tasks, including authentication, authorisation and cloud deployment. This meta-model relies on the CERIF standard [13], used for modelling research organisations, users and roles. It also includes other important security-oriented information, including security policies as well as platform and cloud credentials. To conform and further support CERIF, specific transformation code was developed able to map CERIF models to CAMEL organisation ones. As such, organisation models are not required to be developed from scratch but can rely on existing CERIF specifications, thus reducing the modelling effort of an organisation's respective administrator.

Fig. 10 depicts the organisation meta-model UML diagram where again new concepts are coloured in grey and old ones in white. For organisations, basic information is captured in the form of a name, email and web address. For cloud provider organisations, further organisation information is specified by referencing the respective cloud provider model and describing a provider's own infrastructure in terms of data centres. The latter information can be used to correlate cloud offerings to data centres and determine certain specificities pertaining to such a correlation in terms of different prices and service levels or security capabilities. By representing cloud providers as organisations, we also enable their users to participate in the PaaSage platform, thus allowing the dynamic updating of provider models, when new cloud services or existing ones are modified, as well as the exploitation of the PaaSage platform facilities to adaptively provision applications in their cloud infrastructure.

An organisation model enables specifying roles and users. Roles represent particular organisation entity types associated to a set of security policies specified in the form of access control permissions (see below). As users can simultaneously map to different entity types, they can be associated to many roles, thus being able to inherit the permissions specified for these roles. Role-to-user mappings are specified via role assignments which include important information, such as the assignment start and end date. By accounting such information, a security system can invalidate role assignments when they expire. It can also be used to detect abnormal behaviour not prevented by the security system, thus enabling to remedy it to the possible extent.

The modelling of users includes information pertaining to their own (last/first name or email) and PaaSage platform identification (in the form of a user name). Two different credential types can be associated to users: (a) a password enabling their access to the PaaSage platform facilities; (b) cloud credentials utilised by the PaaSage platform to perform deployment tasks on these users behalf in a certain cloud.

5.2.2. Meta-model extensions

The initial organisation meta-model version included the modelling of simplistic organisation policies in the following triple form ⟨role, action, resource_type⟩, where resource_type indicates the resource type concerned (e.g., service). This simplistic modelling was not enough based on the requirements in Section 3.1.1 as it was quite restrictive and did not enable us to express different permissions for different role types. As such, the policy modelling was enhanced to become richer according to the following two directions targeting the two main platform resource types, i.e., services and models:

- the policies with respect to service resources target both SOAP- and REST-based services and can be posed either on a specific URL or a URL pattern. The latter can be quite beneficial in case that we desire to express that all types of methods on a REST-resource are covered by the same policy.
- the policies with respect to models can be specified in three orthogonal ways: (a) by indicating a path in the CDO repository mapping either to a resource directory or a certain resource. In the first case, the modeller can also specify whether all directory content is recursively concerned; (b) by indicating a specific meta-model class for which all instances will apply to the respective permission (e.g., all instances of *VM*). If such instances are part of a more complete model, not all model content can be accessed unless the respective classes are included in the user permissions; (c) by indicating a certain package for which the permission will hold for all instances of this package classes.

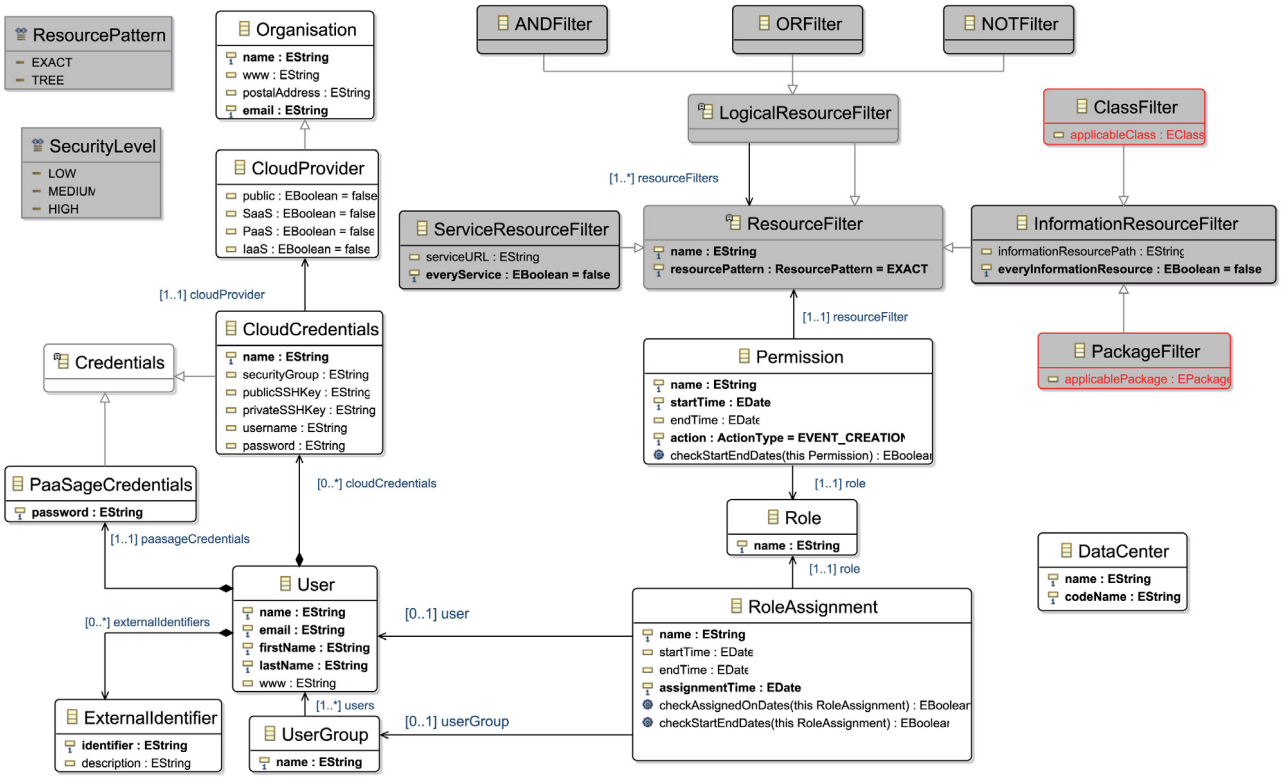

Fig. 10. The CAMEL's organisation meta-model.

To enable the logical combinations of permissions, we have separated the actual permission (see *Permission* class) from its filters (i.e., resource expressions — see *Filter* class). As such, we can specify a resource permission including filters even of a different type to cater for more sophisticated cases. For instance, for a certain role, such as the *devops*, we could indicate that it can have access writes to the organisation package and to a certain resource directory. Such a permission will signify that this role will have access to those directory resources which include instances of the organisation package classes. Permissions are connected to filters via the *filter* property while the logical filter combination is enabled via introducing specialised filter classes with logical operator semantics (e.g., *AND* filter).

To impose clear bounds on permission processing responsibilies, our modelling does not allow the mixture of model and service policies. Information resource policies are amenable for processing by the *CDOServer* respective security component while the service policies by the *PEP/PDP* components.

It should be noted, before closing this section, that the information resource policies have been modelled in such a way that they are compatible with the respective policies that can be specified in CDO. This has greatly facilitated the CDO security feature integration in the extended PaaSage architecture and enabled the respective support for the satisfaction of requirement *IC7* (see Section 3.1.1). Moreover, indirect support to requirements *IC2-3* is also facilitated through enabling the editing of information and service resource policies at a suitable level.

6. Access control solution

Apart from realising the security components functionality as outlined in Section 3, there were additional details to be addressed to guarantee that model-based access control is properly handled. Such details included integrating organisation and the CDO security models, extending the CDO security feature and developing an administration API which enables the management of security-oriented information on behalf of organisation administrators by also guaranteeing the aforementioned integration. These three aspects are now analysed in the following three subsections. Please note that the first two aspects enabled us to satisfy the *IC7* requirement.

6.1. Organisation-to-CDO-security-model mapping

Our access control solution, as indicated in Section 3.2.2, relies on the CDO security feature[14] which maps to creating and maintaining a CDO security model that prescribes the users, roles and policies that govern the access to the models stored in a CDO repository. As we require using an organisation's security model as part of its organisation model, we are facing integration challenges which can be addressed by two exclusive ways: (a) the organisation model encompasses the CDO security one; (b) both models are independent but their information is integrated.

The first way indicates that the organisation meta-model must be modified to include the CDO security one in such way that also service-oriented policies can be specified and the extra information for common concepts in the former meta-model (e.g., cloud credentials for users) is maintained. This represents a hard integration task leading to additional modelling effort and the modification of the respective CDO code related to the modification of the CDO security meta-model. As such, this type does not seem to be suitable.

The second way maps to a lighter integration performed mainly at the model level through model transformations. This integration type was adopted as it is better than the previous one for the following reasons: (a) organisation administrators do not have direct access to the sole CDO security model; (b) there is no need

14 https://wiki.eclipse.org/CDO/Security_Manager.

to modify the CDO code; (c) an organisation administrator does not have to deal with any kind of unnecessary complexity that is inherent in the modelling via the CDO security meta-model.

The lightweight information integration has relied on initially graphically mapping the two meta-models together and then realising the respective transformation code via the designated mappings. Fig. 11 depicts the mapping between the CDO security and the organisation meta-models. As it can be seen, both meta-models contain equivalent concepts which greatly facilitates the integration task. The sole differentiation lies: (a) on the additional permission types on service access captured by the organisation meta-model; (b) the richer specification of users and role assignments in this meta-model; (c) the richness of the CDO meta-model in specifying model access permissions which is not needed in its full extent for the purposes of our security solution. Finally, we should note that the mapping of a CAMEL organisation to a CDO user group was enforced to establish a way to refer to all users of an organisation.

Before entering details about the mapping procedure realised, we must explicate important details about particular constructs needed/re-used to specify permissions: (a) the types of actions allowed in permissions and (b) a basic set of organisation roles. By considering the resource types to be addressed, three main action types are allowed: (a) *read* and *write* for model permissions and (b) *access* for service permissions.

The basic set of organisation roles maps to the three main user type targets of the PaaSage project:

1. *admin*: This is an organisation administrator that can just update the organisation model of its organisation. The rest of the (organisation) roles do not have any kind of access to this model.
2. *business*: this is a business-oriented user type that can have a complete view on what is happening in its organisation in terms of the PaaSage platform (e.g., check current deployments cost) as well pose high-level security and cost requirements to drive the organisation applications' deployment.
3. *devops*: this is the core target use type of the PaaSage platform with the duty to specify applications and all types of (CAMEL) models apart from organisational ones.

Our security solution automatically maps each basic role to a default permission set. As such, permission specification is accelerated as organisations must modify or extend this set only when needed. In fact, our mapping can capture the most usual cases in permission specification and only organisation-specific exceptions must be addressed by a certain organisation via, e.g., adding new roles and permissions on them. As such, requirement *IC2* (see Section 3.1.1) is obviously fulfilled.

Our solution also enables sharing organisation-specific information by introducing the *external* role. This role maps to member users of other organisations for which some of the current organisation's resources can be shared. In this way, any organisation, if desires to share its model resources, can just specify permissions which associate the *external* role to these resources along with the type of access needed.

We actually foresee three main alternative cases involved in sharing of models for an organisation: (a) some models are shared (e.g., only deployment models), (b) all models are shared apart from organisation ones and (c) no model is shared. To speed up permission specification in the second case and reduce the modelling effort of an organisation administrator, the security level attribute was included with levels mapping to these three cases. So, once an organisation indicates that it will have a low security level (second case), our solution automatically creates those permissions which give access to most the organisation's model resources to external users.

The above actually leads to satisfying requirement *IC3* (see Section 3.1.1) as the user has the possibility to specify the desired security level and have the platform automatically creating the respective permission set for external users. If such an organisation needs to intervene to this mapping, it can then just modify the respective permission set automatically generated.

The introduction of the *external* role and the enabling of permission customisation has lead to enforcing a slightly different way of dealing with role modelling. In particular, it is imperative that roles become specific to each organisation such that we can then customise the permissions to be allowed on them. As it will be shown later on, this modelling pattern makes the integration process slightly more complex. However, this is a small penalty that we pay for enabling an increased flexibility in permission specification. The main feature of automatic basic-role-to-permission mapping is still preserved as there will still be specific basic roles for each organisation via which this mapping can be performed. Even if an administrator does not specify such basic roles, our solution automatically adds them to the respective organisation model, as this is needed to construct the respective automatic permissions for these roles for the low security level.

6.1.1. Mapping procedure

The mapping procedure starts with the pre-processing of the organisation model to enrich it with the different permission sets mapping to basic roles, in cases such permissions have not already been specified in this model. This pre-processing also incorporates a set of permissions for the external role (which is also generated) in case the low security level has been selected.

Next, the main transformation logic is performed by loading in main memory the organisation model and performing a set of transformations on certain object types to enforce the respective mappings in Fig. 11 onto the sole CDO security model. The transformation set involved is now analysed as follows:

- an organisation is mapped to a CDO user group named after its name which can then be associated to specific permissions like those mapped to external roles. In particular, our code associates: (a) all external roles of previous organisations to the new organisation user group and (b) the external role of the new organisation to the previously created organisation user groups.
- each user is mapped to a CDO user with the same PaaSage platform credentias where the user name maps to the CDO user *ID* and the password to the CDO *UserPassword*. We do not follow a full information transformation here to minimise the repetition degree and thus only transform information essential for performing access control on model resources via the CDO. The CDO user generated is also associated to the user group of the current organisation created during the previous step.
- each role is mapped to a CDO *Role*, where the CDO Role's *ID* is generated from the name of the CAMEL role concatenated with the name of its respective organisation according to the following pattern: "<camel_role_name>_<organisation_name>". If a role is external, then it is associated to all user groups previously defined and stored apart from the current one.
- Each model-based permission is mapped to a CDO model permission by first setting the access attribute value mapping to the CAMEL permission's action type. Then, a straightforward filter mapping (see also Fig. 11) is enforced by just copying the respective information as we focus only on a subset of all possible identical filters covered by the CDO security meta-model. Finally, the CDO permission constructed is associated to a CDO role discovered by performing a query based on the role naming scheme indicated in the previous bullet by considering the role and respective organisation name.
- For each role assignment, we first identify the respective user and role in the CDO model and then we map the CDO user to the CDO role. No other information is copied.

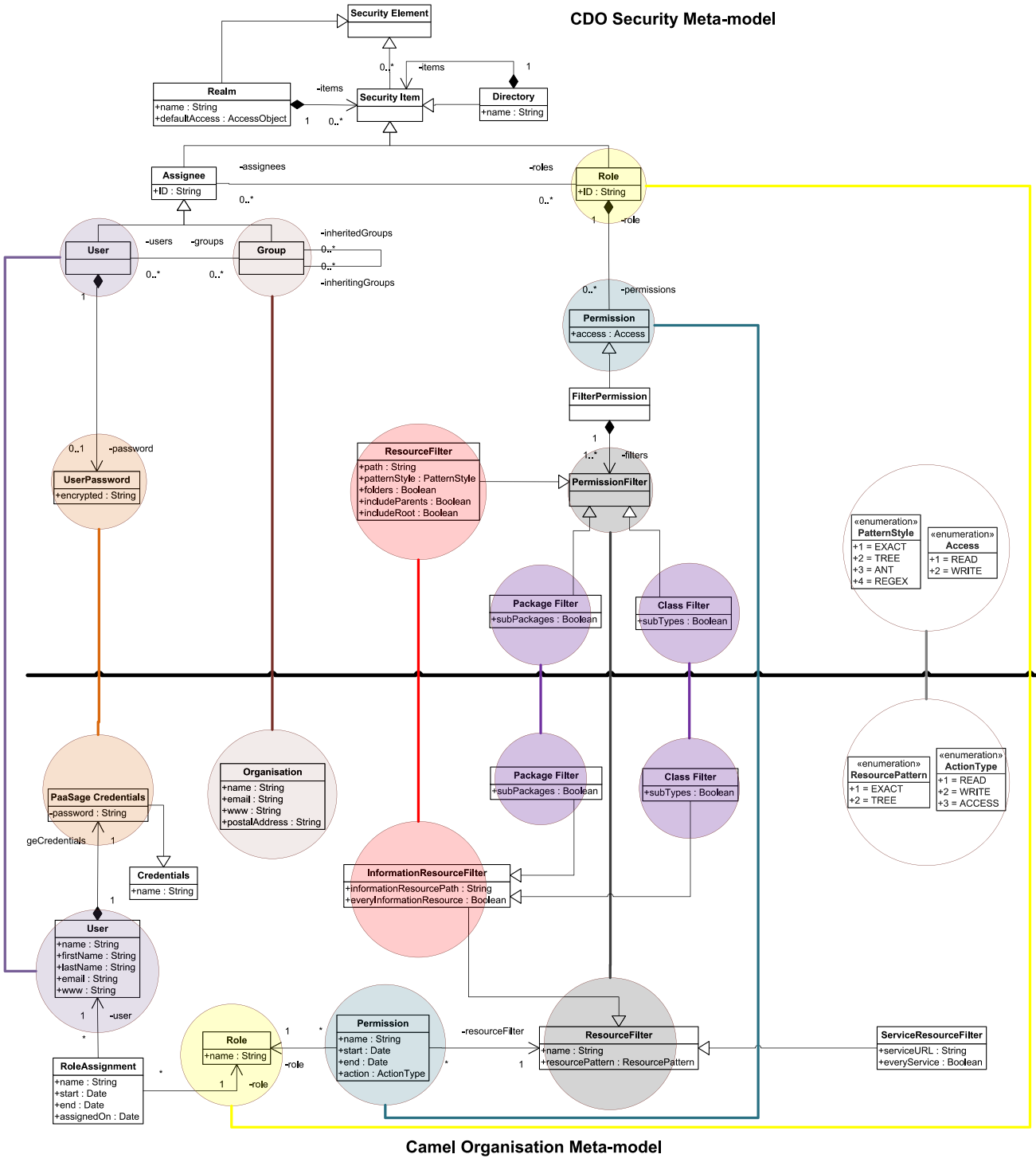

Fig. 11. Camel organisation to CDO security meta-model mapping.

6.1.2. Mapping algorithm application on the use case

In order to enable a better understanding of the mapping algorithm, we rely on the second part of the use case (see Section 4.2) in order to explain the mapping of the application and cloud provider organisation models to the respective cdo security model.

Suppose that the municipality's organisation model (see Fig. 7) is firstly processed, which includes two users mapping to two different roles, respectively. The mapping algorithm will first create a user group called "munic_her". It will then generate three roles, the two basic and the external one, which will be named as "admin_munic_her" "devops_munic_her" and "external_munic_her". The latter role is not associated to any organisation CDO user group as this is the first organisation model being processed. The organisation's permissions are next to process for which the respective CDO permissions are generated. In the end, each organisation user is mapped to a CDO user, is included in the organisation's CDO user group and is mapped to one of the three basic but organisation-specific CDO roles by enforcing the respective role assignments.

Almost the same process is followed for the organisation model (see Fig. 8) of cloud provider A. However, additional work is

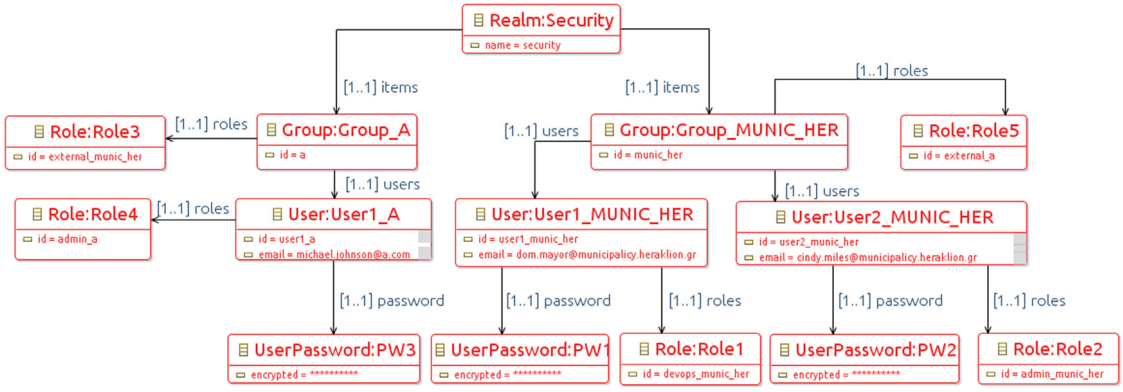

Fig. 12. The CDO security model resulting from the two organisation models of the use case.

performed mapping the provider's CDO user group to the external role of the municipal organisation ("external_munic_her") and the latter organisation's CDO user group to the external role of the cloud provider ("external_a"). As such, the processing of new organisation models apart from the first one always leads to additional work which enforces the low security level requirements posed by certain organisations of the PaaSage platform, including the current one. The respective CDO security model mapping to the processing of the two CAMEL organisation models is depicted in Fig. 12, where policy information has been removed due to the overall complexity that they would introduce to the figure as well as to their rather straightforward CAMEL-to-CDO mapping.

6.2. CDO security feature extensions

While the aforementioned integration procedure more or less guarantees the controlled access to CDO models, particular issues had to be overcome to enable our solution to function properly and correctly. The first issue is related to the CDO security feature's non-consideration of restrictions on the permission and role assignment lifetime. This issue was addressed by enhancing our solution to include a small thread component that constantly checks the permission and role assignment end dates and, in case that they are expired, removes these elements from both the organisation and CDO security model. The checking frequency was chosen in a scale that will not lead to undesired access-control situations. In particular, it was derived that a frequency of 5 min was enough based on the feedback obtained from the PaaSage use case partners.

The second issue concerned the way some filters function in the CDO security feature. In particular, it seems that the *ClassFilter* and the *PackageFilter* filters (see also Section 5.2) work only classes and packages of generic ecore models and not specific ones, like in the case of CAMEL. To solve this issue, CDO security feature code was modified to map the two filter types for any specific permission to our own class implementations that enable the use of concrete meta-model constructs.

Finally, the third issue concerns the way authentication is performed. In particular, once users are logged in the PaaSage platform, they are associated with a certain SAML token to be exploited for SSO and authorisation purposes. In case of controlled model access, once the token is received, it must be exploited to initiate *CDOSessions*. However, the CDO security feature currently works with simple credentials in the form of a username and password. To deal with this issue, the CDO security feature code was again modified. This time the liberty on the way to modify code was quite restrictive based on the current implementation version as many critical classes could only operate only on simple credentials. The solution we followed relied on the principle of minimal code change. In particular, we still relied on issuing of simple credentials during session establishment with the sole exception that the user name maps to the special value of "SAML_TOKEN" and the password to the actual token to be inspected. As such, we only modified two specific classes in the CDO code at the server side: (a) the one which performs the actual user authentication with this credentials type; (b) the one that exploits authentication to establish the session with the *CDOClient* such that the actual username is bound to this session. The former class was extended to deal with our special case of token-based authentication by also returning the actual name of the user authenticated. Such an authentication is performed by validating first the token considered and then checking whether the asserted information maps to an existing user in the CDO security model. Depending on which type of *IdP* has generated the token, the assertion can map to the username (internal *IdP*) or the user email (external *IdP*). In either case, a simple query over the CDO security model can validate the user existence, where in the case of the user email-based search the username is also retrieved.

The validation of the token depends on the *IdP* type involved and the security information available. If the *IdP* has included public security information (see SAML's *KeyInfo* element) on the two main parts in the SAML token on which *Signature* elements are placed, i.e., the whole token or the assertions included, this public key is used to validate the respective signature. In case of an internal *IdP*, if such public key information is missing, we rely on a public key of a certificate already exchanged with this *IdP*. An assertion might also be encrypted in the case of internal IdP-generated tokens depending on the security level imposed on our system. In this case, it is decrypted via the private key of the exchanged certificate.

6.3. Administration API

While the mapping functionality along with the CDO extensions enable the proper functioning of model-based access control, we need to ensure that this functionality is applied in terms of our solution. In addition, we desire to develop a specific wrapper over this functionality to assist organisation administrators to manage their organisation models. This gave rise to the Administration API exhibiting the next main features:

- encapsulates organisation-to-CDO-security-model transformation functionality thus ensuring that both models will be updated and well integrated to each other
- provides the core administration functionality to manage the organisation model and its security-oriented parts with methods related to user, role, role assignment and permission updating. A side-effect of this functionality is that models are lively updated in terms of a particular CDO repository

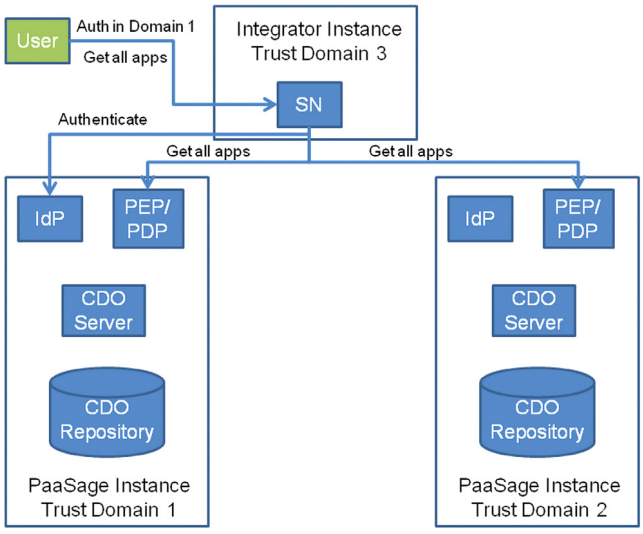

Fig. 13. The PaaSage multi-platform architecture.

- does not impose the administrators to learn the specifics of an organisation meta-model
- can be used to create administration UIs further facilitating the work of organisation administrators.

The transformation code exploited by the administration API is suitable for only inserting a whole organisation model in the CDO repository. As such, this code must be enhanced to handle the addition and updating of organisation model parts. This enhancement involved realising functionality that properly identifies a CDO security model object from its organisation model counterpart and relies on the naming schemes identified in the transformation code analysis. This API covers the updating of both model and service-resource based policies. However, only for the former the updates need to be propagated to the CDO security model.

7. Multi-PaaSage-platform model sharing

While our previous solution can work only in the case of single PaaSage platform instances, we envisage that possibly in the near future, different platform instances can exist. To this end and depending on the requirements and policies of platform instance operators, it might be the case that model sharing needs to be supported across all instances. In this case, we envisage an evolution of the proposed security solution architecture to cater for such situations relying on the extended architecture depicted in Fig. 13.

In this architecture, the *SN* is considered as the integrator of the PaaSage platform instances with respect to the organisation spaces contained constituting a single entry point for all of them. In this sense, all users can access the resources in any instance via this *SN*. The sole exception with the architecture of Section 3, is that we have a single *SN* instance and not multiple, each pertaining to one platform instance. As such, this centralised *SN* is aware of and can draw content from all platform instances for authenticated users. Any user can authenticate via the *SN* which redirects him/her to the *IdP* of the instance on which his/her organisation belongs. After successful authentication, a SAML token is generated and associated to the user session, to be exploited for fetching the information that this user can access in all CDO repositories in the instances involved. Users can access an external organisation's resources only if the permissions on the external role in the respective organisation model allow them to do so. Thus, user requests are propagated to all instances in case of generic access patterns (e.g., fetch the description of all publicly shared applications) or specific instances for concrete access patterns (e.g., read a specific model). In any access pattern case, the *PEP/PDP* and the *CDOServer* (as another type of authorisation entity) of the respective instance should trust the *IdP* authenticating the user to validate the token and issue the respective authorisation. Thus, another requirement for this architecture to function is that a trust chain is created between the different instances.

This architecture is simple and enough to suit the purposes of a distributed PaaSage platform. It also imposes a limited set of requirements on the respective platform instances. As such, it will be easy to evolve the single-instance architecture to a multi-instance one. The sole guarantee for the success of such an evolution is trust. If each organisation trusts its instance operator and all operators trusts each other, then true resource sharing can be really achieved to further promote the adoption and use of Cloud technology.

8. Related work

The related work is split into three main parts: (a) meta-models for adaptation/scalability rules, (b) meta-models for security and especially access control and (c) finally multi-tenancy in cloud platforms, which are mapped to respective sub-sections. The last sub-section explains how the state-of-the-art is advanced via the respective article contributions and how it could influence the further evolution of the PaaSage platform prototype.

8.1. Adaptation/scalability rule meta-models

Before starting the analysis, we should indicate that most languages focus mainly on supporting the specification of scaling rules and do not consider security aspects. To this end, as this article proposes an extension to SRL, in the sequel, we focus on comparing SRL over these languages. We should also highlight that a specific vocabulary of adaptation patterns for cloud-based applications has been proposed in [14]. Such a vocabulary could be exploited in order to use a kind of templating mechanism via which adaptation rules could be formed.

Most existing scalability rule languages are simplistic by being able to express scalability rules as mappings between conditions over fixed metric and single horizontal scaling actions that target single cloud environments. Some of these languages have been developed in the context of European projects, such as Reservoir[15] [15] and Optimis[16] [16]. Compared to these languages, SRL is richer by enabling to specify both logical and time-based operators over conditions on metrics, which can be precisely described thus being not fixed, and map them to not one but multiple scaling actions that can span multiple clouds.

A reactive and proactive approach to cloud elasticity is proposed by Moore [17], involving a language able to define simple scalability rules. Compared to SRL, this language reaches a medium level of metric expressiveness by being able to specify almost sufficient metric details but relies on bad design choices, such as the mixing of scaling policies with rules.

Copil et al. have proposed the SYBL scalability rule language [18] that attains a good level of expressiveness. This language is able to express logical combinations of constraints on metric values and map them to the triggering of specific strategies encompassing particular scalability actions, such as scale-in and scale-out. Compared to SRL, SYBL has a moderate level of expressiveness as

15 http://www.reservoir-fp7.eu/.
16 http://www.optimis-project.eu/.

it is not able to specify time-based combinations of events/metric conditions while it relies on a fixed vocabulary of metrics, thus being not able to define them. Finally, SYBL is not integrated with any other language as in the case of SRL which is closely integrated with the CloudML language [19] in order to associate metric conditions on the application components of a deployment model.

The CloudFormation language[17] of Amazon is a good example of a simplistic scalability rule language which has a low level of expressiveness with respect to SRL. This language defines policies over when to trigger horizontal scaling actions on virtual machine instances (on which application components have been loaded). Such policies specify conditions over metrics that mainly concern the resources used and not the component or application level.

8.2. Security & access control meta-models

8.2.1. Access control meta-models

Our access control sub-meta-model of the CAMEL organisation meta-model has been inspired by role-based access control work as well as the security meta-model in CDO. The related work in access control modelling mainly focuses on defining access control meta-models on user (application) data that are moved and processed in the cloud, explicating the respective rules governing the access to such data. From such meta-models, we should highlight those that are ontology-based [20,21] as they allow performing more sophisticated types of inferencing over the respective policies defined allowing at runtime to take access control decisions as well as validate those decisions that have been already taken. Nevertheless, the focus of this work is on securing the models developed by organisations which mainly concern the modelling of the organisation applications. As such, such models do not constitute (running) user application data. However, as a PaaSage platform could run in the cloud, then there is obviously a connection as models in this case can be considered as data that are actually stored and managed in the cloud. In the former case of actual application data, each data item could map to its own policy or set of policies. In the latter case, we actually raise the level of granularity by explicating particular patterns of models (e.g., all deployment models) for which different types of users can have different types of access. However, even in this latter case, our approach can still enable the specification of policies that focus on a specific resource. As such, our approach can define policies at different levels of abstraction. Moreover, our approach does not only consider information but also computerised/service resources [22]. On the other hand, our meta-model is less sophisticated than other access control meta-models with respect to the specification of the actual policy (e.g., other approaches use Lineal Time Logic and other formalisms and take into consideration additional information) as the focus is on capturing the most common access control scenarios and not advanced ones.

8.2.2. Security meta-models

The CAMEL security meta-model has been partially inspired by the work conducted by CSA, the EU CUMULUS European project [23] and C-SIG [24]. In particular, CSA has proposed a specific security control framework mapping to the CCM [9] which can be used to evaluate the security risk involved in resorting to a cloud provider based on the feedback obtained from cloud vendors and customers. The structure of CCM has been totally reflected in the security meta-model while the respective content can be exploited towards defining high-level security requirements and capabilities in the form of security controls. CCM has also the goal to associate these security controls to concrete and quantifiable metrics. Towards achieving this goal, CSA's Metrics Work Group (CSA Metrics WG) has defined 10 metrics which cover approximately 25 security controls. However, as this initial metric list seems to be quite limited, we have relied on extending it by considering the work conducted in the EU CUMULUS project [23], in the context of a particular extensive security model, as well as the recent recommendation of C-SIG [24], endorsed by the European commission, over the context of cloud-based Service Level Agreements (SLAs). By combining the inputs from CCM, CUMULUS and C-SIG and exploiting a particular method, we have produced in our previous work [5] a rich security model able to connect high-level constructs like security controls over lower-level ones like security properties and metrics.

A particular comprehensive and extensive meta-model [25] has been derived out of the work in the EU CUMULUS European project, able to model security properties. While, apart from security property modelling, this meta-model covers also the characterisation of security requirements (e.g., threat, attacks or assumptions) and their type (e.g., application, certification), it does not exhibit constructs for modelling security metrics. Furthermore, it does not link security properties to security controls.

Fenz and Ekelhart [26] have proposed a security ontology covering important security concepts, such as security controls, threats, vulnerabilities, and security properties. While this ontology is simple, it covers more security-related concepts than our security meta-model, thus explicating ways in which our meta-model can be extended. However, the structure in this meta-model is flat as no specific characteristics or properties are specified for each security concept, while security metrics are not covered at all.

The PoSecCo EU project [27] has developed a set of ontologies, including access control, application, network, and virtualization ontologies, aiming at describing many aspects of a service-based system apart from security. Then, through inferencing and the knowledge of IT resource to security capabilities mappings, security capabilities for a pattern of software components can be derived. The work in this project is also able to produce the set of all possible security configuration alternatives for a certain IT policy. Compared to our security meta-model, the level of richness of the respective ontology in PoSecCo is low.

Almorsy et al. [28] have proposed a security meta-model which covers concepts, such as security controls, but does not capture security properties and metrics. This meta-model is part of a model-driven security engineering approach which decouples security requirements are from the application itself in such a way that multiple security requirements posed by different tenants can be enforced on the same application instance.

Related research work conducted at NIST [29] and CIS [30] has resulted in the proposal of particular metric templates which can be used for providing all suitable information for realising security metrics. However, it should also be highlighted that these templates do not cover the specification of higher level constructs, such as security controls, and their linkage to the metrics specified, while of course they do not cover the notion of security requirements and capabilities.

8.3. Multi-tenant cloud platforms

We can distinguish between two multi-tenancy types at the platform and application level. At the platform level, multi-tenancy maps to the ability to manage multiple customers and offer to them a private space. Such a private space can contain information related to the usage of the platform services by the customer

[17] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html.

but may also contain sensitive customer data. On the other hand, multi-tenancy at the application level maps to the capability to serve many clients within one application instance. Many sophisticated research proposals have been put in place [31,32] for the second multi-tenancy type, while existing proprietary cloud frameworks support it, like the Namespace API in Google App Engine[18] or the SaaS Multi-Tenant Management Framework of TechCello[19]. In this paper, we focus mainly on the first multi-tenancy type along with respective issues concerned with secured access to platform services and the following analysis is based on this.

Most proprietary platforms can support platform multi-tenancy as they must handle business customers which pay for the services used and expect a certain privacy and security level. Major players like Amazon offering platforms such as AWS provide secure interfaces via which clients can access the respective services offered. However, this platform type does not consider information sharing as each organisation information is isolated and restricted to the use for that organisation. No models can also be used to describe deployment, performance or organisation models and no mechanisms exist for model manipulation. As such, users should be developers or administrators with the ability to either use a certain UI or REST API.

Most research proposals focus on providing a security layer on top of one or multiple clouds. Leandro et al. [33] propose a multi-tenant authorisation system with federated identity for cloud environments based on Shibboleth which can provide minimal user information to the *SP* as well as ensure mutual protection of both service providers and customers. An interesting but theoretical approach is proposed by Albeshri and Caelli [34] where reverse access control is exploited for customers to control the way authentication and authorisation is performed by cloud providers while cloud providers guarantee that the customers do not violate the overall security policies of the cloud structure/hierarchy. Alcaraz Calero et al. [35] propose a multi-tenant federated authorisation framework and architecture for cloud services that adopts hierarchical role-based access control and path-based object hierarchies. Bohli et al. [36] propose four different multi-cloud architectures types, categorises the respective schemes employed and discusses their main benefits with respect to security aspects. Finally, Li et al. [37] propose a multi-cloud security solution operating in multiple levels by providing three main services: (a) a VM security service, (b) a virtual network security service and (c) a policy-based trust management service. The latter service relies on techniques and mechanisms which enable access control over federated resource pools as well as support trust federation via optimising task privacy by also considering cost requirements.

8.4. Discussion

In contrast to the currently proposed research and proprietary solutions and platforms, the PaaSage project offers a secure platform catering for different interaction types. It allows organisations to share their knowledge by also explicating the security level imposed on their private information spaces. It also allows exploiting the platform facilities in a web-based or programmatic way. Furthermore, it enables users to specify and manipulate models via which knowledge can be more easily derived and shared.

The combination of all the above features certainly provides an added-value to the PaaSage platform with respect to the competition, especially as this platform also facilitates knowledge and expertise sharing in the cloud to promote the optimisation of application deployments in contrast to the private usage of state-of-the-art cloud platforms. PaaSage also enables multi-cloud application security-oriented deployment and provisioning, something not offered by current platform offerings which mainly cover single or hybrid cloud scenarios and seldomly take into consideration any kind of security requirements. This is enabled by taking into account all possible security aspects. In particular, both high- and low-level security requirements are considered to further optimise cloud application deployments via filtering the cloud provider space based on these requirements as well as considering security metrics in optimisation formulas along with their respective priorities. Adaptation in most cloud platforms is offered via simple scaling actions which attempt to scale in or out application components according to simple events or their logical combinations. On the contrary, this article proposes the use of generic adaptation rules, which are more elaborate, by being able to specify in a rich manner both event combinations as well as composite adaptation plans that involve more than one adaptation actions, and can cater for covering both scaling and security-oriented application adaptation. Furthermore, the capability to security-aware adaptation is completed via the use of state-of-the-art software libraries for monitoring the security level of applications as well as enforcing particular security actions on them.

The PaaSage platform can benefit from the state-of-the-art. It is apparent that it has to evolve to secure not only the access to its data and services but also user applications which are deployed and run on multi-cloud environments by enhancing them with security components and services. It also needs to enable organisations to develop multi-tenant applications. As such, it would be interesting to couple the existing platform services with additional features aiming at realising the missing aspects by exploiting state-of-the-art techniques and services in data isolation, infrastructure protection and multi-tenant application development. In addition, it would be interesting to inspect how well the theoretical approach in [34] could be applied on the PaaSage security architecture to allow a more flexible type of security enforcement based on the customer organisation requirements and the PaaSage platform operators' policies.

9. Conclusions

This paper presented a particular approach towards securing the access to a multi-cloud platform's services and information and enabling security-aware application provisioning. This approach relied on: (1) extending the existing model-based platform architecture of the PaaSage prototype enabling it to exhibit user access control and security-aware application provisioning capabilities as well as handle different security scenarios and (2) extending two DSLs in the CAMEL [4] DSL family, which includes DSLs covering different domains related to cloud computing, to capture the additional information required to support these two capabilities related to the specification of access control policies on platform information resources and services and of security-aware adaptation rules. The different security scenarios cover the support for both web-based and programmatic user authentication and the exploitation of both internal and external identity providers.

The proposed approach also enables transforming a model-based repository into a multi-tenant store on which access to organisation-specific information is restricted. This transformation involves exploiting existing functionality in terms of securing the CDO model-based repository which realises the PaaSage MetaData-DataBase module, mapping each from a set of certain basic roles to a specific default set of access control policies and

[18] https://cloud.google.com/appengine/docs/java/multitenancy/index.
[19] http://www.techcello.com/product/overview.

controlling the level of access of external to the organisation users to the organisation's own information resources. The automatic mapping of basic or external roles to a default set of access control policies reduces the modelling effort of the organisation by also enabling to intervene at an appropriate level in case this policy set requires any adjustment. To further facilitate managing security-oriented information, a particular administration API is offered enabling the creation and updating of users, roles and permissions. Such an API can be integrated in existing organisation administration tools or can be used to construct such tools which could also reduce the learning curve by not imposing administrators to learn any kind of language.

Through combining this approach with previous work on security focusing on managing security requirements and capabilities [5] and considering such information in deployment planning to filter the cloud provider space [6], the PaaSage platform completes the support to all security aspects and is thus differentiated with respect to the main competition. Finally, in case multiple instances of the PaaSage platform become available, this article proposes an extension to the proposed architecture by also explicating the respective updates to the functionality of respective security-oriented components in order to allow the controlled sharing of models across all platform instances. This will further promote the sharing of knowledge with respect to the usage of cloud over possibly different communities thus enhancing the role that the Social Network in the PaaSage platform can take as the central hub enabling this knowledge sharing.

The following work directions are planned: (a) validating our approach according to particular use cases in PaaSage; (b) performing advanced testing to prove that our solution does not exhibit critical security issues; (d) supporting other security standards, like XACML; (e) coupling administration API with a UI vanishing the requirement of good knowledge of CAMEL organisation meta-model and EMF-based modelling expertise for administrators; (f) supporting secure federations of multiple PaaSage platform instances realising the architecture in Section 7.

Acknowledgement

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreement number 317715 (PaaSage).

References

[1] J. Opara-Martins, R. Sahandi, F. Tian, Critical analysis of vendor lock-in and its impact on cloud computing migration: A business perspective, J. Cloud Comput. (ISSN: 2192-113X) 5 (1) (2016) 54:1–54:18.
[2] R. Sahandi, A. Alkhalil, J. Opara-Martins, Cloud computing from SMEs perspective: A survey based investigation, J. Inf. Technol. Manag. XXIV (1) (2013) 1–12.
[3] L. Schubert, J. Domaschka, P. Guisset, PaaSage - making cloud usage easy, in: CloudScape, 2016. URL http://www.cloudwatchhub.eu/paasage-making-cloud-usage-easy.
[4] A. Rossini, K. Kritikos, N. Nikolov, J. Domaschka, F. Griesinger, D. Seybold, D. Romero, D2.1.3 – CloudML Implementation Documentation (Final version), PaaSage project deliverable, 2015.
[5] K. Kritikos, P. Massonet, An integrated meta-model for cloud application security modelling, in: CF, Elsevier, Madrid, Spain, 2016.
[6] K. Kritikos, D. Plexousakis, Multi-cloud application design through cloud service composition, in: CLOUD, IEEE, New, NY, USA, 2015, pp. 686–693.
[7] K. Kritikos, J. Domaschka, A. Rossini, SRL: A scalability rule language for multi-cloud environments, in: CloudCom, IEEE, 2014, pp. 1–9.
[8] A. Rossini, Cloud application modelling and execution language (CAMEL) and the PaaSageWorkflow, in: ESOCC – EU Projects Track, Springer, Taormina, Italy, 2015.
[9] Cloud Security Alliance, Cloud Control Matrix, Tech. Rep., 2011, URL https://cloudsecurityalliance.org/research/ccm/.
[10] C. Quinton, D. Romero, L. Duchien, Cardinality-based feature models with constraints: a pragmatic approach, in: T. Kishi, S. Jarzabek, S. Gnesi (Eds.), SPLC 2013: 17th International Software Product Line Conference, ACM, 2013, pp. 162–166.
[11] P. Xiong, C. Pu, X. Zhu, R. Griffith, vPerfGuard: An automated model-driven framework for application performance diagnosis in consolidated cloud environments, in: ICPE, ACM, Prague, Czech Republic, 2013, pp. 271–282.
[12] R. Oberhauser, Towards dynamic business process management: Adapting processes via cloud-based adaptation processes, in: BMSD, Springer, Milan, Italy, 2015, pp. 1–22.
[13] K. Jeffery, N. Houssos, B. Jörg, A. Asserson, Research information management: the CERIF approach, IJMSO 9 (1) (2014) 5–14.
[14] A. Ahmad, M.A. Babar, Towards a pattern language for self-adaptation of cloud-based architectures, in: WICSA, ACM, 2014.
[15] F. Galán, L.M. Vaquero, S. Clayman, G. Toffetti, D. Henriksson, Deliverable D4.1, D4.2 and D4.3 – Scientific Report, Reservoir project deliverable, 2009.
[16] A. Rumpl, H. Rasheed, O. Waeldrich, W. Ziegler, Service Manifest: Scientific Report, Optimis project deliverable, 2010.
[17] L.R. Moore, K. Bean, T. Ellahi, A coordinated reactive and predictive approach to cloud elasticity, in: Cloud Computing, IARIA, 2013.
[18] G. Copil, D. Moldovan, H.L. Truong, S. Dustdar, SYBL: An extensible language for controlling elasticity in cloud applications, in: CCGrid, IEEE Computer Society, 2013, pp. 112–119.
[19] N. Ferry, A. Rossini, F. Chauvel, B. Morin, A. Solberg, Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems, in: L. O'Conner (Ed.), Proceedings of CLOUD 2013: 6th IEEE International Conference on Cloud Computing, IEEE Computer Society, ISBN: 978-0-7695-5028-2, 2013, pp. 887–894.
[20] L. Hu, S. Ying, X. Jia, K. Zhao, Towards an approach of semantic access control for cloud computing, in: CloudCom, Springer-Verlag, Beijing, China, ISBN: 978-3-642-10664-4, 2009, pp. 145–156.
[21] H. Takabi, J.B. Joshi, Semantic-based policy management for cloud computing environments, Int. J. Cloud Comput. 1 (2/3) (2012) 119–144.
[22] P. Goyal, R. Mikkilineni, Policy-based event-driven services-oriented architecture for cloud services operation & management, in: CLOUD, IEEE, 2009, pp. 135–138.
[23] A. Pannetrat, D2.1: Security-aware SLA specification language and cloud security dependency model, Cumulus Project Deliverable, 2013.
[24] Cloud Select Industry Group (C-SIG), Cloud Service Level Agreement Standardization Guidelines, Technical Report, 2014, URL http://ec.europa.eu/information_society/newsroom/cf/dae/document.cfm?action=display&doc_id=6138.
[25] A.M. Gómez, M.A. Fernández, R. Harjani, A. Muñoz, An engineering process to address security challenges in cloud computing, in: Proc. of the Third ASE International Conference on Cyber Security, ASE, 2014.
[26] S. Fenz, A. Ekelhart, Formalizing information security knowledge, in: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security, ASIACCS'09, ACM, New York, NY, USA, ISBN: 978-1-60558-394-5, 2009, pp. 183–194.
[27] C. Basile, J. Silvestro, A. Lioy, D. Canavese, M.A. Neri, S. Paraboschi, M. Verdicchio, M. Casalino, T. Scholte, D3.2: Security Ontology Definition, PoSecCo Project Deliverable, 2011.
[28] M. Almorsy, J. Grundy, A. Ibrahim, Adaptable, model-driven security engineering for SaaS cloud-based applications, Autom. Softw. Eng. (ISSN: 0928-8910) 21 (2) (2014) 187–224.
[29] E. Chew, M. Swanson, K. Stine, N. Bartol, A. Brown, W. Robinson, Performance measurement guide for information security, Technical Report, National Institute of Standards and Technology, USA, 2008.
[30] The Center for Internet Security, The CIS security metrics v.1.10, Technical Report 28, USA, 2010. URL https://buildsecurityin.us-cert.gov/sites/default/files/CIS_Security_Metrics_v1.0.0.pdf.
[31] H. Cai, N. Wang, M.J. Zhou, A transparent approach of enabling SaaS multi-tenancy in the cloud, in: Services, IEEE, 2010, pp. 40–47.
[32] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, P. Fremantle, Multi-tenant SOA middleware for cloud computing, in: Cloud, IEEE, 2010, pp. 458–465.
[33] M.A.P. Leandro, T.J. Nascimento, D.R. dos Santos, C.M. Westphall, C.B. Westphall, Multi-tenancy authorization system with federated identity for cloud-based environments using shibboleth, in: ICN, IARIA, 2012, pp. 88–93.
[34] A.A. Albeshri, W. Caelli, Mutual protection in a cloud computing environment, in: HPCC, IEEE, 2010, pp. 641–646.
[35] J.M.A. Calero, N. Edwards, J. Kirschnick, L. Wilcock, M. Wray, Toward a multi-tenancy authorization system for cloud services, IEEE Secur. Priv. 8 (6) (2010) 48–55.
[36] J.-M. Bohli, N. Gruschka, M. Jensen, L.L. Iacono, N. Marnau, Security and privacy-enhancing multicloud architectures, IEEE Trans. Dependable Secure Comput. 10 (4) (2013) 212–224.
[37] J. Li, B. Li, T. Wo, C. Hu, J. Huai, L. Liu, K. Lam, CyberGuarder: A virtualization security assurance architecture for green cloud computing, Future Gener. Comput. Syst. (ISSN: 0167-739X) 28 (2) (2012) 379–390. URL http://www.sciencedirect.com/science/article/pii/S0167739X1100063X.

Kyriakos Kritikos obtained his B.Sc., M.Sc. and Ph.D. in Computer Science from University of Crete. He is currently a Post-Doc Researcher at FORTH. He was a ERCIM Post-Doc researcher at CNR and CHT. He was also a Post-Doc researcher at POLIMI. His research interests span the following areas: Business service design, interactive service-based application design, quality-based service life-cycle management activities, Ontology modelling and reasoning; Constraint and Mathematical Programming; Distributed (Information) Systems; Cloud Computing.

Tom Kirkham is a Research Fellow at STFC, England. Tom is currently involved in the PaaSage project while he has previously worked on the OPTIMIS, Akogrimo, SOCRADES and TAS3 project. Tom finished his Ph.D. in 2007 at Bangor University with a thesis focused on the development and exploitation of Dynamic Virtual Organisations in Grids. This work was done part time on the back of employment integrating ERP systems and then research at STFC. Previous to that he had completed a BA in History at Notts Trent and a part time M.Sc. in Geographical Information Systems at Leicester University. His research interests include cloud computing, data privacy, trust, risk, business integration, mobility, employability and embedded web services.

Bartoaz Kryza has obtained his Ph.D. from the AGH University of Science and Technology in Krakow. He has been actively involved in the PaaSage European project. His research interests include Cloud Computing, Distributed Computing, Parallel Computing, Knowledge Management, the Semantic Web, High-Performance Computing and Parallel Programming.

Philippe Massonet is scientific coordinator at CETIC, a Belgian ICT applied research center. His research interest cover the areas of software, service and security engineering, as well as distributed systems such as Grids and service oriented infrastructures. He was recently coordinator of the GridTrust IST European project (Strep) dealing with trust and security in next generation grids, and is responsible for dissemination and security in the RESERVOIR IST European project (Integrated project) led by IBM research. He is currently the coordinator of the PONTE eHealth project that is looking into using semantic web technologies to build decision support systems for the design clinical trials. He is active in the NESSI ETP working groups, and the ERCIM working groups. At CETIC he also works as a consultant for industry and government in his areas of research. He has experience in the prototyping and commercial development of advanced requirements engineering tools. Previously he was manager of the requirements engineering team at CETIC. He has experience in the management of international R&D projects (Eurescom MESSAGE, IST FP6 GridTrust, and FP7 PONTE) and is or has been involved in several IST FP6/FP7 research projects (RESERVOIR, HPC4U, AssessGrid, CoreGrid, GridTrust and Oldes).