PaaSage

Model Based Cloud Platform Upperware

Deliverable D4.1.2

Product Database and Social Network System

Version: 1.0

D4.1.2

Name, title and organisation of the scientific representative of the project's coordinator:
Mr Philippe Rohou – GEIE ERCIM – Project coordinator
Tel: +33 4 9715 5306 - Fax: +33 4 9238 7822
E-mail: Philippe.rohou@ercim.eu

Project website address: http:/paasage.eu/

Project	
Grant Agreement number	317715
Project acronym:	PaaSage
Project title:	Model Based Cloud Platform Upperware
Funding Scheme:	Integrated Project
Date of latest version of Annex I against which the assessment will be made:	03 July 2014
Document	
Period covered:	
Deliverable number:	D4.1.2
Deliverable title	Product Database and Social Network System
Contractual Date of Delivery:	30/09/2015 (M36)
Actual Date of Delivery:	02/10/2015
Editor (s):	Kyriakos Kritikos
Author (s):	Tom Kirkham, Kyriakos Kritikos, Bartosz Kryza, Kostas Magoutis, Philippe Massonet, Christos Papoulas, Maria Korozi, Asterios Leonidis, Stavroula Ntoa, Craig Sheridan, Alisdair Innes, Douglas A. Imrie
Reviewer (s):	Stéphane Mouton (CETIC) and Tom Kirkham (STFC)
Participant(s):	
Work package no.:	4
Work package title:	Communications Hub
Work package leader:	Kostas Magoutis, FORTH
Distribution:	PU
Version/Revision:	1.0
Draft/Final:	Final
Total number of pages (including cover):	117

DISCLAIMER

This document contains description of the PaaSage project work and findings.

The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated in the creation and publication of this document hold any responsibility for actions that might occur as a result of using its content.

This publication has been produced with the assistance of the European Union. The content of this publication is the sole responsibility of the PaaSage consortium and can in no way be taken to reflect the views of the European Union.

The European Union is established in accordance with the Treaty on European Union (Maastricht). There are currently 27 Member States of the Union. It is based on the European Communities and the member states cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice and the Court of Auditors. (http://europa.eu.int/)

PaaSage is a project funded in part by the European Union.

Executive summary

MDDB is considered as a backbone of the PaaSage platform providing significant facilities in terms of model management and inter-component communication, thus enabling the support to the model-driven architecture that PaaSage follows. While MDDB was designed according to a particular database technology in mind, this design has been modified based on a decision to alter the technology focus to better support model orientation. Apart from this important architectural modification, MDDB was enriched with additional facilities which provide added-value support to the rest of the PaaSage modules and components while it has also adopted a specific security solution which can be promoted to the whole platform. The Social Network has also been evolved towards perfectly fitting its design and realising all its intended functionality. Significant re-engineering efforts have been put on the Knowledge Base, partly due to the change of technology focus, which have led to significant performance improvements.

All the above developments have been also thoroughly evaluated to assess whether the respective component performance is adequate and satisfactory. The results produced are not only more than encouraging but also highlight some particular configuration of components that can lead to more optimised performance.

Based on the above analysis, the goal of this document is to report the great work that has been conducted in WP 4 during the last 18 months period since M19. It analyzes the new architecture of MDDB, it explicates the functionality of the main components involved and describes as well as presents the results of thorough evaluation experiments. In the end, it provides a short roadmap which reflects the amount of work that will be conducted in the remaining period of the project spanning the last fourth year.

By considering the roadmap as well as the level of completion of the tasks of the WP4 work plan, we can finally deduce that the work conducted in this WP was so significant and plethoric that it has also influenced or is going to influence the characteristics of the whole PaaSage platform. In particular, the PaaSage platform can benefit from the security solution [4] developed in the context of this WP in order to exhibit a security feature that can enable it to be discriminated with other similar or equivalent offerings. In addition, it can also benefit from one of the most powerful component of the MDDB prototype, which is the Knowledge Base in order to design rules which can further enhance the functionality as well as improve the functionality of some components, like the Reasoner.

Intended Audience

This is a public document intended for readers with expertise in cloud computing and model-driven engineering. The reader should refer to the previous version of this document D4.1.1 [1] as it conveys important knowledge which might be needed for the comprehension of some parts of this deliverable. The reader should also refer to the description of the overall PaaSage architecture presented in Deliverable D1.6.1 [2]. D1.6.1 provides background on the overall PaaSage architecture and the way different modules fit in it, as well as the internal architecture of particular PaaSage components. In D4.1.2, the reader is introduced to the functionality implemented in the M36 prototype of the MDDB and the social network infrastructure as well as to the next steps required to complete a small part of the remaining functionality planned to be conducted during the fourth year of the project.

Contents

1 Introduction

MDDB is considered as a backbone of the PaaSage platform providing significant facilities in terms of model management and inter-component communication, thus enabling the support to the model-driven architecture that PaaSage follows. While MDDB was designed according to a particular technology in mind, this design has been modified based on decision on a change of technology focus to better support model orientation. Apart from this important architectural modification, MDDB was enriched with additional facilities which provide added-value support to the rest of the PaaSage modules and components while it has also adopted a specific security solution which can be promoted to the whole platform.

To this end, the goal of this document is to summarize the goals that have been achieved in Work Package (WP) 4, to analyze its new architecture as well as explicate the technology that has been developed and integrated in the PaaSage prototype. This underlines the work that has been conducted in the past 18 months since the publication of the previous version of the deliverable D4.1.1 [1].

The rest of this deliverable is structured as follows. Section 2 focuses on shortly analyzing of the new architecture by giving an overview of the main components involved and their relationships.

Section 3 focuses on explicating the functionality of the components constituting the core functionality of MDDB in terms of model management. This section also shortly describes the CAMEL meta-models that have been developed in the context of this WP as well as particular model storage guidelines which have been developed to guarantee an error-free storage of the models manipulated by the PaaSage platform.

Section 4 concentrates on the security solution [4] that has been developed by providing architectural details about the components involved and the different ways main security tasks can be performed. It also explicates some information integration efforts that have been performed in order to guarantee the proper exploitation of the underlying technology enabling the realisation of the security features envisaged. Finally, it shortly analyzes a particular administration API that has been developed in order to assist administrators in the management of users and respective organisation security policies.

The re-engineering efforts towards improving the performance of the KB and making suitable to the model-based technology that has been adopted are analyzed in Section 5. The new architecture of the KB is also explicated while the updated domain model is presented. Moreover, technical and implementation details are provided.

Section 6 highlights the work conducted towards delivering a fully-functional Social Network (SN) which constitutes one of the user entry points in the PaaSage platform. It starts by explicating the UI design process and practices followed. Then, it unveils the main functionality supported by the SN. Finally, it shortly analyzes the architecture of the SN and highlights some important technical implementation details.

The description of thorough experiments and the respective results on MDDB, KB and SN is provided in Section 7. The results provided are quite encouraging and indicate that a suitable technology was utilized to realize this main parts of the MDDB architecture. These results also unveil interesting directions with respect to particular configurations of the technology used in order to reach optimized performance.

The work planned for the fourth and final year of the project in terms of WP4 is highlighted in Section 8. Finally, Section 9 concludes the document by shortly summarizing its main content.

2 Architecture

Due to the use of the CDO technology as well as the development of a security solution which spans not only the MDDB but the whole PaaSage platform, the architecture of the MDDB has been updated and simplified. It now includes all the required new components and those not needed any more have been removed. This architecture is depicted in Figure 1.

As it can be seen, MDDB comprises 4 main components:

a. the MDDB core which is the storage and model manipulation layer of the architecture,
b. the Knowledge Base which is the knowledge layer providing added-value information to interested partners and components,
c. the Social Network which is the usage layer and
d. the security components IDP and PEP/PDP which constitute the security layer along with CDO components not actually visualized in this figure.

Thus, through this layered architecture, where security is a cross-cutting concern, the whole intended functionality of MDDB is achieved and its main goals are satisfied.

Let us now focus a little bit more on each main component, while a more elaborate analysis including the internal architecture of each component is provided in later sections of this deliverable (Sections 3-6). The MDDB core is as its name witnesses the main component/module of the MDDB architecture which offers two types of functionalities: (a) enables the manipulation (updating, querying, storage, exporting, transformation) of models through EMF and CDO technology and (b) guarantees the persistence of models through the use of the CDO technology into the underlying DB. As such, MDDB can be really the store and manipulation enabler for the meta-data information pertaining to the whole lifecycle of multi-cloud applications as well as an indirect communication medium between the various PaaSage components.

The Knowledge Base is the component which is responsible for deriving added-value knowledge from the basic information stored in MDDB. The production of such knowledge is computed through the chained firing of rules. The knowledge that is produced through this knowledge base spans quite interesting and important information which involves the derivation of application and component similarity as well as the production of deployment hints in the form of best application deployments. However, through the incorporation of new rules, possibly through the involvement of the users of the Social Network (SN), additional knowledge can be derived which can take other forms, such as suggested adaptation/scaling actions for particular situations, derivation of bad deployments to be avoided and production of performance facts which can provide an insight into the use of particular components or applications in certain deployments or in the context of bigger applications.

The Social Network is the medium through which information and knowledge is shared among users. It constitutes a first point of access to the services of the PaaSage platform enabling users to load and store CAMEL models as well as initiate PaaSage deployment workflows. It also comprises facilities for performance analysis and visualisation which can provide insight about the performance of applications under different deployments. Moreover, it offers basic SN services enabling users to join interest groups and write to blogs, to rate applications and deployments as well as be informed when activities of interest take place. Thus, the Social Network is not a

basic SN but an enhanced one which offers additional facilities that can be of great value to cloud users and especially those that are interested in exploiting the PaaSage platform.

Finally, the security layer comprising various components enables the proper authentication and authorisation of users in terms of accessing both information and service/SN resources. As such, it is not specific to the MDDB module but spans across the whole PaaSage platform as it enables the controlled access to the services offered by this platform. By securing information and services, all assets of the PaaSage platform are controlled and this constitutes a major contribution which differentiates PaaSage from other platforms.

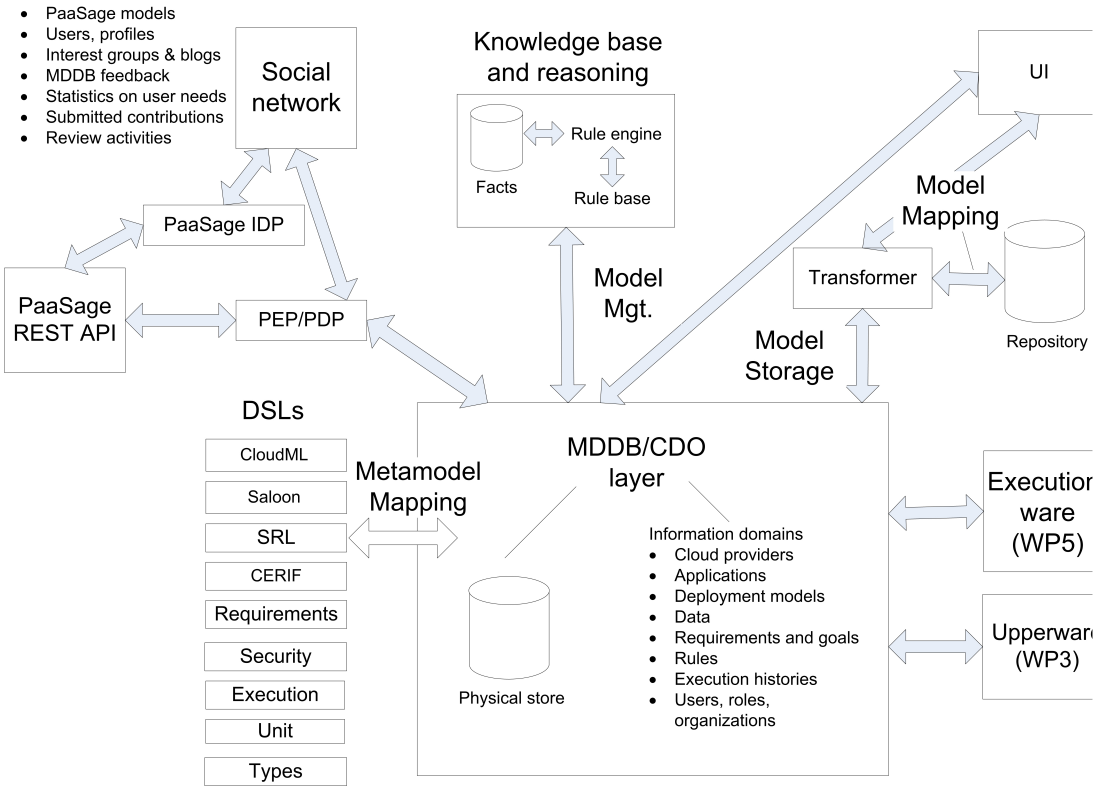

Figure 1 - Metadata database and social network architecture (2015-06-24)

3 Meta-Data Database (MDDB) / CDO

3.1 Introduction

Based on the decisions taken at the consortium meeting at Heraklion, it has been decided that MDDB will be realized through CDO[1] technology. These decisions had a significant effect on the design and core architecture of the MDDB which was significantly updated. To this end, we will attempt to shed light on the various modifications performed by also providing implementation details on the various components comprising the MDDB core. We will also explain what was the evolution of the MDDB schema which has actually been mapped to the CAMEL family of meta-models as they are described in D2.1.2 [3]. Moreover, we will also shortly analyze another important contribution related to the production of guidelines for the storage of models conforming to CAMEL.

This section is organised as follows. First, we analyze in short the evolution of the MDDB schema. Then, we move to analyzing the architecture of MDDB core and the main functionality exhibited by its components. Next, we provide some interesting implementation details. Finally, we conclude by explicating the model storage guidelines specified.

3.2 MDDB Meta-Models

3.2.1 Introduction

The use of CDO technology automates the generation of the schema of the underlying database based on the content of the meta-models which are indirectly manipulated by this database. Indirect manipulation actually means that models conforming to these meta-models when attempted to be stored, will map to the extension of the schema if the respective meta-model is encountered for the first time. To this end, it has been decided that the modelling work in WP4 will focus on complementing basic meta-models re-used, extended or defined in the context of WP2 with the capturing of additional aspects. As such, the work in WP4 concentrated on generating and evolving a set of meta-models encapsulating the aspects of:

a. organisations, roles and users - case of organisation meta-model;
b. execution and adaptation history - case of execution meta-model;
c. security requirements and capabilities - case of security meta-model.

It should be highlighted here that also the scalability rules and metric meta-models (mapping to the SRL language [5]) are considered as joint work conducted in the context of WP2 and WP4.

The modelling work for the three aforementioned meta-models was not conducted from scratch. On the contrary, the original MDDB schema, as defined in D4.1.1 [1], was used as a basis and extended towards: (a) capturing additional parts of information, originally not foreseen (e.g., resource filters) and (b) modifying the way the original information was specified (as in the case of security policies/permissions in the organisation meta-model), by also conforming to a UML-based and not table-based mode of modelling.

[1] http://www.eclipse.org/cdo/documentation/

As the three aforementioned meta-models are well covered in the documentation of CAMEL in WP2 [3], in the sequel, this document will concentrate on a short analysis of the main content and purpose of these meta-models. We should also mention that as the organisation meta-model covers also the specification of security-related information, such as permissions/policies and roles, some further details of this meta-model are provided in Section 4.

3.2.2 Organisation

The main goal of the organisation meta-model is to cover the representation of the most important information for an organisation which is exploited by the PaaSage platform for various tasks, such as authentication and authorisation as well as cloud deployment. The organisation meta-model has been derived from CERIF [6]. As such, it does not only conform to a particular standard but has actually paved the way towards extending it across particular aspects. Moreover, it must be highlighted, as also specified later on in this section, that specific transformation code has been produced able to map CERIF models to organisation ones. In this way, organisations do not need to generate from scratch their organisation models but can rely on existing specifications described according to the CERIF standard.

The class diagram of the organisation meta-model is shown in Figure 2. As it can be seen, an organisation model pertains to a specific organisation or cloud provider, which maps to a special sub-class of organisation devoted to capturing particular generic characteristics for a cloud provider, such as the types of cloud services that it offers. The consideration of cloud providers enables their representation and participation in the PaaSage platform, enabling in this way the update of provider specific models, e.g., specified in Saloon, when new cloud services or existing ones are modified. This also gives the ability to users of the cloud provider to exploit the main facilities offered by the PaaSage platform in order to deploy applications in its cloud infrastructure.

The basic organisation information comprises the name of the organisation, its email and its web address. On the other hand, the information about a cloud provider spans associations to cloud provider models (in Saloon) as well as the description of its own infrastructure in terms of data centres. The latter information can be beneficial as it can allow the correlation of cloud offerings to data centres and the determination of particular specificities pertaining to such a correlation in terms of different prices and level of service or security capabilities (mapping to security controls as it will be indicated in Section 3.2.4).

An organisation model also comprises the specification of roles and users. Roles map to particular entity types which can be associated to a set of security policies via permissions, which are analyzed in detail in Section 4. Users can have many roles and as such, they are able to inherit the permissions specified for these roles. The mapping from roles to users is specified via role assignments. A role assignment includes, in addition, important information, such as when the assignment has started and when it finishes. Such information can allow a security system, part of a specific platform, to check when roles are not any more attributed to users in order to deny access to such users for resources whose access maps to the policies specified for these roles. It can also be exploited for checking abnormal behaviour after access to ensure active security monitoring and enable security adaptation in terms of incident resolution.

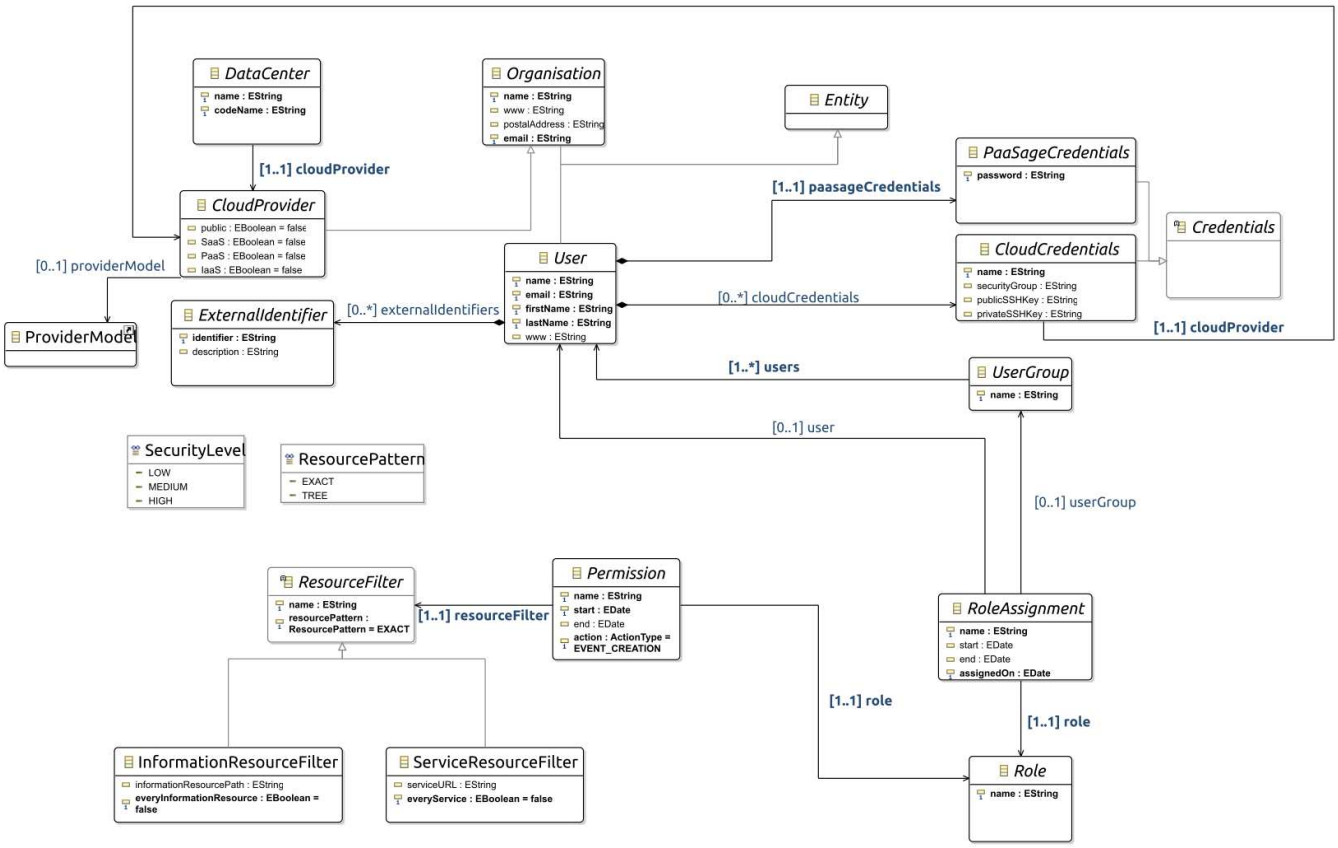

Figure 2 - The class diagram of the organisation meta-model

The specification of users includes information pertaining to their own identification characteristics (last and first name or email) as well as identification characteristics mapping to the PaaSage platform (login). In addition, a user is associated to two different types of credentials: (a) a password to enable its access to the PaaSage platform and to the services provided and (b) cloud credentials which can be used by the PaaSage platform in order to perform deployment tasks on behalf of this user in a particular cloud. Moreover, a user is associated to those models that he/she generates, like deployment or requirement models (see description of deployment and requirement meta-model in CAMEL documentation).

3.2.3 Execution

The goal of the execution meta-model is to capture the execution and adaptation history of cloud-based applications that have been deployed and executed via the PaaSage platform. Execution history spans information concerning complete execution contexts, i.e., application deployments, and their association to monitoring information in the form of various types of measurements, including resource (VM), resource coupling (network), component and application measurements. The production of measurements also leads to the evaluation of Service Level Objective (SLO) requirements and thus execution contexts are also associated to the respective SLO assessments performed. On the other hand, the adaptation history pertains to the capturing of those adaptation actions that have been performed in a specific execution context due to the triggering of scalability rules. Such modelled information can be quite beneficial for the exploiters of the PaaSage platform as it can be used for performance analysis as well as for deriving added-value knowledge. In the case of

WP4, the Knowledge Base can derive such knowledge, in the form of deployments which are appropriate for specific types of applications and of deployments which must be avoided as they have led to the violation of the SLO requirements posed. Such knowledge can also of course be used to update application profiles in the context of the work conducted in the Upperware.

The class diagram of the execution meta-model is shown in Figure 3 (without the root element and cross-references to other meta-models). An execution model comprises a set of execution contexts which map to different deployments for a particular application according to a specific concrete deployment model. The specification of an execution context comprises important information, such as what is the current deployment cost incurred and when the execution context, and thus respective deployment, started and ended. Through such information, we can perform analysis on cost information as well as provide a live feed to application owners in order to have a coherent view of what is the current cost of their applications. In addition, we are able to compute the duration of an execution context which could be exploited for further analysis reasons (e.g., to derive for how long specific deployments are kept live for a certain cloud provider or type of application).

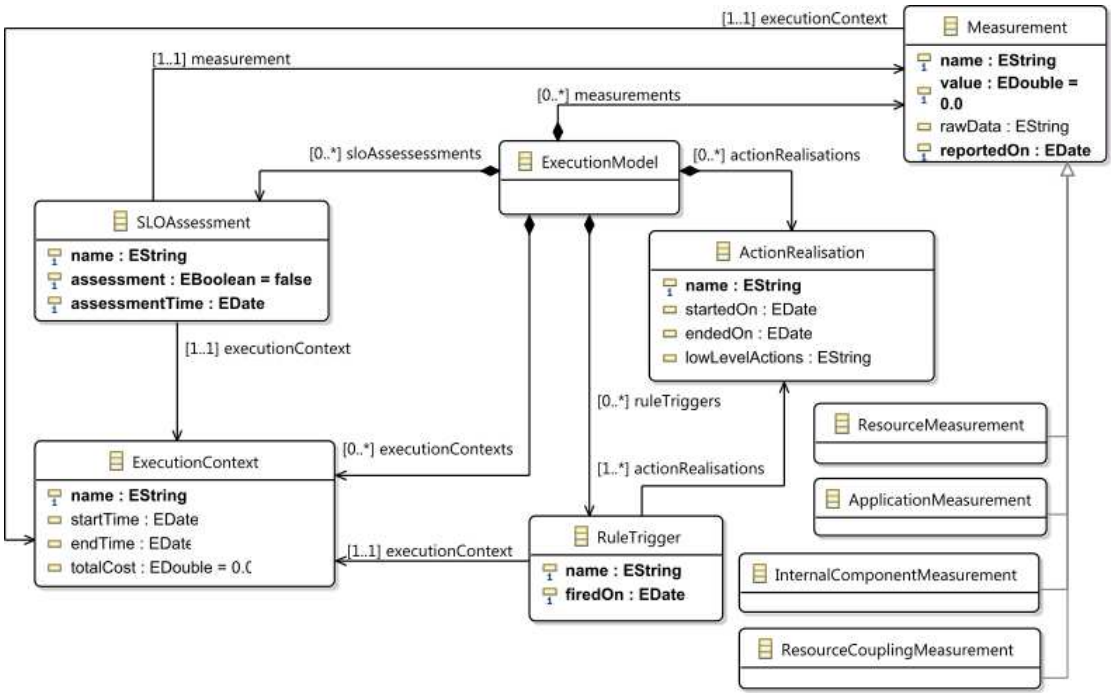

Figure 3 - The class diagram of the execution meta-model

Measurements, SLO assessments and rule triggers are all directly associated to the execution contexts they are related to allowing the correct and appropriate browsing and correlation of the respective information. Measurements are also associated to instances of metrics that are generated in the context of a particular application deployment. As such, we are able to connect SLOs to the metrics that they concern (see description of metric and requirement meta-models in CAMEL documentation) and from the measurements produced from instances of these metrics, we are able to evaluate whether SLO conditions are met. Moreover, as also indicated previously, we are able to provide a live feed to the application owners in terms of the performance of their applications.

3.2.4 Security

The goal of the security meta-model is to provide the constructs to capture the security capabilities of cloud providers and the security requirements posed by users of the PaaSage platform. As requirements are mainly captured by the requirement meta-model (see its description in CAMEL documentation), both the security and the requirement meta-model can complementary capture security requirements in two levels of abstraction. At the high-level, security requirements and capabilities are mapped to security controls which are needed to be in place or have been realized by the corresponding cloud provider, respectively. Such security controls provide a significant view or notion about the actual security level offered by a cloud provider. The high-level security requirements can then be mapped to low-level security ones mapping to particular security SLOs. In this way, we are able to check whether the security level implied by a set of security controls is violated or not through assessing the respective security SLOs associated to these controls. The association between an SLO and a security control is performed indirectly through the correlation of this control with the (security) metric or property that is used in the formation of this SLO.

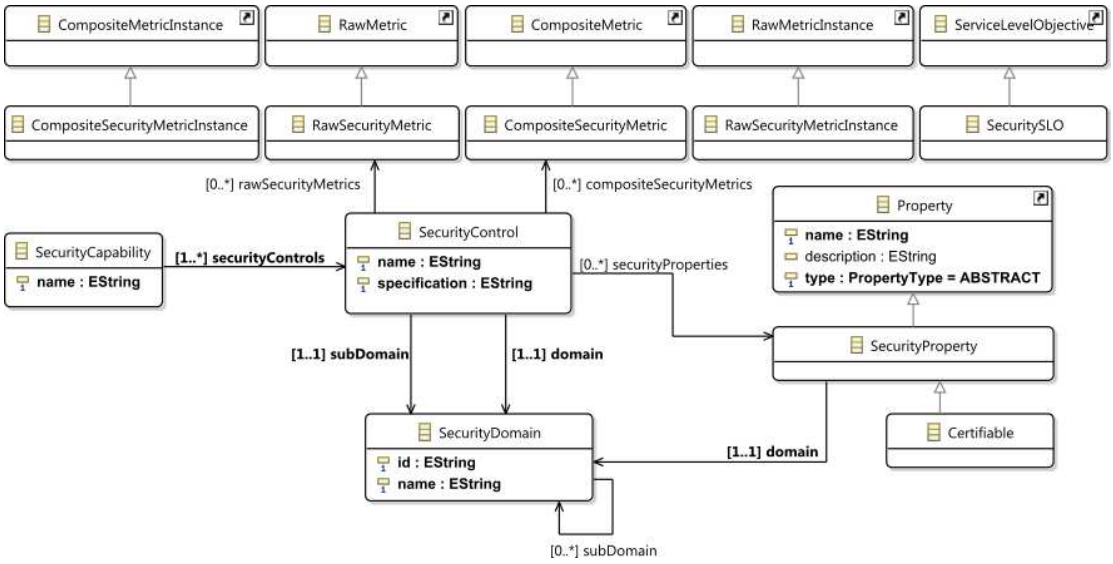

Figure 4 - The class diagram of the security meta-model

The specification of the aforementioned type of security information can provide significant assistance to the following tasks: (a) matching security capabilities and requirements: the provider space in deployment reasoning can be filtered by removing those cloud providers which have not realized one or more of the security controls required by the user; (b) monitoring and assessing security SLOs which can then be mapped to adaptation rules in order to adapt the structure or behaviour of an application to still be in position to exhibit the security level required.

The class diagram of the security meta-model is shown in Figure 4 (without the root element - *SecurityModel*). A security model is mapped to all those constructs that can be used to specify security requirements or capabilities at any of the two levels of abstraction. High-level security capabilities are directly mapped to security controls. Security controls are mapped in turn to security properties and metrics which can be used to formulate security SLOs. Security properties and metrics are extensions of the corresponding constructs defined in the metric meta-model (see its description in CAMEL documentation). The extensions mainly concern the association of these

constructs to the respective security domain and the characterisation of particular types of security properties as certifiable (where this actually means that there is a proof for the values of these properties that can come, in our case, via measurements produced by a particular metric). By extending the metric meta-model constructs and especially the metric ones, we are able to inherit the type of information which is essential for performing the actual measurement of security metrics. Via the complementary subclassing of security SLOs to SLOs (construct/class in requirement meta-model), we provide complete support for the monitoring and assessment of security SLOs, thus actually being able to finally assess whether security service levels are actually achieved.

3.3 MDDB Core Architecture

MDDB Core has its own internal architecture which comprises components dedicated mainly to the manipulation and storage of models. This architecture is shown in Figure 5. As it can be seen, there are four main components which deliver the main functionality of MDDB core. These components are the CDOClient, CDOServer, Importer and CERIF Transformer. Each component has a clear set of distinct responsibilities to perform which are analyzed in detail below.

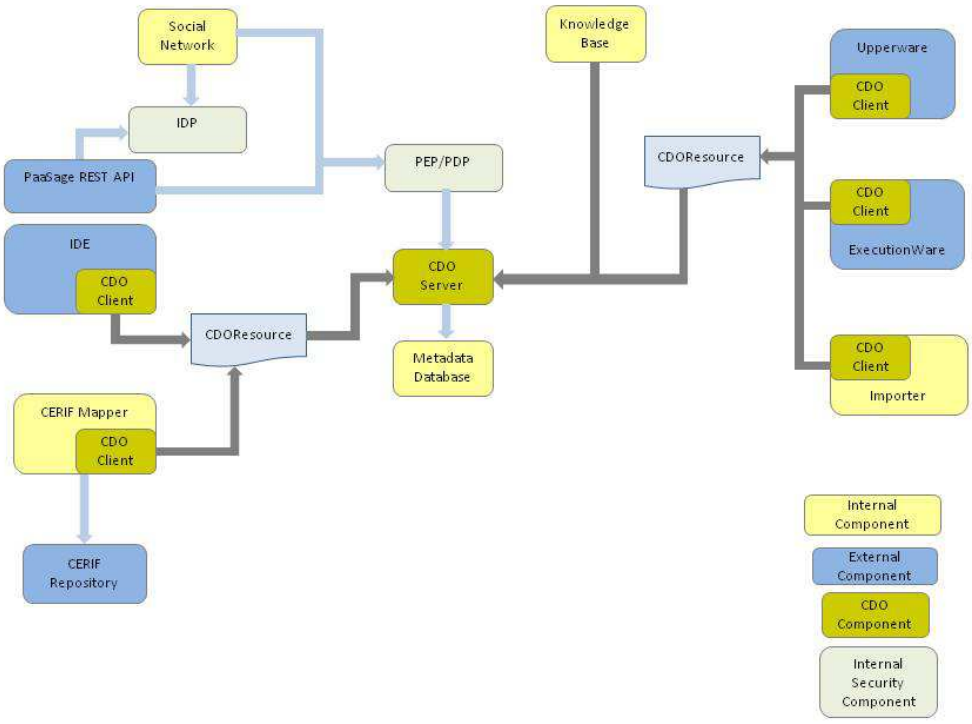

Figure 5 - MDDB Core internal architecture showing also the interaction with both external as well as other internal MDDB components

3.3.1 CDOClient

The CDOClient is the component responsible for the actual management of the models produced or updated by the PaaSage platform components as well as for the issuing of queries. It is the main interaction point with the CDOServer. Thus, any component wishing to interact with the CDOServer, can only use the CDOClient to do so. The CDOClient is also responsible for the authentication of users as it can be

initialized with the PaaSage platform credentials of a specific user. As such, these credentials are then used to establish a session with the CDOServer while they are mapped to particular CDO permissions driving the access to information resources requested in the context of the session that has been established.

The CDOClient provides particular public methods which directly map to the main functionality that it exposes. These methods span the following functionality:

- *Opening and closing CDO transactions or views*. Such functionality can be required in case a component needs to have a greater control over how its model is manipulated and the exact modes and points of interaction with the CDO server. CDO transactions are mainly used for the persistence of models to the underlying database operated by the CDOServer. On the other hand, CDO views are mainly used for the accessing of models stored in CDO (with no capabilities of updating them). Obviously, such functionality goes to lower-levels of CDO interaction and thus can be exploited only by a developer experienced in CDO.
- *Storage of models* (in the CDOServer) in either XMI or CAMEL (textual) form with the option of validating them first (according to OCL constraints).
- *Exporting models to XMI and CAMEL (textual) formats*. Such exportation functionality can then be exploited by visualization or by editing UIs, like the CAMEL textual editor. It should be noted that there are three different modes of exporting: (a) exporting only a model; (b) exporting the model and the models that it cross-references. Even if the latter models have cross-references themselves, these are not resolved in the exportation. The main goal with this mode is to be able to load the exported model as well as follow its cross-references in order to draw additional information without being dependent on a particular CDO Store, where its content can be modified or it can be re-initialized, as the pointers to CDO objects may no longer be valid and lead to exceptions when following the respective cross-references; (c) exporting a model and its cross-models recursively such that every model exported does not have any dependencies with the CDO Store from where it has been exported - such capability is ideal in the case of the CAMEL textual editor as it will enable users to graphically inspect the exported model as well as its cross-references or even update it locally.
- *Exporting of whole CDO Store content* without any dependencies with respect to cross-references on it - this functionality is ideal for back up of the CDO Store such that when there is a fatal error, we can re-initialize it and restore the respective content backed up through the assistance of another functionality that is later on analyzed. The exported content can be in either XMI or CAMEL textual format. This enables users to review the content with either the Eclipse XMI Tree Editor or the CAMEL's textual editor, respectively.
- *Querying of models via different query languages*. Currently, this depends on the types of CDO Stores on which the CDOServer directly manipulates. The DBStore is an implementation that fully realizes all CDO features but maps to the sole use of SQL as the query language. This creates the disadvantage that the developer needs to know the pattern of matching meta-model entities to the tables of the underlying database. The HibernateStore is a hibernate-based implementation of CDO which lacks some advertized CDO features. On the other hand, as HQL (Hibernate Query Language) is a high-level language

above the level of SQL, it allows the SQL-like posing of queries to direct meta-model entities (i.e., classes/concepts). As such, it seems more appropriate for the developer as it does not impose any particular knowledge to be acquired by the developer apart from the basic knowledge of the meta-models whose models are to be manipulated.

- *Validation of models*: CDO always validates models according to the semantics of the respective meta-model to which they must conform. However, in case that the domain semantics are enriched with OCL constraints, then an extra validation is required over these models which is exactly the functionality offered by the CDOClient.
- *Loading of models*: the CDOClient enables users to load models into main memory from specific resources (from the file system or the web). The user can then process and modify the model in order to finally store it in the CDOStore. The loading of both XMI and CAMEL textual models is supported.
- *Importing of models*: XMI and CAMEL textual models can be imported directly into the CDO Store from the file system.
- *Loading of set of models*: This can be used to restore the content of a CDOStore (see above CDOStore export functionality). The models can be in either XMI or CAMEL textual form. The loading of models respects their cross-references.
- *Model transformation from XMI to Xtext and vice versa*: this is an interesting functionality that can be exploited by the developers and end-users for: (a) mapping models generated from the CAMEL textual editor to XMI files (to subsequently store them or use them in the context of a XMI-consuming component) and (b) mapping models from XMI to Xtext (e.g., to transform a model previously generated from a tree-editor to a textual form to be exploited for further editing in the CAMEL textual editor).
- *Event notification*: the CDOClient allows developers to register listeners which can then be informed when the MDDB/CDOStore has been updated in order to capture those updates/modifications that interest them (e.g., the CP Generator might be interested in detecting updates on provider models as this will enable it to produce constraint models that better reflect the current situation)

The aforementioned CDOClient functionality can be considered as complete as it can be exploited in order to cater for different scenarios with respect to the manipulation of models. Thus, no other extensions are expected to be performed on this component apart from any type of bug fixes.

3.3.2 CDOServer

The CDOServer is a server-based component which listens to requests originated from a CDOClient and attempts to satisfy them. The CDOServer actually manages a CDOStore which encapsulates a specific database (which we clearly called MDDB in the context of this project). A CDOStore can be independent of the underlying database technologies. In this sense, through the use of the respective drivers, a CDOStore can operate over different types of relational databases or even over NoSQL databases. In this respect, we have attempted to include the existing (CDO)

support to those databases which are widely used and have been reported not to exhibit any major issue, including MySQL, HQLDB, and Postgres. The individual use of each database is set through appropriately configuring the CDOServer.

The interaction between a CDOServer and a CDOClient is established through a CDOSession. Any transaction or view that can then be exploited by a CDOClient can only be opened through this CDOSession. As the security feature of CDOServer has been included in its realisation, CDOSessions can also be secured. This means that before they are established, the CDOClient should provide valid (CDO) credentials (mainly in the form of user name and password). If the credentials are valid, then the CDOSession is opened but access to the resources is limited depending on the permissions that have been set and mapped to the respective users (via, e.g., the roles assigned to them). In this way, we have managed to secure the access to the information resources (i.e., models) that are stored in the MDDB. Additional details about how this has been completely achieved can be drawn by studying Section 4.

3.3.3 Importer

An empty database is always not of a great value, especially as it should contain information that needs to be exploited by the respective system. To this end, a new component has been developed, called *Importer*, which takes the responsibility of storing in the CDOStore, once it is created and the CDOServer has started running (this constitutes a deployment/execution requirement for the PaaSage platform itself), all appropriate initial and required models. Such models include, use case models, provider models, locations, basic metric and security models, and a basic unit model. These models have a different usage context:

- *Use case models* are needed mainly for showcasing purposes, especially as it might not be desirable to involve any initial showcase step required to store such models into the CDOStore. However, we can consider two types of use case models: (i) organisation models and (ii) CAMEL models which describe the deployment, quality/performance and location requirements for the use case. The former should pre-exist in the CDOStore to enable users of the use-case organisation to perform any kind of manipulation on the use-case CAMEL models, such as the initiation of the PaaSage workflow with such a model. The latter are not required to pre-exist but they can be stored afterwards, once they are specified in the respective UI (such as the CAMEL textual editor).
- *Provider models* are required for the generation of constraint problems and the subsequent filtering of the provider space. In this sense, provider models for a basic set of cloud providers must pre-exist in the CDOStore/MDDB in order to allow the immediate exploitation of the PaaSage platform through the initiation of the corresponding workflow. Obviously, in the sequel, additional provider models can be stored to be additionally exploited.
- *Location models* are required in order to express location requirements (either at the physical or cloud level). Such models could be specified and stored while the PaaSage platform is in operation or they could be available from the very beginning. In the first case, the users have to discover whether a particular location exists in the existing location model. If this is the case, then they have to update it to include this new location. In the second case, the users just specify their location requirements by just selecting the appropriate

location from the pre-existing location model. From these two alternative cases, we have chosen the second one for the following reasons: (a) the users are not bothered in specifying locations and can concentrate immediately on providing location requirements; (b) duplication of locations is avoided as well as any other human-originating errors; (c) availability of physical location ontologies. To this end, we have selected the FAO (Food and Agriculture Organisation of United Nations) physical location/geopolitical ontology[2] and used it as input to generate the respective basic location model. This model has been enriched via incorporating cloud locations from those cloud providers that have already been modelled. Of course, this location model will be constantly improved as new cloud providers are used by the platform. However, we need to stress that it constitutes a quite detailed model which contains a standardized hierarchy of locations (in the form of regions hierarchy with countries as leaves). As such, users/developers can exploit normal region or country names (in one or more languages) to find the locations that interest them or even specific ISO codes.

- *Basic metric models* include a basic set of attributes and metrics that can be used to specify metric conditions for SLOs or non-functional events. As such, it is important that such models pre-exist in the platform to not burden users in specifying such basic quality of service (QoS) terms along with the respective modelling details needed. Of course, such models can then be enriched with additional domain-dependent or independent attributes and metrics, based on the requirements of the user and the respective application domain. It should be highlighted here that we have focused on including in this basic metric model especially metrics that can be immediately measured by the platform through respective sensors/probes. We have also opted for including some basic, quite common and widely used metrics which will surely lead to the development of the respective requirements and probes at the user side.
- *Basic security model*: in order to assist in the formulation of security requirements and capabilities at the main two levels of abstraction, we have created a security model which includes the specification of all the security controls as specified by the Cloud Security Alliance (CSA) as well as the specification of a set of security domain, attributes and metrics that have recently be proposed by the European Commission as those constructs which should definitely be incorporated in cloud-based SLAs [7]. In this respect, we also support standardization or recommendation efforts which seems to have a complete or representative coverage of the security constructs that must be in place before security requirements are expressed.
- *Basic unit model*: this model contains all possible units that can be expressed based on the (unit) enumeration that has been incorporated in the specification of the unit meta-model (see its description in CAMEL documentation). As such, such a model enables users to select units that can be subsequently used in the specification of metrics or feature attributes.

Apart from its main responsibility to populate the CDOStore, the Importer also creates the main CDOStore structure according to the model storage guidelines that are analyzed in Section 3.5. This enables the proper support of these guidelines as well as the immediate retrieval of models based on their CDO path.

[2] Available at: http://www.fao.org/countryprofiles/geoinfo/geopolitical/resource/

3.3.4 Model Transformers

The plan of work in WP4 includes the generation of model transformers which are responsible for the transformation of models from one standardized language (like CERIF) to CAMEL. This enables the exploitation of models not specified in CAMEL, thus relieving users from the burden of performing this task in a manual transformation manner, enabling them to concentrate on basic tasks, like the deployment of their applications.

Four languages have been planned to be supported:

a. WS-Agreement for the coverage of user quality/location requirements,
b. CERIF for the coverage of organisation models,
c. XACML for the coverage of security policies and
d. TOSCA for the coverage of deployment models.

As it can be seen, most of the languages are standardized or map to existing standards, and this increases the possibility that such languages have been already been exploited by the users in order to model the respective aspects (e.g., organisation or deployment).

From the transformation tasks pertaining to these four languages, only one, concentrating on supporting CERIF, has been finished completely. In addition, good progress over supporting the TOSCA language has been reached. To this end, in the sequel, we analyze in detail the main functionality of the respective two components.

3.3.4.1 CERIF Transformer

In order to support import of organizational models from third-parties into the MDDB, a tool to import CERIF models into MDDB has been developed. The tool enables conversion of organisational CERIF models, in particular organization structure as well as employee information and automatic creation of accounts based on information in CERIF input files. The tool allows for mapping of CERIF defined taxonomies into specific roles in CAMEL (e.g. user roles or organization types) as well as conversion of specific entities descriptions (e.g. concrete organization, its structure and its employees).

The input to the mapping tool includes the CERIF taxonomy (which is optional), the CERIF organization description and connection details to the CDO Server. An excerpt from a sample taxonomy is presented below:

```
<cfClassScheme>
    <cfClassSchemeId>CAMEL</cfClassSchemeId>
    <cfName cfLangCode="en" cfTrans="o">CAMEL</cfName>

    <cfClass>
      <cfClassId>CAMEL.Role</cfClassId>
      <cfTerm cfLangCode="en" cfTrans="o">Role</cfTerm>
      <cfDef cfLangCode="en" cfTrans="o">Users role</cfDef>
    </cfClass>

    <cfClass>
      <cfClassId>CAMEL.Administrator</cfClassId>
```

```
      <cfTerm cfLangCode="en" cfTrans="o">Administrator</cfTerm>
      <cfDef cfLangCode="en" cfTrans="o">Users role</cfDef>
      <cfClass_Class>
       <cfClassId2>subClassOf</cfClassId2>
       <cfClassSchemeId2>RDFS</cfClassSchemeId2>
       <cfClassId>CAMEL.Role</cfClassId>
       <cfClassSchemeId>CAMEL</cfClassSchemeId>
      </cfClass_Class>
    </cfClass>
    ….
</cfClassScheme>
```

The actual CERIF organization description can use the concepts from the taxonomy to define actual organizational entities and users, such as below:

```
  <cfPers>
    <cfPersId>54e91de6-6e95-4dbb-9a5d-4986817a1a00</cfPersId>
    <cfGender>f</cfGender>
    <cfPersName_Pers>
      <cfPersNameId>name-54e91de6-6e95-4dbb-9a5d-4986817a1a00</cfPersNameId>
      <cfClassId>id-known-as-name</cfClassId>
      <cfClassSchemeId>id-person-name-types</cfClassSchemeId>
      <cfFamilyNames>Pagac</cfFamilyNames>
      <cfFirstNames>Edmond</cfFirstNames>
    </cfPersName_Pers>
    <cfPers_EAddr>
      <cfEAddrId>priscilla@cartwright.ca</cfEAddrId>
      <cfClassId>Email</cfClassId>
      <cfClassSchemeId>CAMEL</cfClassSchemeId>
    </cfPers_EAddr>
    <cfPers_Class>
      <cfClassId>CAMEL.Administrator</cfClassId>
      <cfClassSchemeId>CAMEL</cfClassSchemeId>
      <cfStartDate>2013-02-10T07:50:30</cfStartDate>
      <cfEndDate>2015-02-13T02:30:51</cfEndDate>
    </cfPers_Class>
    <cfPers_OrgUnit>
      <cfOrgUnitId>53b72cef-194c-4a4b-9611-3a662117cfb4</cfOrgUnitId>
      <cfClassId>Person</cfClassId>
      <cfClassSchemeId>CAMEL</cfClassSchemeId>
    </cfPers_OrgUnit>
  </cfPers>
```

Current version of the tool supports the following concepts from the CAMEL organizational model:

- Organization
- User
- Role
- RoleAssignment
- ExternalIdentifier
- CloudProvider
- PaaSageCredentials

3.3.4.2 TOSCA Transformer

Transformation Language Selection Rationale

Before entering into any kind of implementation effort, a research over existing transformation languages was performed. An overview of the three main languages considered along with their characteristics is given in Table 1.

Language	Direction	Paradigm	Implementation	Documentation
ATL	uni-directional	mixed	yes	adequate
QVTo	uni-directional	imperative	yes	limited
QVTr	Bi-directional	declarative	partial	limited

Table 1 - Overview of the transformation languages considered

ATL is a well-established mapping language. It is well-documented and has multiple examples of transformations of various models. The downside of using ATL is that it can only provide an uni-directional mapping, while one of the main objectives was to support a bi-directional mapping due to the following reasons: (a) support users in transforming models already specified in TOSCA to CAMEL and (b) use the facilities of the PaaSage platform, the CAMEL textual editor in particular, to create a CAMEL model out of which a TOSCA model can be produced to be exploited according to the main goals of the user (e.g., use it in the user's TOSCA based framework for application deployment).

This was the main reason to search for other languages, capable of writing bi-directional mappings. This search led to the QVT languages. QVTo seems to be implemented, but transformations seemed to be very verbose, partly because of it's imperative nature. QVTr seemed to be the best of the QVT languages, but it lacks a complete well-maintained implementation.

Due to the better capturing of most of the requirements set, a specific tool supporting QVTr was exploited, which however did not properly work with the recent versions of Eclipse versions which were decided to be followed by the project. Apart from this, the lack of documentation made harder the specification of the transformation logic via QVTr. To this end, the final decision that was faithfully followed is to use ATL due to its adequate documentation and the respective tools which are constantly updated to match the newest Eclipse versions.

Transformation Logic

A coarse visualisation of the yet incomplete mapping defined in ATL is given by Table 2. This is currently limited roughly to the elements appearing in the deployment meta-model in CAMEL. In particular, *RelationshipTemplates* are mapped to either *Hostings* or *Communications* while *NodeTemplates* are either transformed to *VMs* or *InternalComponenst*. The coverage of the other meta-models is restricted to the following two mappings: *RequirementTemplates* are mapped to the respective *Requirement* concept in CAMEL, while *CapabilityTemplates* to the provider models. In general inheritance isn't supported yet, and might not need to be trivial to implement either.

TOSCA	Conditions	CAMEL	Notes
Definitions		*CamelModel* *DeploymentModel* *RequirementModel*	
NodeTemplate	type = *VM*	*VM* *Configuration*	
Requirement	type = *VMRequirements*	*VMRequirementSet* *OSRequirement* *ImageRequirement* *QuantitativeHardwareRequirement* *QualitativeHardwareRequirement*	
NodeTemplate	type = *InternalComponent*	*InternalComponent* *Configuration*	
Requirement	type = *MandatoryCommunicationRequirement*	*RequiredCommunication*	*isMandatory* = true
Requirement	type = *CommunicationRequirement*	*RequiredCommunication*	*isMandatory* = false
Capability	type = *CommunicationCapability*	*ProvidedCommunication*	
RelationshipTemplate	type = *Communication*	*Communication* *Configuration*	
Requirement	type = *HostRequirement*	*RequiredHost*	
Capability	type = *HostCapability*	*ProvidedHost*	
RelationshipTemplate	type = *Hosting*	*Hosting* *Configuration*	

Table 2 - The current set of mappings between TOSCA and CAMEL elements

Example Transformation

In this subsection, a specific example of the way the current implemented TOSCA transformer works is provided. The input to the transformer is a deployment specification in TOSCA concerning one component which has to be deployed on a particular VM with specific requirements set on it, spanning the following:

- the OS should be Debian Jessie
- CPU frequency should be between 1 and 2.5 GHz
- Number of cores should be between 1 and 2
- The amount of RAM should be between 512 and 1024 Mega Bytes
- The size of disk storage should be between 4 and 20 Giga Bytes

This component should be downloaded from a particular URL. This input is illustrated in the following snippet.

```
<?xml version="1.0" encoding="UTF-8" ?>
<tosca:Definitions               name="tosca2camel"               id="id0001"
targetNamespace="http://example.be"           xmlns:tosca="http://docs.oasis-
open.org/tosca/ns/2011/12"        xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:paasage="http://paasage.eu/tosca/2015/07">
  <tosca:Import                   namespace="http://paasage.eu/tosca/2015/07"
location="file:///home/kevin/vcs/ugent/stage/types.xsd"
importType="http://www.w3.org/2001/XMLSchema"/>
  <tosca:ServiceTemplate id="id0002">
    <tosca:TopologyTemplate>
      <tosca:NodeTemplate id="DebianVM" type="VM">
        <tosca:Requirements>
          <tosca:Requirement     id="DebianVMRS"     name="vmRequirementSet"
type="VMRequirements">
            <tosca:Properties>
              <paasage:osOrImageRequirement>
                <paasage:osRequirement>
                  <paasage:os>Debian Jessie</paasage:os>
                  <paasage:is64os>false</paasage:is64os>
                </paasage:osRequirement>
              </paasage:osOrImageRequirement>
              <paasage:quantitativeHardwareRequirement>
                <paasage:minCPU>1.0</paasage:minCPU>
                <paasage:maxCPU>2.5</paasage:maxCPU>
                <paasage:minCores>1</paasage:minCores>
                <paasage:maxCores>2</paasage:maxCores>
                <paasage:minRAM>512</paasage:minRAM>
                <paasage:maxRAM>1024</paasage:maxRAM>
                <paasage:minStorage>4</paasage:minStorage>
                <paasage:maxStorage>20</paasage:maxStorage>
              </paasage:quantitativeHardwareRequirement>
              <paasage:qualitativeHardwareRequirement>
                <paasage:minBenchmark>2.0</paasage:minBenchmark>
                <paasage:maxBenchmark>2.5</paasage:maxBenchmark>
              </paasage:qualitativeHardwareRequirement>
            </tosca:Properties>
          </tosca:Requirement>
        </tosca:Requirements>
        <tosca:Capabilities>
          <tosca:Capability    id="IRCProv"    name="providedCommunications"
type="CommunicationCapability">
            <tosca:Properties>
              <paasage:portNumber>1024</paasage:portNumber>
            </tosca:Properties>
          </tosca:Capability>
          <tosca:Capability   id="CoreIntensiveHostC"   name="providedHosts"
type="HostCapability"/>
        </tosca:Capabilities>
      </tosca:NodeTemplate>
      <tosca:NodeTemplate id="ExperimentManager" type="InternalComponent">
        <tosca:Requirements>
          <tosca:Requirement  id="EvilPortReq"  name="requiredCommunications"
type="CommunicationRequirement">
```

```
            <tosca:Properties>
              <paasage:portNumber>666</paasage:portNumber>
            </tosca:Properties>
          </tosca:Requirement>
      <tosca:Requirement   id="Ohla"   name="requiredMandatoryCommunications"
type="MandatoryCommunicationRequirement">
            <tosca:Properties>
              <paasage:portNumber>69</paasage:portNumber>
            </tosca:Properties>
          </tosca:Requirement>
          <tosca:Requirement   id="Foefelare"   name="requiredCommunications"
type="CommunicationRequirement">
            <tosca:Properties>
              <paasage:portNumber>1024</paasage:portNumber>
            </tosca:Properties>
          </tosca:Requirement>
          <tosca:Requirement    id="CoreIntensiveHost"    name="requiredHost"
type="HostRequirement"/>
        </tosca:Requirements>
        <tosca:DeploymentArtifacts>
          <tosca:DeploymentArtifact      name="ExperimentManagerImplDownload"
artifactType="configuration">
            <paasage:downloadCommand>wget
http://www.google.be</paasage:downloadCommand>
          </tosca:DeploymentArtifact>
        </tosca:DeploymentArtifacts>
      </tosca:NodeTemplate>
      <tosca:RelationshipTemplate                 id="ExperimentManagerToIRC"
type="ExperimentManagerToIRCType">
        <tosca:SourceElement ref="Foefelare"/>
        <tosca:TargetElement ref="IRCProv"/>
        <tosca:Properties>
          <paasage:type>remote</paasage:type>
        </tosca:Properties>
      </tosca:RelationshipTemplate>
      <tosca:RelationshipTemplate id="CoreIntensiveHostH" type="Hosting">
        <tosca:SourceElement ref="CoreIntensiveHost"/>
        <tosca:TargetElement ref="CoreIntensiveHostC"/>
      </tosca:RelationshipTemplate>
    </tosca:TopologyTemplate>
  </tosca:ServiceTemplate>
  <tosca:RelationshipType name="ExperimentManagerToIRCType">
    <tosca:DerivedFrom typeRef="Communication"/>
  </tosca:RelationshipType>
  <tosca:RelationshipTypeImplementation
name="ExperimentManagerToIRCTypeImpl"
relationshipType="ExperimentManagerToIRCType">
    <tosca:ImplementationArtifacts>
      <tosca:ImplementationArtifact
interfaceName="requiredCommunicationConfiguration"
artifactType="configuration">
        <paasage:downloadCommand>wget
https://duckduckgo.com</paasage:downloadCommand>
      </tosca:ImplementationArtifact>
    </tosca:ImplementationArtifacts>
```

```
  </tosca:RelationshipTypeImplementation>
</tosca:Definitions>
```

Based on the above input, the transformer will perform the mapping and produce the following equivalent CAMEL model in XMI form:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<camel:CamelModel    xmi:version="2.0"    xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:camel="http://www.paasage.eu/2015/06/camel"
xmlns:requirement="http://www.paasage.eu/2015/06/camel/requirement"
name="tosca2camel">
  <deploymentModels name="tosca2camelDM">
    <internalComponents name="ExperimentManager">
      <configurations                    name="ExperimentManagerImplDownload"
downloadCommand="wget http://www.google.be"/>
      <requiredCommunications name="EvilPortReq" portNumber="666"/>
      <requiredCommunications name="Foefelare" portNumber="1024"/>
      <requiredCommunications          name="Ohla"          portNumber="69"
isMandatory="true"/>
      <requiredHost name="CoreIntensiveHost"/>
    </internalComponents>
    <vms                                                    name="DebianVM"
vmRequirementSet="//@deploymentModels.0/@vmRequirementSets.0">
      <providedCommunications name="IRCProv" portNumber="1024"/>
      <providedHosts name="CoreIntensiveHostC"/>
    </vms>
    <communications        name="ExperimentManagerToIRC"        type="REMOTE"
providedCommunication="//@deploymentModels.0/@vms.0/@providedCommunications.
0"
requiredCommunication="//@deploymentModels.0/@internalComponents.0/@required
Communications.1">
      <requiredPortConfiguration            name="ExperimentManagerToIRCRPC"
downloadCommand="wget https://duckduckgo.com"/>
    </communications>
    <hostings                                        name="CoreIntensiveHostH"
providedHost="//@deploymentModels.0/@vms.0/@providedHosts.0"
requiredHost="//@deploymentModels.0/@internalComponents.0/@requiredHost"/>
    <vmRequirementSets                                      name="DebianVMRS"
qualitativeHardwareRequirement="//@requirementModels.0/@requirements.2"
quantitativeHardwareRequirement="//@requirementModels.0/@requirements.1"
osOrImageRequirement="//@requirementModels.0/@requirements.0"/>
  </deploymentModels>
  <requirementModels name="tosca2camelRM">
    <requirements  xsi:type="requirement:OSRequirement"  name="DebianVMRSOSR"
os="Debian Jessie" is64os="false"/>
    <requirements      xsi:type="requirement:QuantitativeHardwareRequirement"
name="DebianVMRSQuantHR" minCPU="1.0" maxCPU="2.5" minCores="1" maxCores="2"
minRAM="512" maxRAM="1024" minStorage="4" maxStorage="20"/>
    <requirements       xsi:type="requirement:QualitativeHardwareRequirement"
name="DebianVMRSQualHR" minBenchmark="2.0" maxBenchmark="2.5"/>
  </requirementModels>
</camel:CamelModel>
```

As it can be seen, the TOSCA model was transformed into a CAMEL model which comprises a deployment and a requirement model, where the latter specifies the

hardware and operating system requirements imposed on the sole VM defined in the deployment model. In addition, the deployment model includes the hosting of the internal component to the VM as well as configuration information about how to download this component.

3.4 Implementation Details

The *CDOClient* has been implemented in Java as a standalone component which can be exploited by the PaaSage platform components. It has been built on top of the basic CDO, EMF, OCL and Xtext libraries. It depends on the CAMEL meta-model library as it exploits the domain model as well as Xtext specific classes (part of the CAMEL textual editor code) in order to support the transformation from XMI to textual (Xtext) models. This component is accompanied with a configuration file which enables the setting of various parameters pertaining to the host, port and repository/store name information of the CDOServer and to the level of logging.

The *CDOServer* is another Java component which can be used as a server in the PaaSage platform to manage the access to the MDDB. It has immediate dependencies with CDO, EMF and OCL as well as with the CAMEL meta-model library (due to a particular restriction of the security feature - normally any model can be stored in the CDOServer as long as its package/meta-model has been registered). Similarly to the case of CDOClient, this component is accompanied with a configuration file which can be used to set-up the following information: (a) port to listen to incoming connections, name of repository to handle and type of store, (b) details about how to connect to the underlying database (c) security feature setting (on or off), and (d) level of logging.

The *Importer* standalone component has been implemented in Java and directly depends on CDOClient as it needs to store the basic models used to initialize the CDOStore. It also depends on CAMEL meta-model library (but this dependency is already covered by the CDOClient library) as it requires to manipulate particular domain classes. Moreover, it depends on the Jena library in order to be able to load the location ontology and process it such that the CAMEL location model can be produced. This component is not accompanied with its own configuration file but the respective component that it exploits (CDOClient) needs the existence of such a file.

The CERIF-to-CAMEL conversion tool has been developed in the Clojure language and can be deployed as a regular JAR file on any JVM using the command line or can be integrated as a library with other Java projects. With respect to the version described in the previous deliverable [1], the conversion tool has been updated to operate over CAMEL domain code. In order to support this, a custom CDOClient has been implemented in Clojure and updated to reflect changes in the latest version of CAMEL.

3.5 Model Storage Guidelines

We have come up with a particular set of guidelines regulating the storage of models in terms of structure of the CDO Repository and the names of these models/resources. Such guidelines must be followed by both developers and end-users to avoid particular issues from occurring which include: (a) the repetition of information and (b) naming conflicts for models/resources which can hinder their storage in the CDO Repository. These guidelines are shortly analyzed in this section while a complete document further elaborating on them is provided in the Appendix of this deliverable.

The guidelines rely on the solution sketched in Figure 6. The solution envisaged requires that a CDO repository should be organized in such a way that it depends on the type of information that needs to be stored. This solution is twofold: (a) first, we need to identify the CDO folder paths on which a particular model has to be stored and (b) second, we need to determine the name of the CDOResource to contain this model (which should be contained in turn in the designated CDO Folder path. Both parts of the solution are needed in order to establish a particular and unique CDO resource path through which the model mapped to a specific CDOResource has to be stored.

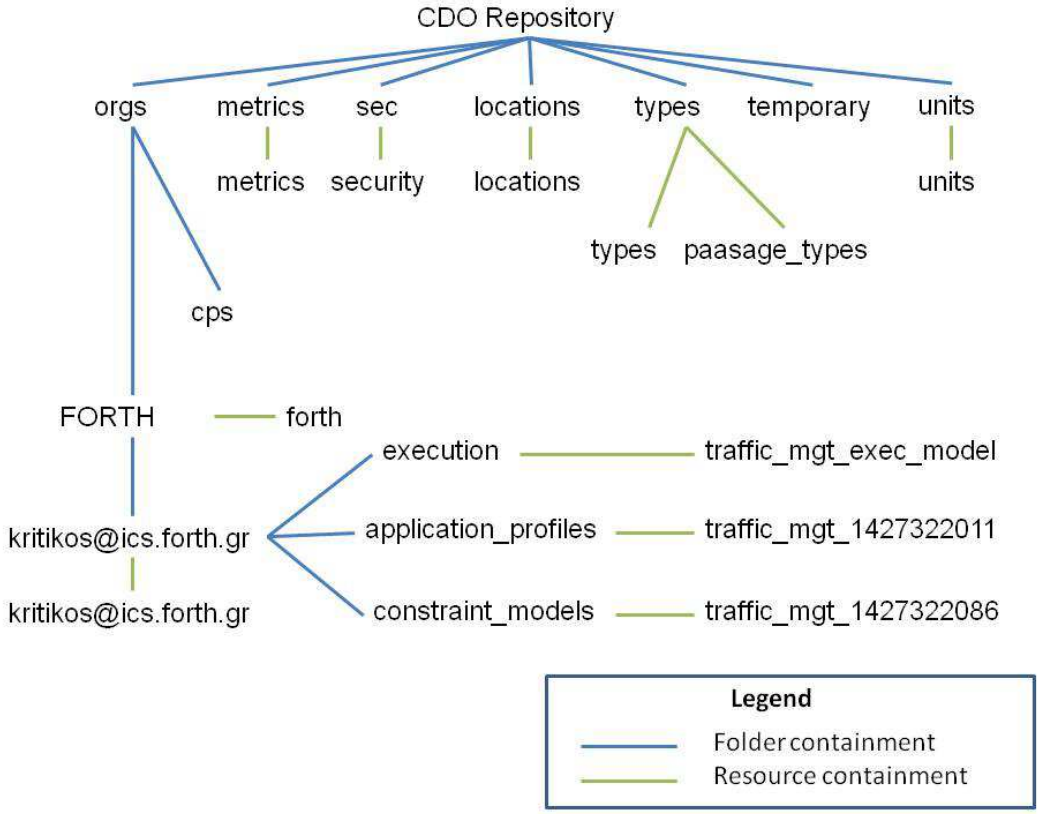

Figure 6 - Solution envisaged for model storage in CDO

As it can be easily seen from Figure 6, the solution partitions the CDO repository space into organisation specific and global directories/folders. The global folders map to basic models that become available in order to assist users and developers specify their models in a quicker and more compact way. The organisation specific part attempts to gather in a structured way all the information that has been generated and is owned by a particular organisation. The structuring is required in order to distinguish between the different types of organisations (normal and cloud providers) as well as discriminate which users of an organisation have created which models (owned by that organisation). The structuring involves also the splitting of the user models into different categories according to their type where : (a) a single CAMEL model storing information about the user applications, deployment models, requirements and metric models is stored at the user folder level, and (b) execution, application profiles and constraint models are stored in separate folders. The rationale for this is that it is better to have all models mapping to user requirements in a single encompassing model (this also caters for the circular dependencies that exist between such models as well as for their immediate access through a single place) while the

unification of other models is not needed as such models can be quite big, as it can be the case of execution models (where their splitting can be quite meaningful as someone could then focus its analysis on execution models of only one or applications of a similar type).

In our opinion, the guidelines formulated are correct and lead to a storage mode which is error-free, as long as the user/developer conforms to the semantics of the particular domain. Thus, these guidelines, along with the careful integration of the meta-models of CAMEL into one coherent meta-model, should be definitely followed by the PaaSage members as this would lead to a better integration of the platform at the information level.

4 Security, Privacy, Trust

4.1 Introduction

While the initial goal of the MDDB was to secure the access to the information it contains, it became apparent that there are additional resources for which controlled access must be in place. To this end, it was decided to come up with a solution which is able to secure all types of resources that are offered by the PaaSage platform. Such a solution will obligatory work for single-site deployments of the platform, while it could evolve to a multi-site solution in the future, if such a need arises.

Before analysing this solution realized, we should also mention particular principles and requirements under which the solution has been designed. These principles and requirements span the following:

- Unique credentials are needed for accessing all types of services or information of the PaaSage platform - this facilitates users as they do not need to keep in mind different credentials for MDDB and different credentials for the SN or the REST API.
- Different permissions should be created for different roles of an organisation by default; administrators of these organisations can then update these permissions according to the organisation needs and policies. This facilitates and accelerates the specification of permissions by considering that more or less the same roles across different organisations map approximately to the same set of permissions.
- An organisation should have a control on which users from other organisations (indicated from now on as external users) have what type of access to which information owned by this organisation. We actually consider here that MDDB can be considered as a multi-tenant store where each organisation has its own private space and controls the external access to such a space.
- A super administrator[3] is needed to address possible security issues requiring a role having full access to the whole MDDB space.
- External identity providers can be exploited in a similar manner as the internal identity provider of MDDB - this means that there should be a capability which will enable to exploit the externally certified information in order to properly identify a user and map it to the respective roles assigned to it.
- Adoption of security standards, such as SAML and XACML, is more than encouraged as it not only enables the conformance to such standards but also the re-use of existing state-of-the-art (open-source) software in order to realise part, if not the whole, required authentication and authorisation functionality.
- As CDO offers a particular security technology to secure access to CDO repositories, it is advocated that this technology should be exploited in order not to lose time and redevelop the required functionality from scratch.

[3] Such a role can be also delegated to creating the admin users for each organisation while additional details about these organisations, including other users, can then be specified by them.

4.2 Solution and Architecture

The solution [4] that we have developed relies on the existence of particular security components which cooperatively interact with the CDO Repository. These components are mainly used to authorize the access to service and SN-based resources. This means that that we have taken the direction of exploiting the security features of the CDO technology which enable us to authorize the access to information resources (i.e., models stored in MDDB) through the interaction between the CDOClient and CDOServer. We should also highlight here that security components interact with the CDO with the main purpose to retrieve the required security information (user, role and permission information) and thus be able to fulfil the respective security tasks.

The overall architecture of our envisaged security solution is depicted in Figure 7. As it can be seen, there are six main components which have particular responsibilities to take care of. These components are now analyzed in more detail. The *CDOClient* is exploited by any other component in the architecture to access the information resources of the underlying CDO repository. This access can involve normal information, in the form of CAMEL models, needed by PaaSage components or UIs, or security information (contained in organisation models) required by the two main security components, mainly IDP and PEP/PDP, responsible for user authentication and authorisation. Obviously, we need to secure the access to these two types of information and CDO security is actually used in this case.

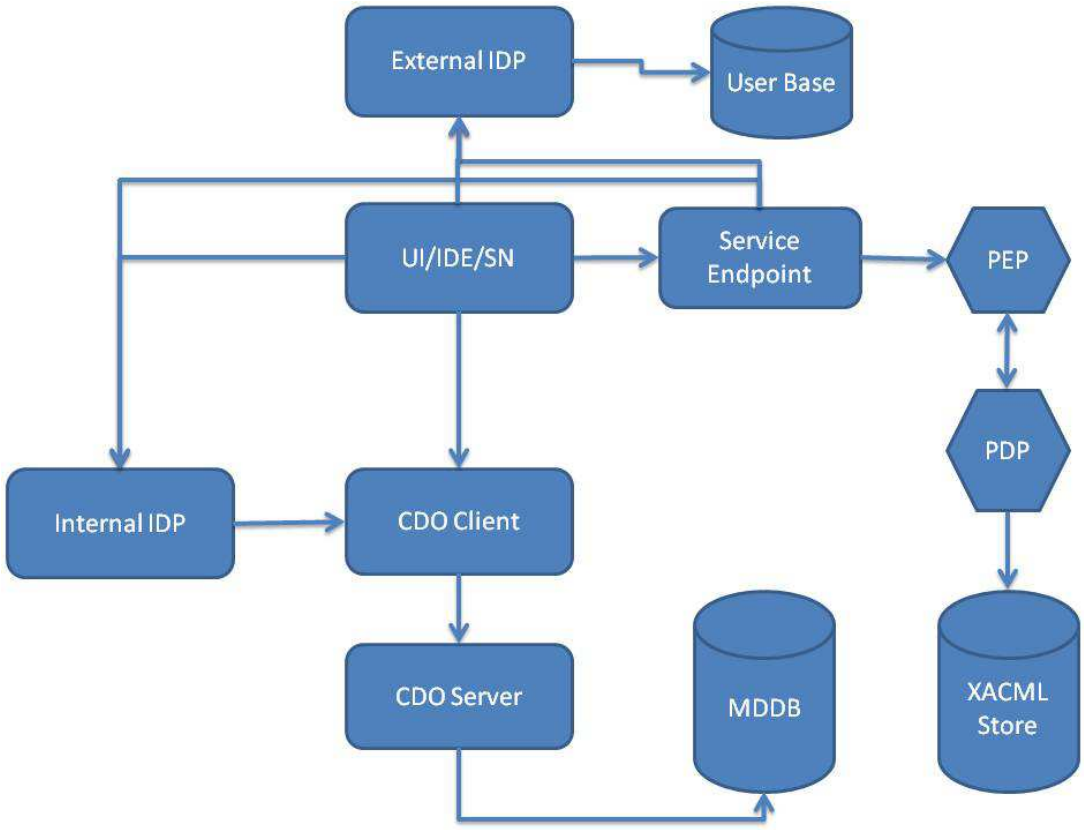

Figure 7 - Overall security architecture

The CDO Server is a component which actually encapsulates an underlying database and is responsible for directly performing respective actions on it. This component

has been regulated to include the security feature of CDO in order to work in a secure mode and allow access only to legitimate CDOClients. A legitimate CDOClient is a client which provides correct CDO credentials during session establishment. If the credentials are indeed correct, then a CDOSession with the CDOServer is established which can then be used to perform any kind of action on the CDO Repository (store, update or query models), provided that the user mapping to these credentials has the respective rights to do so. The communication between the client and the server can be secured via the use of SSL-based sessions (in contrast to normal net4j[4] ones). Such a measure is appropriate when a security component, like the IDP, is not situated inside the same VM with respect to the one hosting the CDO Repository/Server.

An Identity Provider (IDP) is a component responsible for user authentication. Two types of IDPs are envisaged: (a) external IDPs, using standards such as OAuth, (like facebook) which authenticate users and return a token encompassing a valid identifier through which the security components of our solution can exploit in order to properly identify these users and retrieve the appropriate security information related to them; (b) an internal IDP which authenticates users and returns a token containing all appropriate information that can be used to subsequently authorize users. This internal IDP operates, as indicated previously, on the security-related information in the CDO repository, while external IDPs obviously operate on their own user bases.

The Service Point (SP) is a REST service which enables accessing the services offered by the PaaSage platform allowing users to run the PaaSage workflow such that their applications are deployed, executed and adapted. This SP should be capable of retrieving user tokens, when produced in user authentication, and pass them along with the user requests to the PEP/PDP component. It should also be able to redirect users to the IDP for authentication in case of web and not programmatic access requests to platform resources.

The Policy Enforcement and Policy Decision Points (PEP/PDP) are a couple of our security solution components which are responsible for the authorisation of users by relaying to the SP the decision of whether users have the appropriate rights to access the resources requested. Such a decision obviously relies on the availability of the appropriate information that should have been passed by the SP to the PEP/PDP. It also relies on the information of XACML permissions which are stored in the XACML Policy Repository. This means that the PEP/PDP should discover those permissions that pertain to the requesting user and examine whether these permissions include the right to access the requested resource according to the access type (e.g., read/write/execute) involved.

The UIs, IDEs and the SN constitute the entry points of users in the platform which enable them to perform various tasks depending on the services offered by the PaaSage platform. However, before these tasks can be performed, the users have to be authenticated. Such an authentication can be performed either via the internal IDP or one or more external ones, where the selection of the IDP burdens the user. Upon IDP selection, the users are redirected to the respective authentication page of the IDP in order to supply their credentials. In some cases, there is no redirection but the UI/IDE/SN takes care of authenticating the users and communicating with the IDP in the background. Upon successful authentication, the authentication result, in the form of a user token, is exploited as the user authentication pointer to validate the access to the user requested tasks or information.

[4] https://wiki.eclipse.org/CDO/Net4j_Authentication

In case of information accessing requests, the UI/IDE/SN should use the CDOClient in order to authorize the access to the information requested. In case of service access requests, the UI/IDE/SN should communicate with the SP by sending the user token and access request such that the user is subsequently authorized via the PEP/PDP points. Thus, as it becomes apparent, there should be some (lightweight) integration between the UI/IDE/SN with the other security components as well as with the CDOClient (which is actually the case of the SN).

Figures 8-11 visualize the interactions between the various components in the architecture in the context of user authentication and authorization. In all figures, the interactions have numberings which reflect their order, while the prefix number indicates type of security task involved (1 - authentication, 2 - authorisation). To reduce the complexity of interactions and thus cater for clarity and better comprehensiveness, only positive/successful interactions are shown while also some obvious steps, like the return of decisions or the execution of a service, are omitted. Figures 8-9 illustrate the cases of web-based and programmatic authentication, respectively, while Figures 10-11 illustrate the authorisation of services and CDO models, respectively. In the sequel, we will attempt to analyze the interactions in the context of the security cases on which these figures map. It must be highlighted that the above authentication and authorisation cases can be obviously mixed, as we also need both security tasks, and this leads to four main security scenarios: (a) web-based authentication and CDO authorisation; (b) web-based authentication and service authorisation; (c) programmatic authentication and CDO authorisation; (d) programmatic authentication and service authorisation. This means that we are actually able to cater for different security cases and scenarios and this enhances the added-value of our solution.

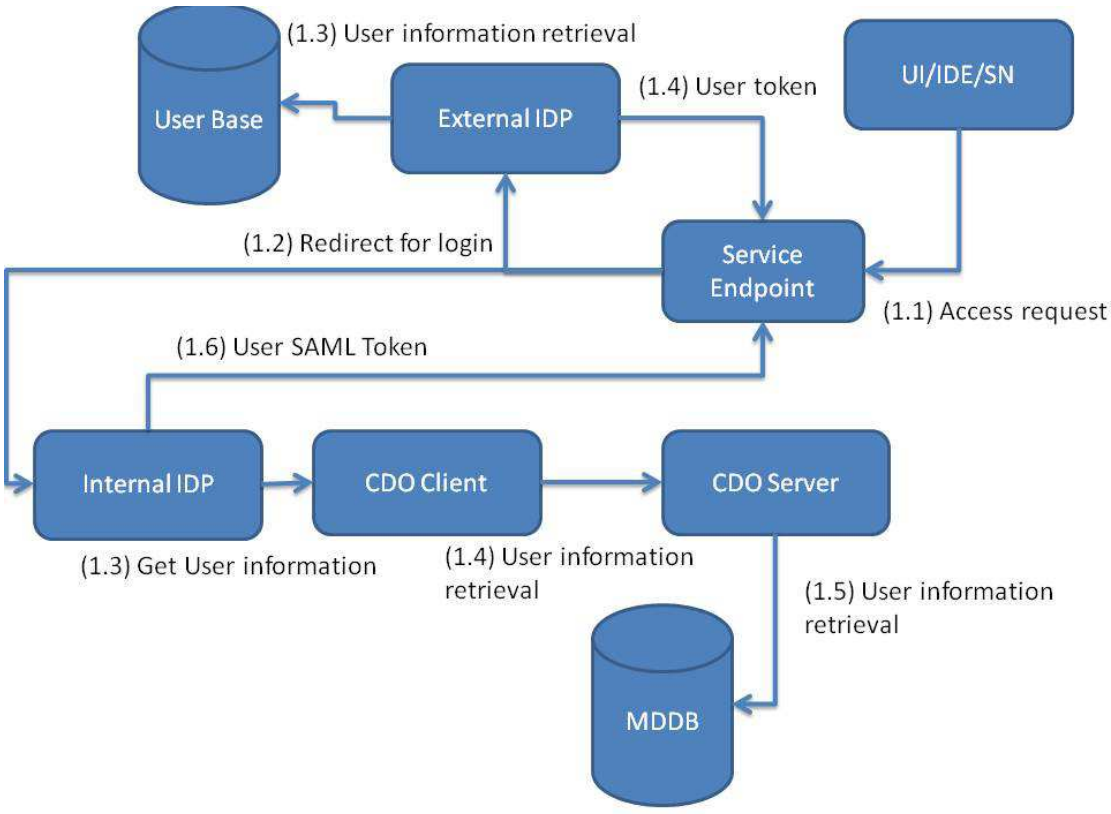

Figure 8 - Web-based authentication interactions

During a web-based authentication, the UI/IDE/SN communicates with the SP which redirects the user, after selecting the appropriate IDP, to the authentication page of this IDP. Then, the user provides his/her credentials which are checked against the user base of the IDP, which can be any type of user base or a CDO repository, depending on the IDP type. If authentication is unsuccessful, then obviously the IDP returns an error message to the user.

In case of the internal IDP, the checking is performed on the CDO repository and this involves the use of a CDOClient. The CDOClient is initiated with the user credentials and a session is attempted to be established with the CDOServer. Upon successful session establishment (thus mapping to a successful authentication), a subsequent query is submitted to obtain the additional user information that will be used for the authorisation which is exploited to construct the user token. This token is then returned to the UI/IDE/SN, i.e., the component which initiated the authentication.

In case of an external IDP, the checking is performed against the own user base. Upon successful authentication, a user token is constructed, containing a particular identifier of the user, and returned to the UI/IDE/SN.

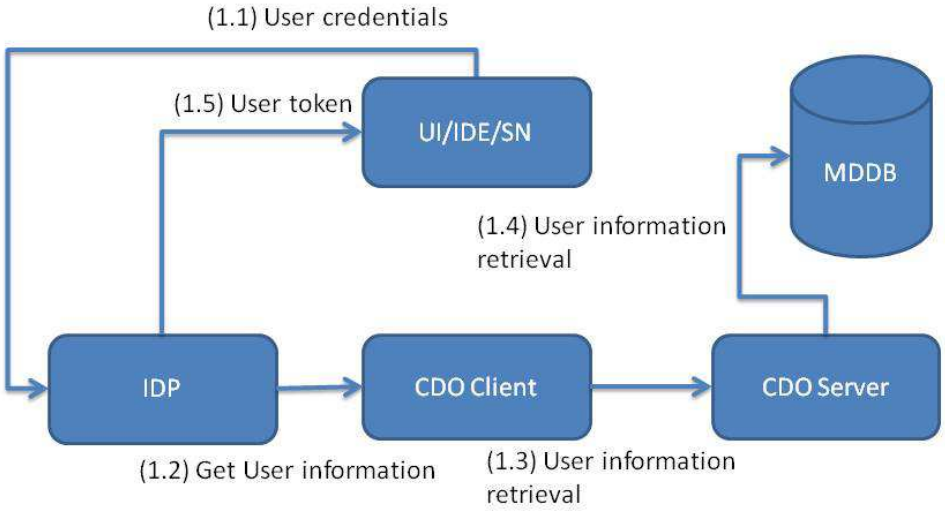

Figure 9 - Programmatic authentication interactions

The main difference between the programmatic and the web-based authentication lies on the fact that for the former there is no redirection as the initiating component has to have the control of the security workflow, passing in this way from all suitable component of the solution architecture. This means that the initiating component should obtain the user credentials and contact the internal IDP to validate them. The selection of the internal IDP lies on the fact that we are certain that it supports programmatic authentication. The checking of user credentials then proceeds as in the web-based authentication with the sole exception that now the error message should be relayed to the user by the initiating component.

CDO-based authorisation involves a quite simple interaction process. The initiating component exploits the CDOClient to establish a session with the CDOServer and issue the user request. To this end, the CDOClient is initiated through the user token, attempts to obtain the user credentials through it and then these credentials are used to establish a session with the CDOServer. Upon successful session establishment, the CDOClient issues the user request which is checked by the CDOServer with respect to the CDO permissions that pertain to the respective user. If the user is allowed to

have access to the respective information resource, then the user request is executed against the underlying store and the respective result is returned. Otherwise, an error message is sent back. In all cases, the authorisation and probable request execution result is sent back to the user by the authorisation-initiating component which controls the CDOClient.

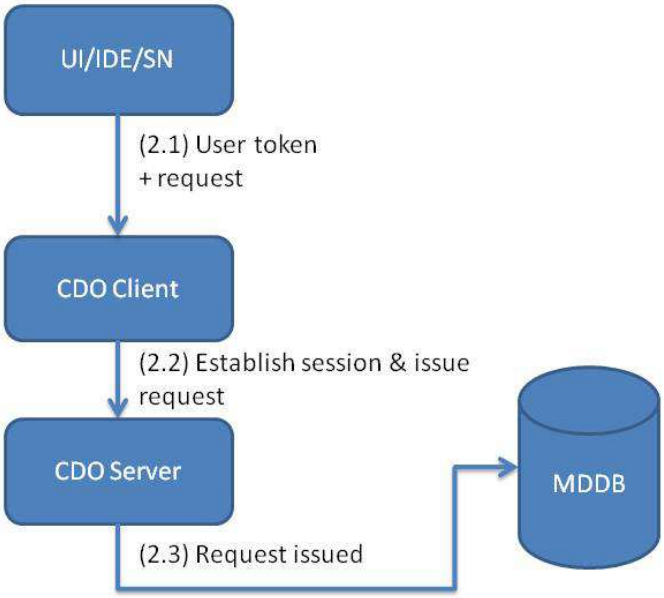

Figure 10 - CDO authorisation interactions

Service-based authorisation starts with the authorisation-initiating component sending the user token and request to the SP. This information is redirected by the SP to the PEP which redirects it in turn to the PDP. The PDP then, either checks the authorisation information immediately or communicates with the CDOServer in case the user token does not contain all necessary user information (actually mapping to the case of a previous external authentication). In the latter case, the user token and respective user identifier is exploited to obtain the necessary user information mapping mainly to the roles assigned to this user from the CDO repository. In all cases, the authorisation checking is performed by examining the policies in the XACML store to discover those associated to the user roles. The latter policies are then checked to see whether they allow the user to perform the requested action (service call). Upon successful checking, an access grant decision is issued; otherwise, an access denied decision is issued. The decision result is finally propagated to the PEP and then to the SP which tries to respect it. The latter means that the SP will either allow the call to the service or send an error message to the authorisation-initiating component, depending on the authorisation outcome.

4.2.1 Information Integration Implications

While the security solution architecturally presented looks quite clear and precise and involves the catering of different security scenarios, it does come with particular issues that had to be dealt with. In particular, due to the need to have a single CDO repository, exploit the CAMEL organisation meta-model for expressing security-oriented information, utilize the CDO security feature as well as manipulate XACML policies, we had to perform two main information integration tasks:

1. As CDO security relies on a particular CDO security meta-model, we needed to integrate the information specified through this meta-model and the CAMEL organisation one.
2. As XACML expresses policies and so does the CAMEL organisation meta-model, we also had to integrate models of XACML with organisation ones.

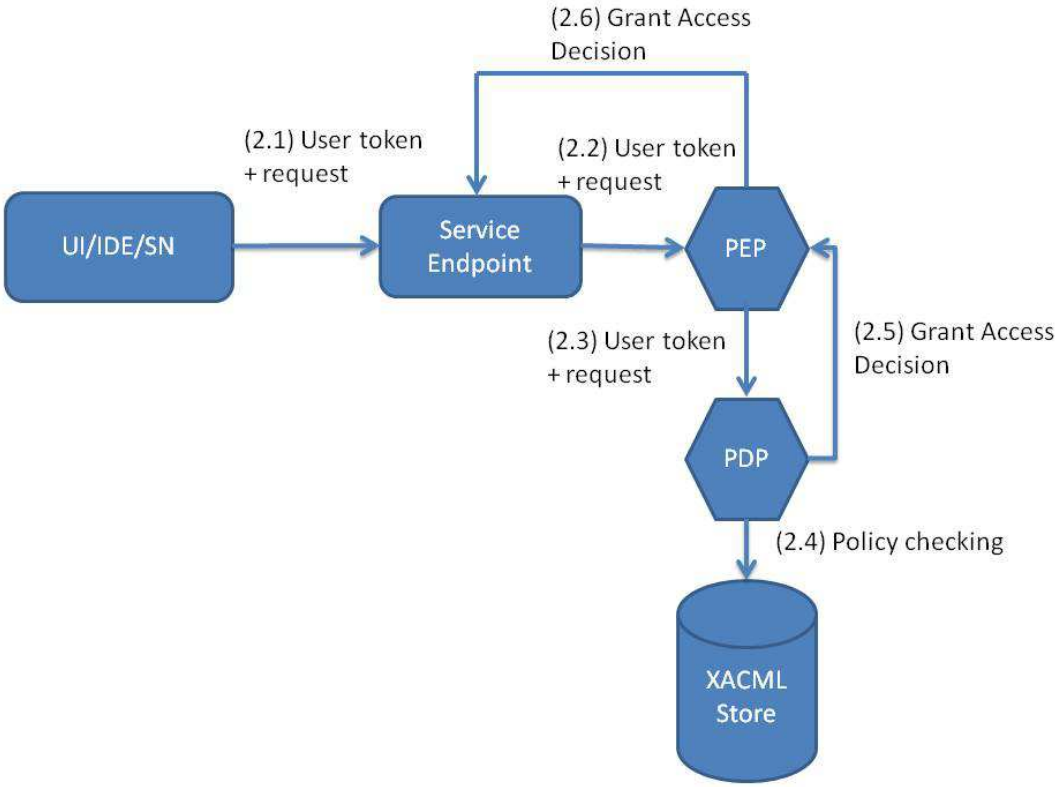

Figure 11 - Service authorisation interactions

For both tasks, we have followed a rather lightweight integration approach relying on the following two principles: (a) the meta-models remain independent and not truly integrated; (b) information integration is performed at the model level through transformations. This approach was followed for the following reasons: (a) CDO authentication and authorisation relies on the information specified by the CDO security meta-model; (b) service authorisation relies on XACML model checking; (c) the use of a CAMEL model mapped into information pertaining to the CDO security meta-model and the XACML language hides any complexities inherent in the latter two meta-models/languages and enables users to deal with specifying one model mapping to only one non-complex language, thus not requiring from them to be acquainted with many languages each mapping to a different security case/scenario.

We should note here that the approach followed should not increase the risk of having models which are not correctly updated (e.g., updates on a CAMEL organisation model are not propagated to the CDO security one). To this end, we have provided an administration API which, by relying on the CAMEL organisation meta-model as the one whose models are being updated, guarantees that the propagation of updates is performed for the models of the CDO security and XACML languages. Such an API supports not only the updating of whole models but also their constituting parts, such as information about users, roles and role assignments, while also enables the initial

storage of organisation models to the CDO repository. As such, this API really serves as a companion to administrators greatly assisting in supporting administrative tasks related to security information manipulation.

Based on the above rationale, we now start analyzing the way information integration has been achieved for the two tasks previously identified.

4.2.1.1 Organisation Meta-Model to CDO Security Meta-Model Mapping

The CDO security meta-model includes concepts which have a more or less one-to-one mapping to the organisation meta-model which greatly facilitates our mapping/integration task. The mapping between the CDO security and CAMEL organisation meta-models is visualized in Figure 12.

As it can be seen from Figure 12, there are concepts which have exactly a one-to-one association, like users and roles. On the other hand, there are also concepts with a more involved mapping, mainly related to the specification of permissions. The more involved mapping in this case relies on the fact that the CDO security model is quite rich as it is able to specify logical combinations of resource filters on permissions. On the other hand, the CAMEL organisation model was designed with simplicity as its main design goal. As such, it allows only simple resource filters to be specified for each permission. However, these resource filters are as expressive as needed as they can map to filters on concrete CDO resources or on the content (folders and resources) of CDO resource folders. In this way, we believe that we actually cater for the most usual cases with respect to the description of information resources and there should not be any exceptional case that might be needed to be expressed. Apart from information resource filters, also service filters can be expressed in permissions but this will be explained in more detail in Section 4.2.1.2. We should also highlight the non-obvious (and possibly irrational for some) mapping between a CAMEL organisation and a user group. As will be indicated afterwards, this mapping is required in order to have a way to refer to all users of a particular organisation.

Before entering details about the mapping procedure, we need to explicate the following. We envisage that three main action types can be used in permissions, depending also on the types of resources to be involved. In particular, read and write action types should be used in permissions on information resources, while the access action type can be used in permissions on service resources. Moreover, we consider that in each organisation, the following three main roles will be involved:

- *administrator*: this role can update the organisation model of the current organisation at hand. All other roles are not able even to read this model.
- *business*: this role maps to a business-oriented type of user. Such a role should be able to have a complete high-level business view on what is happening in its organisation in terms of the PaaSage platform (e.g., to check the current cost of deployments, to see the application performance and so on) as well as pose high-level requirements (e.g., generic security requirements as well as cost requirements) which can drive the organisation applications deployment.
- *devops*: this is the main role of an organisation which is actively involved in the management of the organisation's applications. In this respect, the duty of this role is to specify applications and all appropriate technical details supporting their management, spanning almost all types of models apart from the organisational ones, as well as initiate the deployment of the applications in a multi-cloud environment.

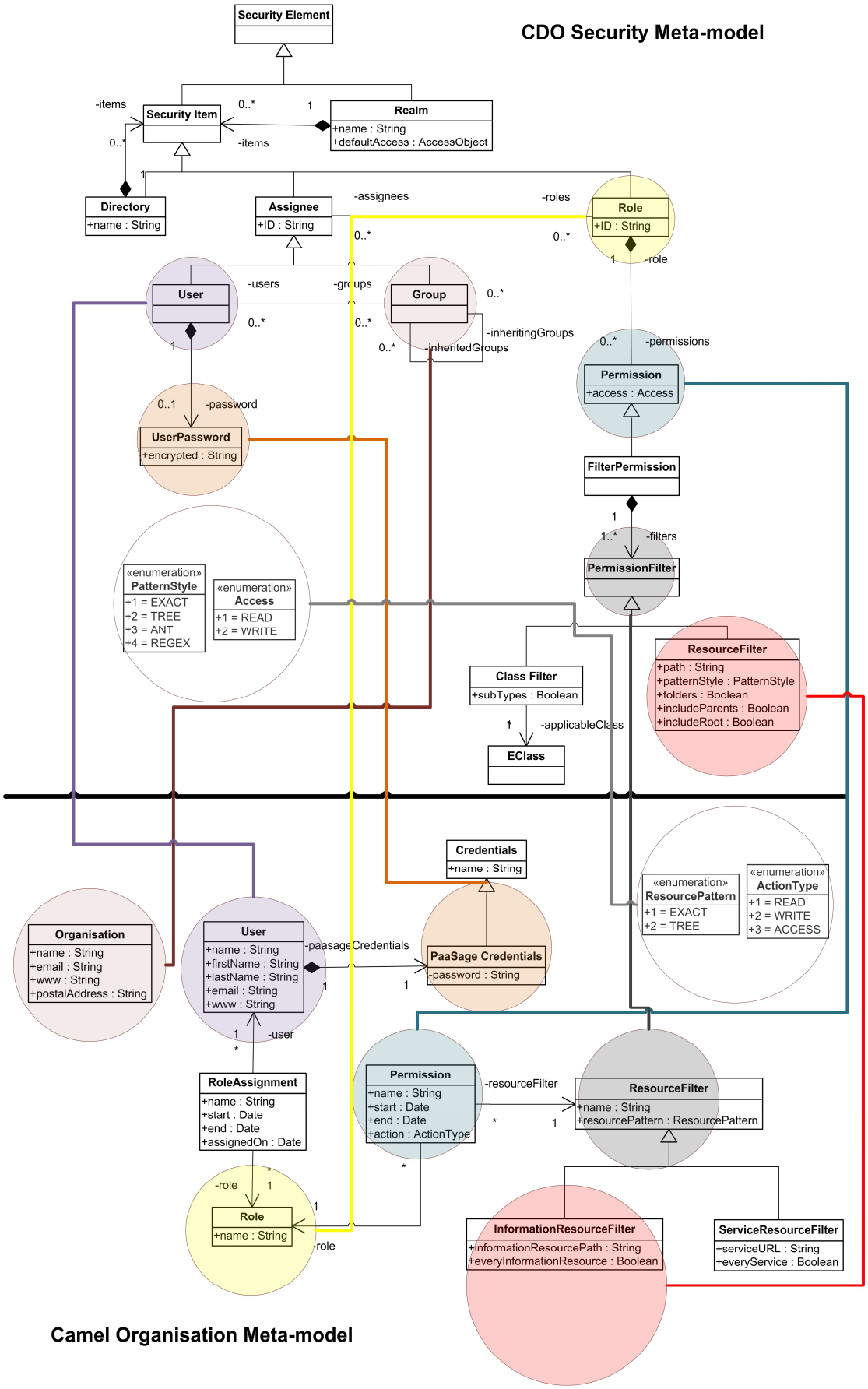

Figure 12 - The mapping from CAMEL organisation to CDO security meta-model

We should mention here that the above mapping of roles to the corresponding permissions/rights is performed by default. Of course, an organisation can modify these default rights of its users/roles based on its requirements as well as include the description of new roles. As such, what we actually propose here is a basic set of security permissions per role that might reflect the appropriate duties and rights of such a role in any organisation. Thus, we actually capture the generic case and leave exceptions to be handled by the organisations themselves through the appropriate modification of these permissions.

We need to also stress here that the CDO repository is initialized with some basic models (as indicated in the previous section) whose access must also be controlled. The super administrator[5] of the platform will of course have full access to these models in order to be able to modify them, when the need arises (e.g., consideration of a new cloud provider or update of cloud provider offerings). In the case of critical information (with respect to the way this information is exploited by the platform), only this role will only have write access to this information as this should be the sole role to trust in this case for the whole running PaaSage platform instance. As the information criticality becomes lower, then the respective rights of particular types of basic roles can be modified (to write access) to reflect this. This actually reflects that there should be some kind of cooperation and trust between the organisations participating in the PaaSage platform in order to increase the quality and extend of the information specified in the basic models. Based on this analysis, we foresee that the following rights and roles are relevant for the basic models of the platform:

- *location model*: only *devop* users should be involved and enabled to modify geographical/physical locations but just see cloud locations.
- *metric model*: *devop* users can modify the information stored in the metric model. The *business* users must be able to read this model as they might need it in order to express requirements.
- *security model*: devop users will be able to extend the basic security metric model but they will not be able to modify security controls and capabilities. *Business* users will be able to see everything.
- *type model*: the level of criticality here is quite low so we expect that *devop* users will be able to write new types into this model, as required.
- *unit model*: the *devop* users will be able to write new units, not originally covered by the basic unit model.
- *temporary model folder*: the level of criticality here is the lowest possible. We expect all types of users for all organisations will be able to write models here.

We need to stress here a special role that must be specified to reflect what all other users of other organisations in a running PaaSage platform instance can see or do with respect to the information owned by a current organisation. We call such a role *external* and we emphasize that the set of permissions attributed to this role can vary depending on each organisation needs. For instance, an organisation might not want to share anything with any other organisation. In this case, this will be reflected on the

[5] There are other responsibilities related to the running PaaSage platform instance, such as bug fixing and platform performance monitoring which could be undertaken by this role or additional roles could be involved, pertaining to the organisation that runs the platform.

set of permissions to be specified for the *external* role. In the general/average case, we expect that an *external* role will be able to see an organisation's applications, deployment and execution models and possibly requirement, metric & scalability models. In any case, such a role must not be able to write anything or see any details of the respective organisation model. Based on these two cases, we enable an organisation to choose one of the two options for the permissions of the *external* role, which are named as "totally_private" and "totally_visible", respectively. Then, depending on its needs, an organisation might modify them but this will certainly avoid write everything from scratch.

However, the existence of such a role as well as the customisation of permissions for the three basic roles creates a major issue that must be handled appropriately. In particular, we cannot have basic roles or an external role which can generically be applied for every organisation. On the contrary, all roles should be quite concrete and pertain to only one organisation. As such, the mapping between an organisation model and the CDO security model should be carefully adapted to cater for this restriction which maps to the case that a generic role in an organisation model will map to an organisation specific role in the CDO security model.

Before the mapping procedure is executed, we pre-process the CAMEL organisation model to enrich it based on the basic-role-to-default-permission-mappings and the security level required for external users. Depending on the security level desired, an external role is created for the organisation named as "external" which is also mapped to a set of permissions.

The transformation attempts to enforce the more or less one-to-one mappings illustrated in Figure 12 by processing an organisation model and its security-oriented information in particular. We should highlight, before entering details about this transformation, that a secured CDO repository comprises only one security model created and stored, upon the repository initialisation, in a CDO resource named as "security" at the root level. This means that each organisation model should be mapped to one and only CDO security model. In the context of the mapping procedure, the transformation involves the following set of mappings:

- An organisation is mapped to a CDO user group named after its name. In this way, we are able to connect all organisation users to a particular group. This not only provides the ability to create correct user partitions but also enables associating user groups to any kind of permissions that might hold for all users of these groups through directly associating the group to the particular role associated to these permissions. In our case, a created CDO user group is associated to all external roles for all organisations that have been previously accounted for (and thus their organisation model processed). In this respect, the users of the group will have those rights to access external resources as dictated by the respective permissions attributed to these roles.
- Each user is mapped to a CDO user with the same (PaaSage platform) credentials as those that have specified for him/her in the organisation model, where the user name maps to the CDO user id and the password to the CDO *UserPassword*. Thus, we have a one-to-one correspondence between CAMEL and CDO users but we have preferred to transform only a very small information from a CAMEL user to its equivalent CDO part (CDO User) in order to be focused, i.e., use the organisation model for any kind of information related directly to a user and the CDO security model only for information directly pertaining to user authentication (and thus only

credentials). Each CDO user is associated to the user group of the current organisation.

- Each role is mapped to a CDO Role, where the id of the CDO Role is constructed from the names of the CAMEL role and respective organisation based on the following pattern: "<camel_role_name>_<organisation_name>". The external role is associated then to all user groups that have been previously defined and stored apart from the current one. In this way, we are able to indicate the exact permissions that should apply to all users of all external organisations with respect to the resources owned by the current organisation.
- Each permission is checked for the type of actions involved. If the action is read/write, then we have the case of a model resource. Otherwise, we have the case of a web page or service resource. For a model-based resource, we have to create the respective policy/permission in the CDO security model. Otherwise, we neglect the permission as it will be actually exploited for the mapping to XACML policies analyzed later on. So, let us focus on the mapping of permissions on model-based resources. First, the role involved is checked such that undefined roles are specified in the CDO model via role mapping (see previous bullet). Then, the actions and the actual resource of the CAMEL permission are mapped. We initially create a CDO Permission with the *access* attribute having a value that maps to the type of action involved in the CAMEL permission. Next, depending on the resource description, we create the respective CDO filter (permission core). A resource description can indicate either (a) a name of a CAMEL class or (b) the name/id of a specific resource or a path in the CDO Repository. In case of a CAMEL class, the permission should hold for all the instances of this class. As such, we create a Class filter as the core of the CDO permission with reference to the desired CAMEL class. In case a CDO path is provided, we create a ResourceFilter as the core of the CDO permission and regulate the *pattern* style as well as other boolean attributes to reflect on whether we are dealing with a CDO resource or path (where in the latter case, all resources and folders contained in this path are affected by the permission). In the end, one CDO permission will be created which will be associated to the identified CDO role.
- Each role assignment in CAMEL maps to creating an association from a user to the respective role assigned to it in the CDO security model.

In order to enable the reader to better understand the above mapping algorithm, let us provide a specific scenario. Suppose that we have organisation A with 3 users U1, U2 and U3 and the aforementioned four basic roles (including the external one). Further suppose that there is another organisation B with again with three users V1, V2 and V3 and the four roles. The mapping algorithm will start processing the organisation model of A. It will first create one user group named as "A". Then, it will create the four required roles which will be named as "administrator_a", "devops_a", "business_a" and "external_a". As no other organisation has been created before, the external role is not associated to any other user group. Then, we process the organisation's permissions and we create the respective CDO permissions that are mapped to a particular role. In the end, we create the users of the organisation which should be included in the organisation's user group and will be mapped to one of the above three basic roles, while we also enforce the respective role assignments. Next,

the processing of the organisation model of B can start which should lead to similar transformation results. However, there is a specific exception in this case: (a) the created user group of B called "B" should be associated to the external role of A named as "external_a" while when processing the roles of the organisation model of B, the user group of A should be associated to the external role of B named as "external_b". Thus, we can follow the processing of forthcoming organisation models in a similar fashion and associated all previous external roles to the new organisation model's user group as well as update past user groups with the association of the new organisation model's external role.

Before processing and storing any organisation model, we need to stress the necessity of introducing special roles which can be loaded from the very beginning once the initialisation of the CDO repository has finished. As such, a particular code is required which will update the CDO security model to include the description of three main users and respective roles: (a) a user/role mapping to the internal IDP which should have read access only to user-related information - write access is not needed in this case as the IDP will not be responsible for creating new users; (b) a user/role mapping to the PEP/PDP components which should have read access to some user-related information (for exploiting tokens created by external IDPs - mainly roles assigned to a user will be retrieved); (c) an administrator user/role which as indicated will have full access to the CDO repository and be responsible for updating basic models when the need arises.

4.2.1.2 Organisation Meta-Model to XACML Mapping

The mapping between the organisation meta-model and the XACML XML Schema is depicted in Figure 13. As it can be seen, the mapping is not obvious and there is no direct one-to-one mapping between elements but actually there are two two-to-one mappings in both directions.

In particular, a *Permission* in the organisation meta-model maps to a policy and a rule element in XACML meta-model/schema. This follows the semantics that one permission maps to one policy in XACML which comprises just one rule with a *PERMIT* effect. This is also related to the fact that just one condition/filter in the organisation meta-model is mapped to a permission and not many, indicating that obviously the XACML meta-model is richer in this aspect. We should highlight here that as we have opted for allowing the modelling of only positive permissions in CAMEL, the effect of a rule will always be *PERMIT*.

The Target of a policy rule in XACML should map to three combinations of *AnyOf-AllOf-Match* elements, where each combination maps to a different element in a CAMEL permission: (a) the role affected; (b) the URL of the service to which access should be granted (taken from the respective *ServiceResourceFilter* instance); (c) the action allowed. In this way, we cover all the required information in an organisation meta-model with respect to a specific permission for a service-based resource.

Thus, as it can be easily understood, the mapping is not very extensive. However, there is additional information to be specified for a XACML model when being produced from a certain organisation model spanning the following elements: (a) *PolicySet*, and (b) *PolicyCombiningAlgorithm*. This is related to the fact that as an organisation model comprises many permissions and not just one, this means that a policy set should actually be created with as many policies as the number of permissions which should be related to a specific algorithm combining the

encompassing policies. By taking this into account, we will then be able to map one organisation model to one XACML policy file which would simplify a lot the XACML permission management in contrast to the case where multiple XACML policy files are created (containing a policy mapped to a CAMEL permission) for one organisation model.

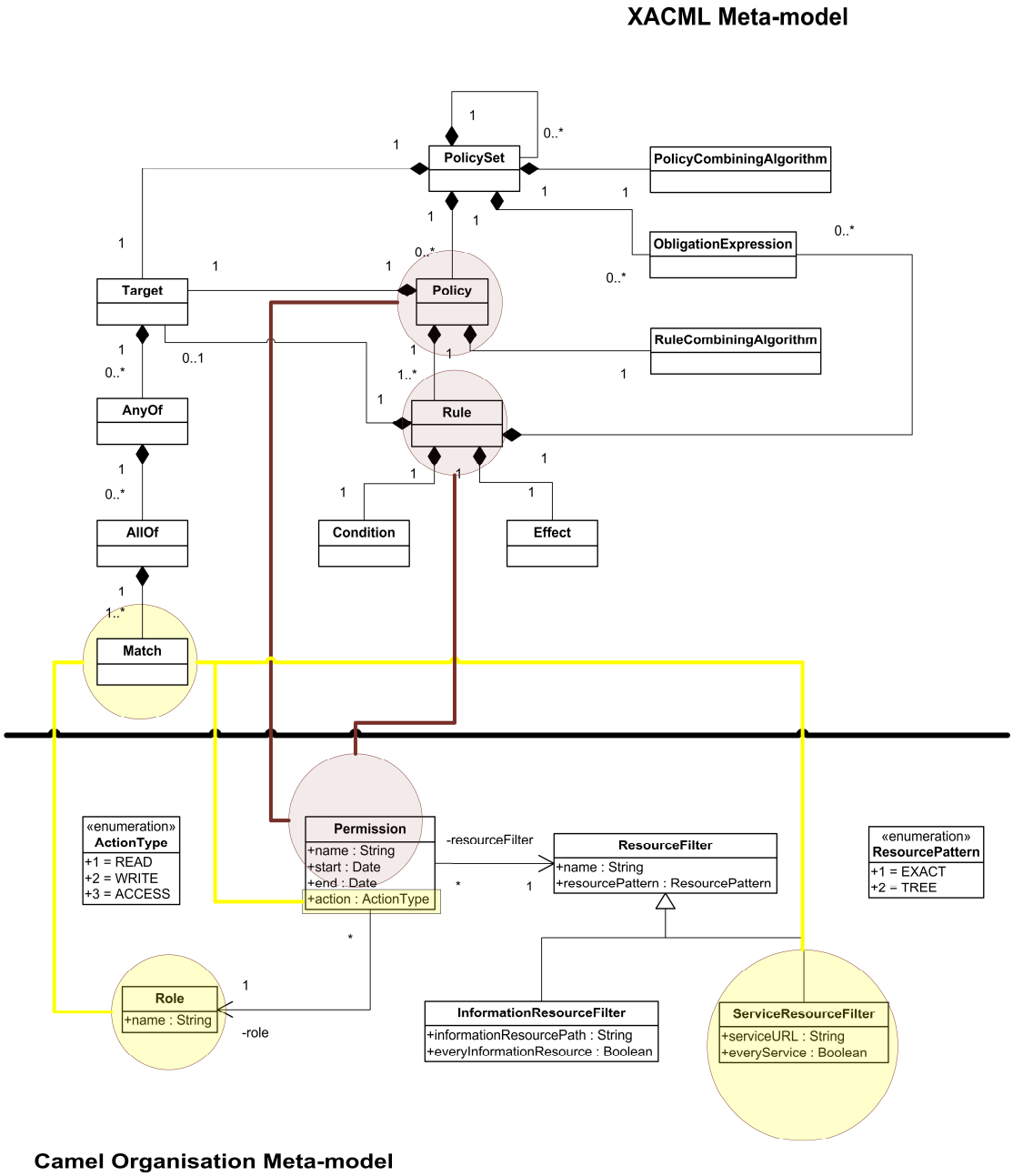

Figure 13 - The CAMEL-to-XACML Mapping

Based on the above analysis, the CAMEL-to-XACML mapping algorithm proceeds as follows:

- One policy set is created with id named after the name of the organisation as follows: "urn:oasis:names:tc:xacml:3.0:camel:policysetid:" + <orgName>. This policy set is associated to the "first-applicable" policy combining algorithm which indicates that the first applicable policy is checked and its result is considered globally for the whole policy set. We could choose

another algorithm alternative, such as "only-one-applicable" but in this way we enforce a particular constraint on the way permission/policies can be expressed by the modeller. In particular, we indicate that for each user context, only one policy will always apply. As we are dealing with positive policies and we do not want to impose particular constraints to the modeller in the way he/she can express policies, we do not resort to this alternative in the end.

- For each permission in the organisation model, we create one policy and one rule which are named as follows: "urn:oasis:names:tc:xacml:3.0:camel:policyid:" + <permName> for the policy and " urn:oasis:names:tc:xacml:3.0:camel:ruleid:" + <resourceFilterName>, where <permName> is the name of the permission and <resourceFilterName> is the name of the (service) resource filter associated to the permission. The policy generated, similarly to the case of the policy set, is associated to the "first-applicable" rule combining algorithm. The policy rule is associated to just one target element which will contain three *AnyOf* elements. Each *AnyOf* element will contain one *AllOf* element which will contain in turn one *Match* element. The first *Match* will apply to the role associated to the permission, the second *Match* will apply to the service URL of the service resource filter associated to the permission and the third *Match* will apply to the action allowed by the CAMEL permission.

In order to better understand the above simple mapping algorithm, we now provide a specific example of its application. Consider a particular organisation model for the AGH organisation which contains one permission indicating that a *devops* user can access the following service URL: http://www.paasage.eu/sp/runPaaSageWorkflow. The following textual model designates the description for this organisation in CAMEL:

```
camel model ScalarmModel {
 organisation model AGHOrganisation {
  organisation AGH {
    www: 'http://www.agh.edu.pl/en/'
    postal address: 'al. Mickiewicza 30, 30-059 Krakow, Poland'
    email: 'morzech@agh.edu.pl'
  }

  user morzech {
   first name: Michal
   last name: Orzechowski
   email: 'morzech@agh.edu.pl'
   paasage credentials 'morzech_at_agh_dot_edu_dot_pl'
  }

  role devops;

  role assignment{
   role: ScalarmModel.AGHOrganisation.devops
   user: ScalarmModel.AGHOrganisation.morzech
   assigned on: '2015-09-30'
   end: '2016-09-30'
  }
```

```
  permission AGH1{
   role: ScalarmModel.AGHOrganisation.devops
   action: ACCESS
   end: '2016-09-30'
   service resource filter Filter1{
    resource pattern: EXACT
    service url: http://www.paasage.eu/sp/startPaaSageWorkflow
    any service: false
   }

   security level: HIGH
 }
}
```

By following the mapping algorithm analyzed above, the respective XACML policy file that will be generated will be the following:

```
<?xml version="1.0" encoding="UTF-8"?>
 <PolicySet
  xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  PolicySetId="urn:oasis:names:tc:xacml:3.0:example:policysetid:AGH"
  Version="1.0"
  PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-
combining-algorithm:first-applicable">
  <Description/>
  <Policy
PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:AGH1"
   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:deny-overrides"
   Version="1.0">
   <Target/>
   <Rule RuleId="
        urn:oasis:names:tc:xacml:3.0:example:ruleid:Filter1"
    Effect="Permit">
    <Description/>
    <Target>
     <AnyOf>
      <AllOf>
       <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string"
               >devops</AttributeValue>
        <AttributeDesignator MustBePresent="False"
         Category="urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject"
         AttributeId="urn:oasis:names:tc:xacml:3.0:example:
attribute:role"
         DataType="http://www.w3.org/2001/XMLSchema#string"/>
       </Match>
      </AllOf>
     </AnyOf>
     <AnyOf>
      <AllOf>
       <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-
equal">
        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#anyURI"
```

```
        >http://www.paasage.eu/SP/startPaaSageWorkflow</AttributeValu
e>
         <AttributeDesignator MustBePresent="False"
          Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:resource"
          AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-
namespace"
          DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
        </Match>
       </AllOf>
      </AnyOf>
      <AnyOf>
       <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
         <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string"
                >access</AttributeValue>
         <AttributeDesignator MustBePresent="False"
          Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:action"
          AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Match>
       </AllOf>
      </AnyOf>
     </Target>
    </Rule>
   </Policy>
</PolicySet>
```

4.2.2 Administration API

In order to enable the automatic mapping between CAMEL models and CDO security and XACML models as well as ease the administrators with the management of security-related information, an Administration API has been developed. This API exposes a functionality which spans the following aspects:

- *Importing of organisation models in CAMEL form* - these models are then mapped to CDO security and XACML models depending also on their content (e.g., if no service resource policies are contained, then obviously no XACML model will be produced and stored in the XACML Store).
- *Adding roles* - in this respect, the CAMEL organisation and the CDO security model are only updated to include the new roles based also on the mapping algorithm aforementioned.
- *Adding users* - both the CAMEL organisation and CDO security model are updated to include the new users; the mapping of CAMEL to CDO users is as indicated in the mapping algorithm.
- *Updating user information* - here the CAMEL organisation model is surely updated; if the update corresponds to changing the username/logic or password of the user, then obviously the CDO security model has also to be updated as this represents the information that is stored for users in the CDO security model.
- *Removing users* - here the CAMEL organisation and CDO security model are affected as the respective users are deleted. In this functionality as well as in

the previous one, we need to stress our ability to easily discover users as they are identified by their username (login).

- *Adding role assignments* - the CAMEL organisation and CDO security models are updated with the addition of the role assignment and the respective association from a user to a role, respectively.
- *Updating role assignments* - updating a role assignment corresponds to different cases depending on what is being changed; if just attribute-based information is modified, the CDO security model does not need to be modified; otherwise, we have updates on both the CAMEL organisation and CDO security model.
- *Removing role assignments* - here we have an update of the CAMEL organisation model through the direct removal of the role assignment and a modification of the security model through the removal of the role from the roles association of the user. In this and previous functionality, we need to stress again, based on the naming pattern adopted for roles, that it is quite easy to discover those roles that need to be removed/added from the respective association of a CDO user.
- *Adding permissions* - the CAMEL organisation model is updated and depending on the type of permission, either the CDO security model is updated or a new XACML model is produced.
- *Updating permissions* - apart from the CAMEL organisation model, we can have also updates to either the CDO security or a XACML model. It is important to stress here that there must be a means to identify in a XACML store which is the XACML model that is being affected. This can be done through querying the store to discover the required model. Moreover, we have also discovered the means of identifying the affected permission in the CDO security model.
- *Removing permissions* - the CAMEL organisation model is updated and either the CDO security model is modified or a XACML model is deleted.

It must be highlighted here that it would be ideal if the developed API is coupled with an administration UI. This will cater for administrators who prefer to interactively manage security information with respect to those that prefer a code-based approach. The existence of such a UI (along with the API of course) would relieve administrators from having to learn the CDO security meta-model or the XACML meta-model. They might need to know how to express CAMEL organisation models but this could be delegated to a CAMEL expert in the same organisation or such an organisation model could be completed in a form-based way without requiring the knowledge of CAMEL. As such, as the knowledge of CAMEL will not be required, administrators could just use the UI to update an existing organisation model by providing common information which is usually part of common administration APIs and UIs.

4.3 Implementation Details

The internal Identity Provider (IDP) in PaaSage provides means for delegated authentication and authorization. Several existing technologies have been evaluated

for this purpose including ZXID[6], WSO2[7], simpleSAMLphp[8]. The main criteria were that the tool should have been open-source, simple in usage, and easy to extend. In the end, the simpleSAMLphp solution was chosen and extended[9] to support the ECP (Enhanced Client Profile) profile which enables programmatic authentication for users. The selected IDP solution has been integrated with the CDOClient, in order to provide means for user authentication based on respective CDO Repository content (users and roles in organisation models). The integration has been achieved by implementing a custom backend plugin, PaaSageIdp, for simpleSamlPHP using PHP/Java-Bridge technology[10]. The plugin is implemented in PHP, and works with both browser based as well as ECP mode authentication.

The PEP/PDP point was realized via the OpenAZ [11] library. This library was developed as part of the Kantara initiative to provide policy based access to resources. It has been used to enable the creation and enforcement of XACML policies within the software implementation.

Implementation details about the SP (PaaSage REST service) can be found in D5.1 2(!), while such details about the CDOClient and Server have already been provided in Section 3.4.

The administration API exploits the CDOClient to store and update the CAMEL organisation and CDO security models. The CAMEL-to-XACML transformation/mapping code is under implementation based on the OpenAZ library useful for loading XACML models in memory and updating them and on the CDOClient library. To cover this implementation gap, a specific XACML editor has been generated based on PhP and MySQL as a modification of the policy editor developed by the PaaSage project partner CETIC for the Framework 7 project PONTE (Efficient Patient Recruitment for InnOvative CliNical Trials of Existing Drugs to other Indications)[12]. The policies created through this editor enable the generation of associations between users, roles and specific actions to specific resources. In this respect, an organisation administrator can use this editor in order to create and store XACML policies for its organisation which can then be exploited in order to perform user authorisation for service-based resources. These policies could eventually be expressed via the CAMEL tree or textual editor and transformed into XACML once the respective transformation code is in place. However, the XACML editor could still be exploited, once the XACML-to-CAMEL transformation direction is realised, in order to cater for scenarios where users edit XACML models (if XACML is more familiar to them) which are then used to inform the respective CAMEL organisation models.

4.4 Demonstration Scenarios

Two demonstration scenarios about programmatic and web-based authentication as well as information and service-based authorisation will be provided:

---

[6] http://www.zxid.org/html/zxid-idp.html

[7] http://wso2.com/products/identity-server/

[8] https://simplesamlphp.org/samlidp

[9] https://github.com/bkryza/simplesamlphp

[10] http://php-java-bridge.sourceforge.net/pjb/

[11] http://www.openliberty.org/wiki/index.php/OpenAz_Main_Page

[12] http://www.ponte-project.eu/

(1) programmatic authentication and information-based authorisation - a particular IDE or programmatic component will be involved in this scenario which programmatically authenticates the user and then uses the CDOClient to have controlled access to the information stored in the CDO repository

(2) web-based authentication and service-based authorisation - a client will be involved in this scenario which uses a UI or the SN to have access to the platform REST service methods. The UI/SN redirects the user to the IDP and then sends the user token and request to the SP which contacts the PEP/PDP point to check whether the user request is legitimate.

5 Knowledge Base

5.1 Introduction

Based on the findings of the evaluation conducted in the context of the previous main WP4 deliverable, D4.1.1, which demanded the quite compact representation of facts in order to reduce the communication burden between KB clients and the KB service, as well as the move from a pure relational realization of the MDDB to a CDO one, it was decided that the Knowledge Base had to be re-engineered at two main levels: the code level as well as the domain model level. However, as the CAMEL meta-model has evolved and still continues to evolve over time, the re-engineering was expanded also at the rule level.

5.2 Re-Engineering Efforts

5.2.1 Code Level Re-Engineering

The re-engineering at the code level concentrated on how the KB can exploit the CDO facilities in order to obtain information from the MDDB. Fortunately, the mechanisms for imposing this according to the realisation solution of the KB (Drools Expert) were quite generic and enabled us to move from hibernate sessions to CDO sessions and respective client. This was possible as the KB allows the passing of global parameters in the evaluation of rules which can be arbitrary objects so this means that any object able to connect to any type of database could be in use here. As such, now CDO queries could be issued through a global CDOClient and the respective information from the MDDB could be statically retrieved for the evaluation of rules.

We need to stress here the static nature of the queries - the KB evaluates all queries in rules in the very beginning and does not allow the re-evaluation of queries in the context of the same knowledge session, unless some tricks can be performed at the rule level. Such tricks, fortunately, were not required in the current set of rules that were designed. The focus is now on rules which are evaluated at once, especially as the content of MDDB evolves so previously derived facts can be invalidated, thus requiring the re-evaluation of rules in order to produce new and up-to-date knowledge.

Apart from this significant modification, it was also decided that two versions of the KB could be offered to prospective clients: (a) the existing version which relies on a REST-based client-server architecture and (b) a standalone version where the KB remains locally at the client. The reason for introducing the second version was mainly the fact that the communication overhead from the transport of derived facts from the REST server to the client can be big, if the size of the facts to be transported is enormous. In addition, there can be certain requirements which demand the private use of a KB which comprises rules which have been modelled and are exploited by the user. While the REST-based version of the KB does offer a private space to the user, it cannot overcome the transportation issue, whatever is the design of the domain model. Obviously, it cannot also guarantee an ultimate privacy level as certainly an administrator has to maintain the KB server and can certainly have access to the rules of the users/clients. Another issue with the REST-based version of the KB can be related to the fact that the performance can be reduced when a certain load reaches the KB server. However, such a scenario has not been anticipated until now and of course

the current number of KB users is not quite big such that the respective load limit is reached.

5.2.2 Domain Level Re-Engineering

The domain meta-model of the KB was modified in order to cater for the following: (a) updates to the CAMEL meta-model and (b) information compactness reasons. To this end, the meta-model was re-engineered in order to include additional classes, which pertain mainly to the way particular EMF/CDO constructs are exploited as well as modify existing classes in order to contain pointers and not the actual information to be returned. Actually, the design of fact classes relied on the following principles: (a) only the most important information is modelled outside the CAMEL information space and (b) in case, CAMEL information is to be included, it should only be referenced. The second principle leads to the requirement that any code that exploits the facts derived from the KB should be able to connect to the CDO repository and obtain the information that is pointed out/referenced. In this way, we guarantee that the transportation delay and cost is as small as possible and that there can be no significant problems in the marshalling and un-marshalling of the information to be transported. The latter problems were inherent in the previous version due to cyclic dependencies between Camel classes which lead us to introduce JAXB annotations only at certain places in fact/domain classes in order to stop the marshalling before a cycle is reached.

The main construct through which CAMEL information is referenced is the *CDOID*. This EMF/CDO class maps to a pointer which can uniquely identify any object that has been stored in a certain CDO repository. This means that the actual object can be retrieved through this *CDOID* as long as the code connects to the appropriate repository on which the object has been stored. Fortunately, the *CDOID* is serializable and this means that it can be persisted and transported via the network. However, it cannot be easily marshalled and unmarshalled. To this end, we had to include in the domain meta-model specific marshalling adaptation code which is able to map a *CDOID* to a String, which is of course then easily marshallable, and the opposite (from a String to CDOID when un-marshalling).

According to our consideration, there is a clear separation between basic information drawn from a database (MDDB) and added-value knowledge derived from this information. However, it might be argued that CAMEL could be extended to include the modelling of this added-value knowledge. In our opinion, this must be avoided for the following two reasons: (a) as the added-value knowledge relies on the basic information stored, any change of this information will require the updating of the added-value knowledge - this requires a component which constantly polls the MDDB for such changes and then fires the corresponding rules, thus leading to a substantial overhead which is not actually needed as the added-value knowledge can be retrieved on demand; (b) the decoupling of knowledge from information facilitates users to independently extend the knowledge domain model with their own classes pertaining to the rules that they define thus catering for an individualized and customized exploitation of the KnowledgeManagement API offered.

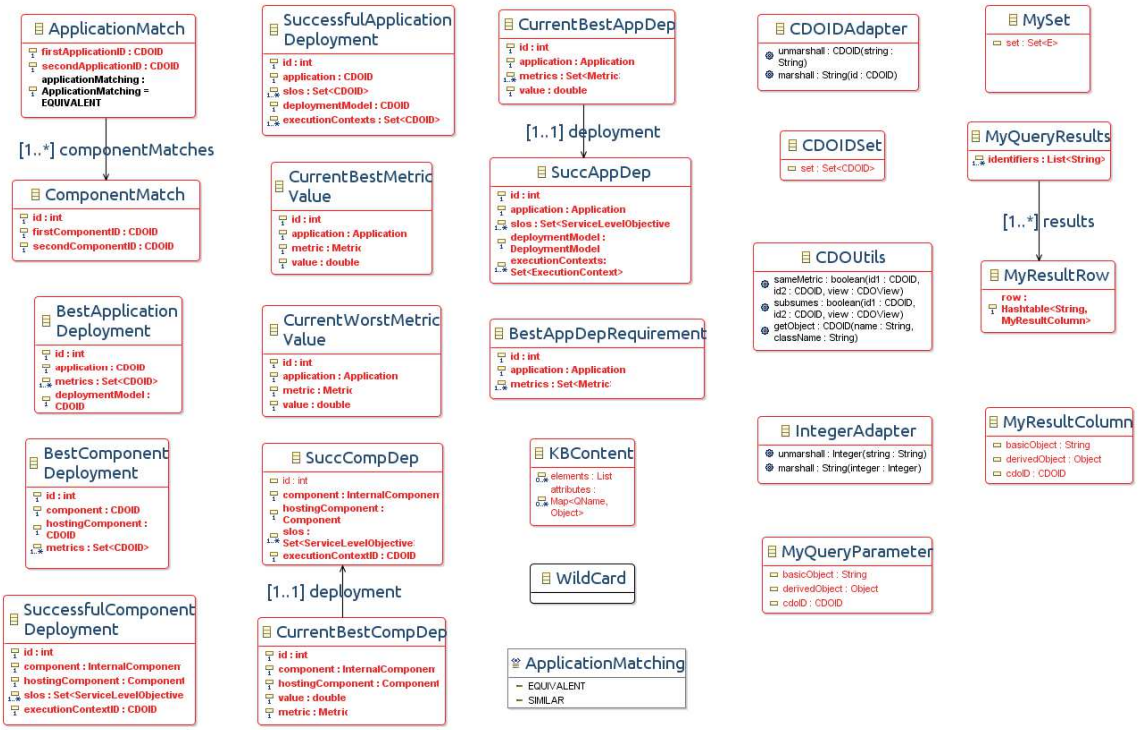

Figure 14 - The class diagram of the KB domain meta-model

Based on the above analysis, the domain meta-model re-engineered can be viewed in Figure 14. This meta-model comprises 24 classes that are now analyzed in more detail.

Classes *ComponentMatch*, *ApplicationMatch*, *BestApplicationDeployment*, *SuccessfulApplicationDeployment*, *BestComponentDeployment* and *SuccessfulComponentDeployment* represent the main outcome of the basic rules and represent the added-value knowledge produced through the firing of these rules. We now separate their analysis according to the type of knowledge that is derived:

- *ComponentMatch* and *ApplicationMatch* represent the matching between two (internal) components and two applications, respectively. The matching information of two applications, apart from referring to the *CDOID*s of the applications themselves also contains information about the type of matching (full or partial) as well as the set of *CDOID*s of the (common) containing components that have been matched for these two applications. A partial matching between an application pair occurs when a specific percentage above a certain threshold (set as a global variable in the respective session) of components of the first application has been matched with those of the second application. A full matching occurs when all components of the first application have been matched and both applications have an equal component set size. In this sense, full matching is symmetric while partial matching is not, as in the second case we can have an application with a bigger component set size where only a percentage of the components of this set are matched with those of the second application. The matching of components relies at the moment on their name but could rely on other information aspects, if they are specified by the user, involving techniques focusing on the information-retrieval-based matching of the component descriptions or on code-based similarity assessment. Provided that the latter information aspects are

available, the respective rules could be enhanced in order to make the matching more semantically rich, especially as the KB allows using of arbitrary (Java) code which could focus on addressing the matching on one or more information aspects.

- The remaining (derived knowledge) classes from the aforementioned set refer to best and successful deployments of applications and components. Each class, apart from referring to basic information (*CDOID*s of components/applications and respective deployments), is also linked to the some other related CAMEL objects (again through *CDOID*s). In particular, successful deployments are associated to the set of *CDOID*s of the SLOs that have been respected and of the execution context(s) involved. On the other hand, best deployments, as derived from successful deployments, do not require referencing whole SLOs but particular metrics involved at the top-level in these SLOs. Another rationale for this relates to the fact that a best deployment should be the most optimal one irrespective of any SLO imposed on a particular metric. Thus, a deployment is regarded as best when it leads to the best value of a metric which can satisfy even the most demanding SLO posed for this metric in the context of any application. It should also be highlighted that best deployments are associated to a set of metrics and not just one metric based on the fact that if a deployment is the best with respect to more than one metric, then it is better to gather the references to these metrics into a single object rather than having only individual best deployments for each metric that are equivalent. This also facilitates the posing of (KB) queries that take into account multiple metrics and desire to discover those deployments that are the most optimal ones according to the conjunction of these metrics.

Intermediate and convenience classes have also be generated and are analysed as follows:

- The *CurrentBestAppDep*, *CurrentBestCompDep*, *CurrentBestMetricValue*, *CurrentWorstMetricValue* classes represent intermediate facts that lead to the final generation of the added-value knowledge. The first two represent the current best deployment that has been identified so far during the progress in the rule firing for applications and components, respectively. The latter two classes represent the best and worst metric values that has been so far seen for a particular application and metric such that these values can then be compared with the remaining successful application deployments in order to finally derive the best one.
- The *SuccAppDep* and *SuccCompDep* are convenience classes equivalent to those of the *SuccessfulApplicationDeployment* and *SuccessfulComponentDeployment* classes, respectively. Their use leads to less requirements with respect to the posing of queries to the CDO Repository as they map to full copies of the underlying CAMEL objects and not just *CDOID* references. As such, the firing of rules is not delayed (due to the issuing of queries attempting to obtain actual objects from *CDOID*s) while it does not also lead to substantial load related to the handling of CDO Repositories of even greater size.

The following classes relate to the REST-based I/O handling, especially in terms of object marshalling and un-marshalling. In this sense, they mainly apply to the REST-based version of the KB.

- *KBContent* represents the information in the form of objects/facts derived in a particular session of a specific KB. Objects of this class can also be regarded as potential content to be stored in the KB in the context of a particular session.

- *MyQueryParameter* represents query parameter information to be passed for running a particular query to a stateful knowledge session. Special care must be given to the way the class information is specified as there are three types of objects that can be used: (a) basic Java objects to be mapped to a String (see *basicObject* parameter) related to the representation of e.g. names or any other kind of attribute pertaining to a certain class that is expected to be used as input parameter to a query; (b) *CDOID*s to indicate a required cross-reference to a particular CAMEL object; (c) complex Java objects which could be related to particular knowledge domain classes that have to be passed as input parameters.

- *MySet* represents a set of any domain object to be passed as input or output to the KB API methods. For instance, it can be used to represent the content of a query parameter (*MyQueryParameter*) in the third object type case as explained previously.

- *MyQueryResults* represents the resulting information from a particular query to a stateful knowledge session. It comprises row-based information represented by another class called *MyResultRow* along with a particular set of column identifiers that can be used to fetch the respective column value of a row-based result. The columns of a result row are represented by the *MyResultColumn* class which has similar content with respect to a query parameter.

- *CDOIDAdapter*: As a CDOID is just an interface, we need to define an extension to *XMLAdapter* to be able to map the content of a *CDOID* into a String during marshalling and back from String to *CDOID* during un-marshalling.

- *Wildcard*: It is used to represent a parameter with any possible value to cater for the posing of generic KB queries which should give back as a result all objects that match the query focus (e.g., the best deployment for all applications if no concrete application or metric is provided as input to the query). It should be noted that any expected input parameter to a query can be represented in this way catering for fully generic (all parameters are generic) or partially generic queries (some parameters are fixed).

- *CDOIDSet*: It is a special class used to represent a set of *CDOID*s. This is required in cases that a knowledge object (e.g., instance of the *BestApplicationDeployment* class) refers to a set of *CDOID*s (e.g., mapping the respective metrics for which this application deployment is the best) and not just to a single CDOID.

Finally, the *CDOUtils* is a utility class including the specification of static functions which can be used in the context of rules specification to facilitate the compact representation of rules by encapsulating functionality repeated across rules in the form of a method call. For instance, whenever a rule desires to inspect in the antecedent

part whether two SLOs map to the same metric, it can call the *sameMetric* method of the *CDOUtils* class in order to obtain the respective boolean evaluation result. The static function set of this class can be extended by including additional special-purpose functions which can be employed in order to support more advanced types of rule inferencing (as in the case of code which can be used to infer the semantic matching of two components).

5.2.3 Rule Re-Engineering

As the substitution of Hibernate with CDO technology maps to the same type of information access, some of the rules, especially those mapping to the matching of components and applications were not significantly modified apart from the fact that the rule code used to perform the query has been slightly modified. However, the rules used to derive best application/component deployments was re-engineered for the following additional reasons: (a) to be able to derive best deployments in the context of metrics and not SLOs - it is not desirable to produce best deployments for a specific SLO as this does not make sense. It is more appropriate to discover the best deployment in the context of a particular metric that is used to formulate the SLO as this is truly a best deployment because it leads to the best possible value that can be achieved by the examined component or application for the respective metric; (b) the CAMEL meta-model has evolved in a significant manner, including the modification of the content and structure of particular meta-models and the splitting of some of these meta-models. In this respect, the previous rules for best application/component deployment have been modified to a great extent and have become richer. The prospective reader can refer to the source code of the PaaSage prototype[13] at the OW2 AppHub European Open Source Marketplace in order to find out more technical details about the current set of basic rules modelled.

We need to stress here that this basic set of rules can be extended in order to derive extra added-value knowledge. By considering that the CAMEL meta-model will not undergo significant modifications in the near future, we will concentrate more on the development of new rules in the forthcoming working period in order to highlight in an even more increased and explicit manner the power of exploiting a KB.

5.3 Architecture

The architecture of the KB is shown in Figure 15. As it can be viewed, there are two versions of the KB, as already mentioned. The REST-based version includes two components, a server and a client, where the former can reside for example in a central VM, such as the one hosting the running PaaSage platform instance, while the client resides at the code which requires to have access to the KB in order to fire rules and obtain back the results. The standalone version comprises just one client which encompasses a KB realisation that can directly manage it through the creation and management of respective sessions. In both versions, the REST-based server and standalone client need to interact with the CDOServer in order to issue queries and that is the reason why they incorporate a particular CDOClient. Figure 15 also shows what are the main clients of any version of the KB: (a) the SN which can exploit it in order to provide added-value information to its users, e.g., in the form of deployment suggestions; (b) a UI/IDE or standalone component which desires to derive added-

[13] http://www.ow2.org/bin/view/ActivitiesDashboard/PaaSage

value knowledge in order to present it to the user (e.g., a monitoring UI could employ the KB in order to visualize statistics-based or -derived facts about the performance of an application); (c) a deployment reasoner which could rely on the best deployment suggestions in order to reduce the complexity of the constraint problem to be solved as well as filter the provider space.

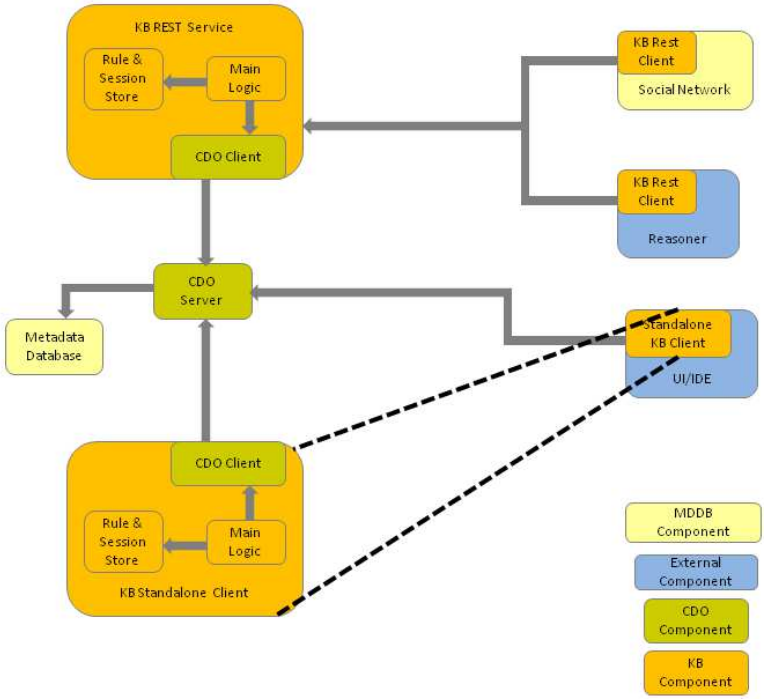

Figure 15 - The architecture of the KB

5.4 Technical and Implementation Details

The main functionality of both versions of the KB has not been modified. In this sense, the KB still offers capabilities to manage knowledge bases and respective stateful knowledge sessions. A conceptual view which indicates the lifecycle of knowledge bases and stateful sessions is depicted in Figure 16 which also clearly indicates the type of individual functionality offered by the KB. As it can be seen, stateful sessions exist in the context of knowledge bases and thus are immediately deleted when their encompassing knowledge bases are removed. In addition, it is quite clear that many sessions can map to the same knowledge base, especially as the knowledge base can be considered as a container of rules while a session represents a particular live exploitation-based instantiation of a knowledge base which includes the ability to fire rules and produce new knowledge which can be used to populate the fact base attributed to this session. In this sense, a knowledge base is actually a Rule Base while a stateful knowledge session is a live Fact Base which is continuously updated through the firing of the rules of the Rule Base. Of course, this kind of terminology used (e.g., knowledge base vs. rule base) is relevant in the context of Drools Expert[14] but does not move from the main point that a session or fact base cannot live independently of a certain rule base.

Moreover, we would like to stress that no significant API modification was involved, apart from the fact that the standalone version has a different signature for each API

[14] www.drools.org

method. This is of course related to the fact that the standalone version does not require any special class to perform any kind of marshalling or un-marshalling. However, we have to stress here that the API methods map to identical names and this can be convenient for potential exploiters that might require switching between the two different KB versions or test which one is more appropriate.

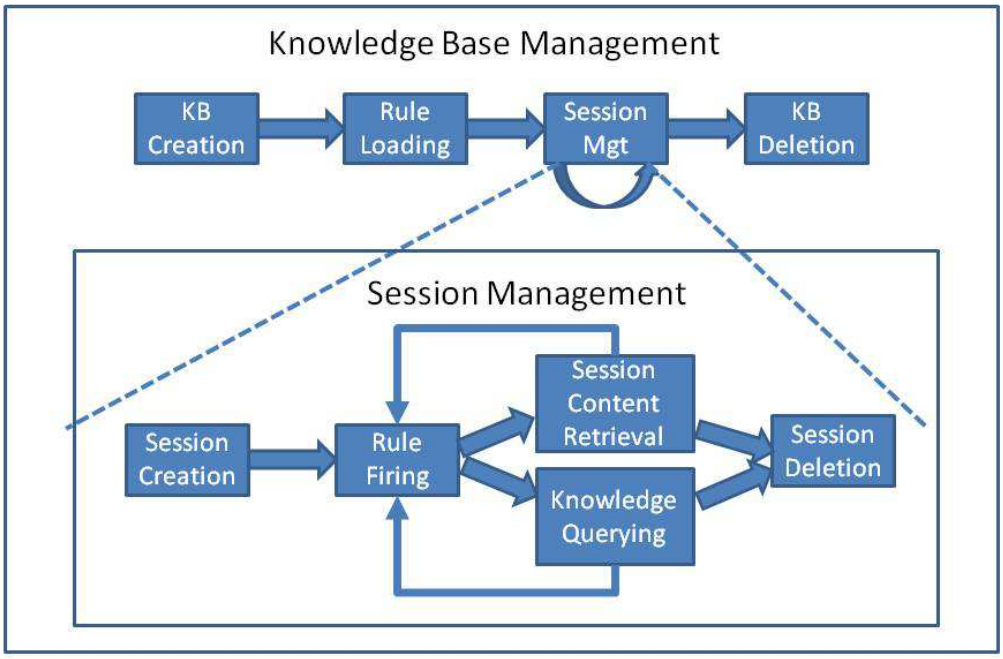

Figure 16 - The knowledge base and session management lifecycle

Concerning technical information about the exploitation of any of the two KB versions, the interested user can refer to the technical documentation of the KB, which is included in a separate section in the Appendix of this deliverable.

As the KB API was kept constant while also the same KB realisation technology was used, the code re-engineering efforts actually lead to an additional implementation dependency which comes via the use of the CDOClient, needed to connect to the CDO repository and obtain the information stored there.

6 Social Network

A key design objective of the PaaSage Social Networking platform is to create a strong bond between: (i) software engineering services for managing and deploying cloud-targeted application models and (ii) community-oriented facilities for communication and collaboration between users. The interconnections between the two in the design of the user interface are depicted in Fig. 17 and further explained in Section 6.3. The prototype implementation is publicly accessible online at http://socialnetwork.paasage.eu.

Users of the PaaSage Social Networking platform are expected to be familiar with basic principles of software modelling and distributed cloud-based application design and deployment. Their roles will range from DevOps engineers and cloud deployment specialists to IT architects, cloud developers, software engineers, application designers, and system administrators.

We should highlight here that we interchangeably use the terms PaaSage professional network and PaaSage Social Networking platform. The former indicates that the social networking platform is not just for normal social network users but also for professionals in cloud computing which use the social networking platform in order not only to share knowledge about cloud computing but also to exploit some of the facilities provided by the PaaSage platform, such as the deployment of applications.

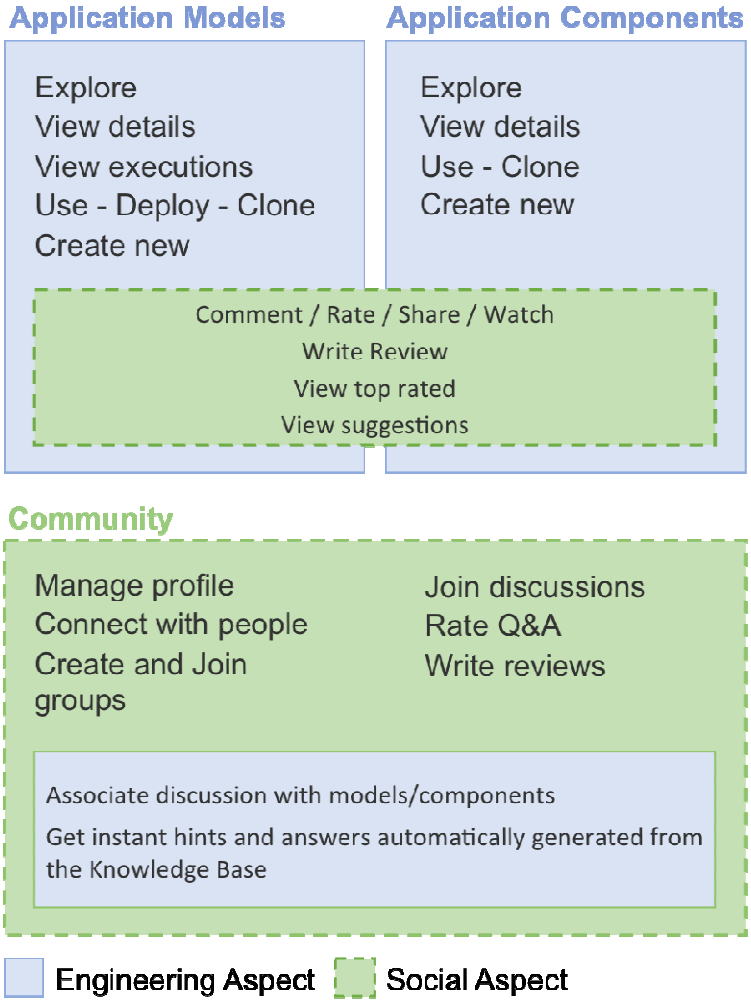

Figure 17 - Engineering and social activities are seamlessly interweaved within the PaaSage professional network

6.1 UI design

6.1.1 User interface design process and practices

The user-facing interface of the professional network was designed following an iterative approach [8, 9], alternating design with expert-based evaluation involving both usability and domain experts. The main benefit of this approach is that problems are identified early in the development lifecycle and can be corrected before the implementation, even in cases where the agile programming method is applied [10], ensuring high quality of user experience [11]. Discussion of user requirements and evaluation of the designed mock-ups were carried out in focus groups involving users of social and professional networks, software engineers, cloud computing experts and usability experts.

The focus groups involved seven participants with the following characteristics: two cloud computing experts with limited experience in social and professional networks, three software engineers –all of which were regular users of social networks and one was also experienced in using professional networks– and two usability experts, one of whom had some experience in cloud computing platforms and DevOps environments. During the iterative design process the same group of experts and users was engaged in discussions regarding requirements elicitation and evaluation of the mockups. Given the extent of the design (100 mockups were produced) the process required 13 sessions to evaluate the mockups. The discussions carried out within the focus group led to thorough redesigning and usability fixes for several parts of the professional network. An example of how the design of a system component (the Application Model Overview Panel) evolved over time using our approach is illustrated in Fig. 18.

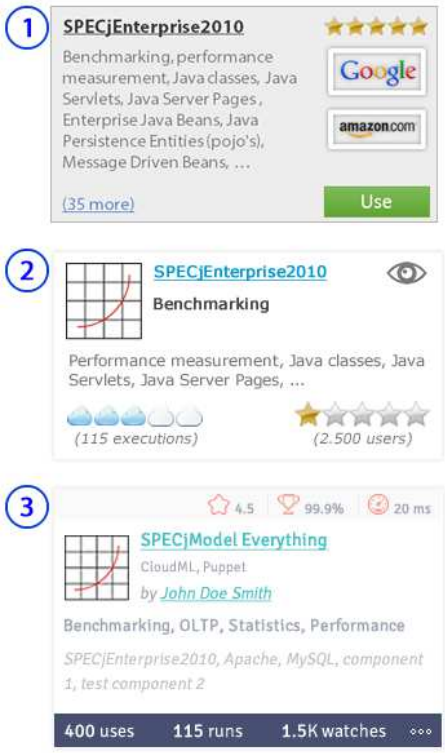

Figure 18 - Evolutionary design of the application model overview pane

Overall, the design of the PaaSage professional network had five objectives:

6.1.1.1 Strong bond between software engineering and community-oriented activities

The design ensures that pages oriented towards software engineering information include details and prompts for community-building and vice versa in order to exploit the generational experience [12] and enable new ways for software developers to work together [13] and build stronger bonds [14]. For instance, in the Application Model page users can view information about the model and also: (i) information on model contributors, enhanced with direct options for sending them a message or asking them to be connected; (ii) the most popular tags that describe the model, with facilities for adding tags; (iii) overall rating of the model and most popular reviews. Similarly, in a user's Profile page, his/her top model and component contributions are displayed at a prominent area, along with ready-to-use options for using or running these models. The two types of user actions are seamlessly interweaved; being distinct perspectives however, they can be easily distinguished through design cues, such as different colors used consistently throughout to indicate different action types: blue (turquoise) for model-related and red (terra cotta) for community-related activities. Figure 17 summarizes these concepts.

6.1.1.2 Prioritization of personalized information

Personalized information [15] (relating content to the user and his activities) is included and prioritized in all relevant pages to increase persuasion [16] and facilitate participation for both novice and expert users [17]. For example, when browsing application or component models, system recommendations for models that might be of interest to the user are displayed topmost. In the Application Model page a user can select to view and manage his/her own configurations for the specific model. The importance of the user's personal stuff is also reflected in the design of the Main Menu (Fig. 19). The Main Navigation Menu can be divided into three sections: (i) user's homepage and navigation to network content (applications, components, community); (ii) search; and (iii) user's personal material, including his own model and component designs and deployments, profile, notifications, messages, cart and settings. Thus, the design promotes three modalities of interaction: browsing, searching for, and personally contributing content.

Figure 19 - Main navigation menu

6.1.1.3 Rich home page

An important concern during the design of any website is its home page functionality and layout. The home page is the flagship of every site [18] and should be designed accordingly, displaying vital information both to first-time and regular users. Such information may include [19]: site identity and mission, site hierarchy, search, content and feature promotion, timely content, and shortcuts. Along these guidelines, the home page of the PaaSage professional network was designed to act as an information point for activity related to the user's models and components and to community activity that might be of interest.

The home page of a signed-in user includes: (1) Statistical information and graphics regarding the user's models and components (e.g., top rated and most-run models, number of reviews, discussions, runs, uses and watches of his models and components, etc.) as shown in Fig. 20; (2) Live feed including model- and component-related information (e.g., new models or components made public, updated configurations of models, badges received and new reviews for a model or component, etc.) and community-related facts (e.g., who has an updated profile or was endorsed for a skill, who has made new connections, who posted questions and answers regarding a specific topic, etc.); (3) Information on the user's profile, such as profile completeness, skills and areas of interest. (4) Suggested content according to the user's profile and recent activity, including potentially interesting models, groups and connections.

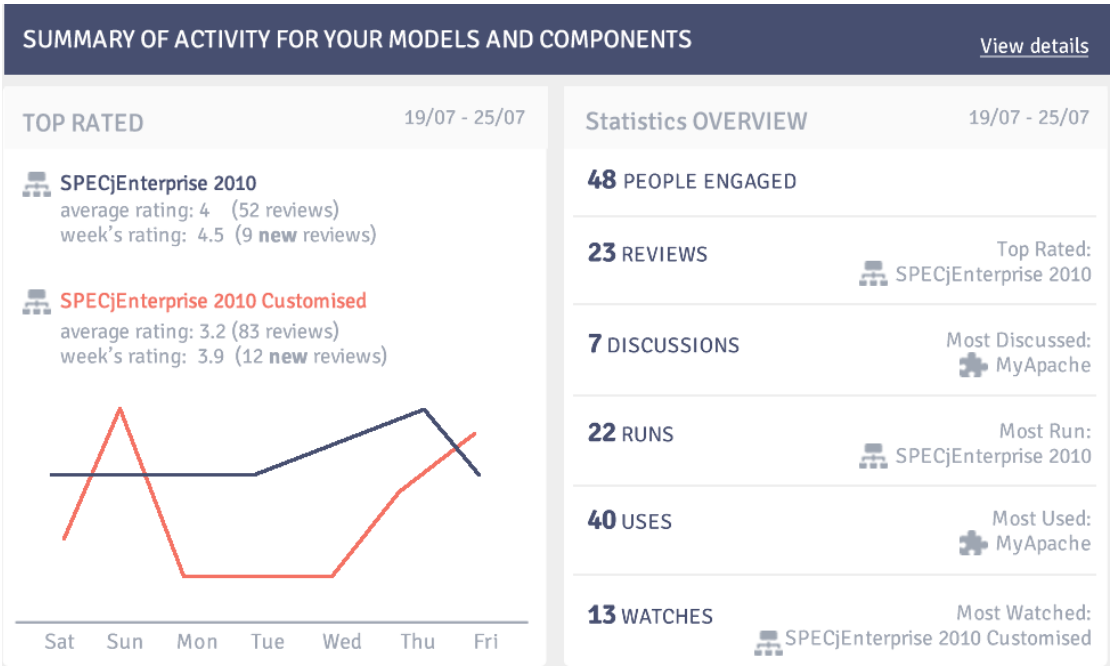

Figure 20 - Statistical information and graphics for user's models and components

6.1.1.4 Nurture an active and sustainable community

A vital consideration for any online community is how actively its members are involved in it, which can be determined by the quality of the content itself [20] and by the design of its user interface [21, 22], which should support and facilitate communication, collaboration, and content contribution. To nurture an active and sustainable community, our design applies the following practices: First, it makes content contributions (models, components, questions, answers, ratings and reviews) visible to the community. Second, it cultivates a sense of belonging by presenting activities and contributions of familiar people which may be relevant to the current context. Third, it supports both major contributions (e.g., models or components) as well as smaller contributions (e.g., ratings, reviews, or question-asking). Fourth, it recognizes high quality contributions (e.g., top rated models, mostly up-voted answers) as well as high quantity contributions: the more a user contributes, the more expertise points he gains (top contributors receive special badges as an indication of expertise). Fifth, it enhances the feeling of safety and trust, by making links to the terms of services and privacy policy easily accessible through any network page. Furthermore, users can define privacy settings regarding who can see their data, send requests to them, or look them up [17]. Finally, any user can report members or

groups to network administrators if they feel that they violate intellectual rights or exhibit abusive behaviour.

DevOps professional communities have indeed shown interest in supporting and using such platforms. For example, the community that has formed around Chef Supermarket is an active and sustainable. While Opscode, Inc. is the single largest cookbook maintainer in the repository, about 96 % of the cookbooks in the site are maintained by other developers, students, organizations, etc. Most of the shared cookbooks seem to be maintained by independent developers rather than companies, if we take contributor e-mail domain as an indication (gmail.com is by far more popular (507 users) than opscode.com (121 users) or getchef.io (68 users)). Finally, a majority (about 62 %) of cookbook maintainers have shared only a single cookbook. We expect that the Chef Supermarket community will share characteristics with the community to be formed around PaaSage professional network.

6.1.1.5 Motivate active and regular participation

Following recent trends in HCI design and with the aim to motivate active and regular participation in the PaaSage professional network, the design employs gamification features, namely use of video game elements to improve user experience and user engagement in nongame services and applications [23]. One gamification feature in our current design is the reward system for active community members. As users contribute content (models, components, ratings, reviews, questions, or answers) they receive experience points leading to special badges visible to all community members. Other features are the Profile completeness bar with suggestions on how to increase it, and a skills' endorsement system: a user can be endorsed by others for specific skills. Such endorsements are visible to all his friends' live feeds, increasing his popularity in the community. Finally, the concept of Model badges awarded to application and component models in case of excelling performance. Badges can serve among others as goal-setting devices, status symbols, and indications of reputation assessment procedures [24].

6.2 Main Functionality

6.2.1 Application Modelling

The PaaSage Social Networking platform aims to support users who wish to explore, deploy, and optimize application models and components.

6.2.1.1 Model Exploration

The platform offers personalized and universal exploration facilities to accelerate model discovery. The models page depicted in Fig. 21 concentrates information on application models available in the network. The level of personalization on the displayed content is known to strongly influence overall user experience [25]. Based on this principle, our platform delivers (i) personalized recommendations based on each user's areas of interest (Fig. 21(a)) and (ii) context-sensitive lists of recently-viewed models and components for quicker reference (Fig. 21(b)). Among universal features, the most commonly used yet highly efficient one is content classification based on explicit categories. Previous studies have shown that it simplifies browsing [26] for almost 50 % of targeted users [27, 28]. Content organization is further enhanced with tag annotations that can be set in a category-independent manner,

motivating a classification-by-use basis [29]. To leverage the collective experience of professional network members, we facilitate the discovery of new and noteworthy content by promoting featured, popular and trending application models based on their community impact [30]. To assist users that prefer searching over browsing [27, 28] we provide an advanced filtering mechanism and a faceted-search facility to enable that type of content discovery. Filters are differentiated based on context. Filters on the initial models' category page include model categories and recently-used models. When viewing models under a specific category, filters narrow down results based on: model status, deployment platform, modelling framework, model cost, minimum uptime, maximum response time, minimum throughput, geographical distribution of the model's executions, and specific tags related to the model. When viewing the page of an application model, filters narrow down the displayed executions based on execution start and end date, cost, uptime, response time, throughput, geographical distribution, cloud provider, and execution owner (user, user's network, all users).

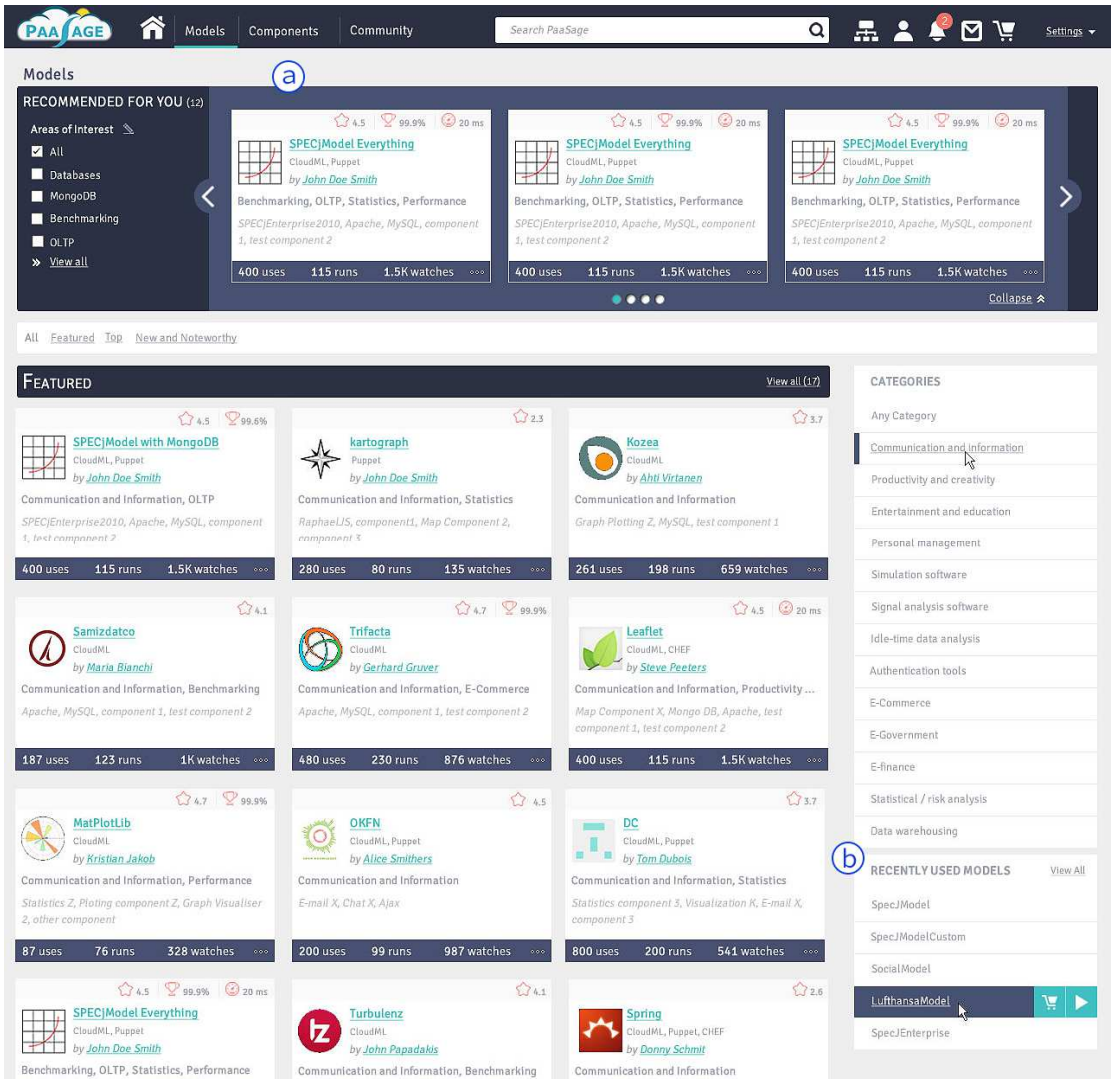

Figure 21 - Personalized models page

6.2.1.2 Focusing on a model

An application model is a discrete entity with a wealth of heterogeneous information that must be delivered in a well-structured manner. Combining this requirement with the fact that the popularity of a model is based on efficient broadcasting of its strong features, the PaaSage professional networking platform employs the concept of Pages typically used to promote organizations/businesses in social networks (e.g., Facebook fan Pages, Google+ Pages, LinkedIn Company Pages, etc.) [31].

At the top of an application model Page (Fig. 22(a)) a user can find essential information regarding that model, including its identity and impact in the community. Apart from the basic description and contributors, a mix of social information (e.g., rating, top review, number of watches and shares) and engineering information (e.g., version, number of deployments and uses) allows the user to form a clear picture about the model's capabilities. Interactive controls permit rapid action from an engineering (e.g., cloning, deploying, use) and social perspective (e.g., watching or sharing a model).

Besides the model overview, a secondary menu (Fig. 22(b)) allows users to get additional information through separate screens that present: the hierarchical structure of its components; any related discussions and reviews; similar models; and lists with extended execution statistics (runs). The past-executions list stored in the CAMEL information repository offers the ability to compare execution statistics and identify the most successful configurations (i.e., identify best practices). Due to the large amount of information, we use a rich filtering mechanism to narrow down the displayed data, while graphical visualizations facilitate quicker analysis (e.g., uptime, response time, throughput, cost, etc.). The same design approach was used on component Pages; the only exception is the fact that the execution statistics screen is missing as such data are not available for individual components.

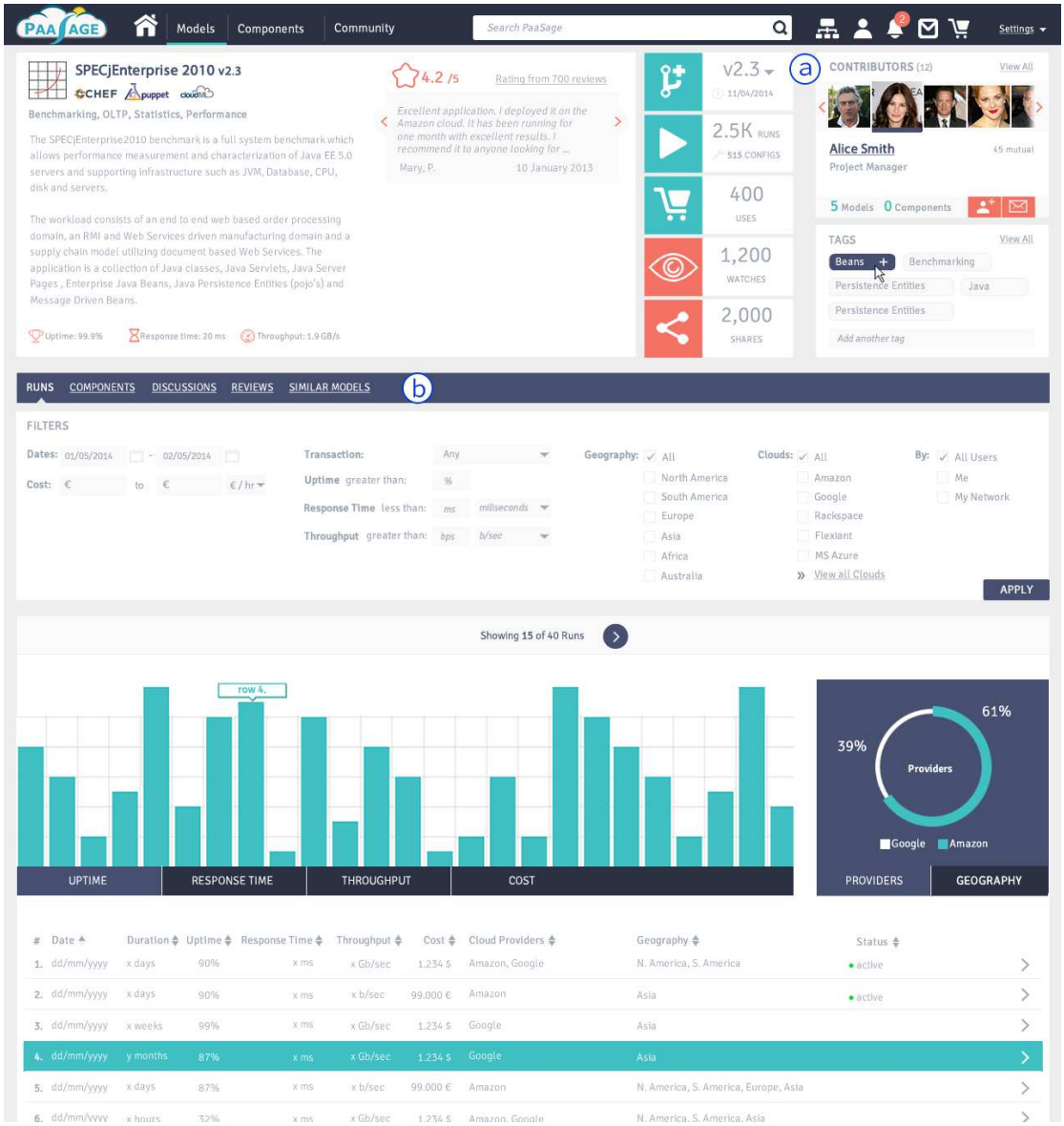

Figure 22 - Application model page

Although the PaaSage professional network does not provide an integrated model editor, it does provide some support towards creation of application models. Members can import new models or reuse (clone and modify) existing models to benefit from the design principle of service composability. We have applied the shopping cart metaphor (Fig. 23) so that users, while browsing, can save application and/or component models of interest for later use.

Among cart features, the most advanced is the ability to collect entire categories of components (e.g., Web Servers). The PaaSage professional network relies on proven external model editors for creating or modifying models. The cart can deliver components to such an editor, in which model composition and further editing would take place, prior to importing the new model back into the platform.

To further support engineers, new models are initially marked as drafts and their visibility limited to the model owner and any contributors explicitly invited. At any time the owner is able to publish a model to the community by marking it as published. From that point on the model becomes public and every user can use,

clone, or deploy it. Prior to publishing, the owner is also able to determine whether the model is a self-contained element that can be deployed as-is (i.e., application model) or it is a composite component that must be embedded in other application models.

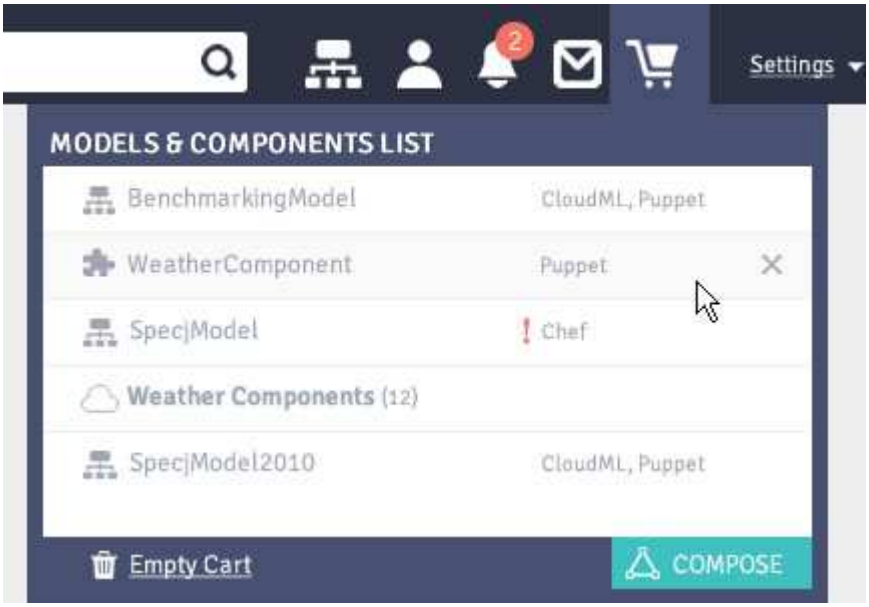

Figure 23 - Models and components list

6.2.1.3 User's personal area

An important component of the network is the user's personal area, which reflects the user's activity towards building and deploying models and components. It is also an information retrieval point on community activities related to the user's models. The PaaSage professional network introduces various tools in a user's personal area to facilitate management of model-based distributed applications. Users get live statistics of their active executions and can easily control them (e.g., stop, undeploy one or more instances) (Fig. 24). They can modify the details of individual configuration schemes (e.g., VM properties, resource allocation, distribution, etc.) on a model-by-model basis. For convenience, context-sensitive actions and aggregated statistics regarding engineering and social performance (e.g., number of uses, rating, average execution time, etc.) are visualized adjacently.

Figure 24 - List of currently running applications

6.2.2 Community

Apart from social features that facilitate distribution and promotion of application models and components (e.g., share, rate, review, watch), the professional network encourages the creation of an active organized community that supports collaboration (Fig. 25).

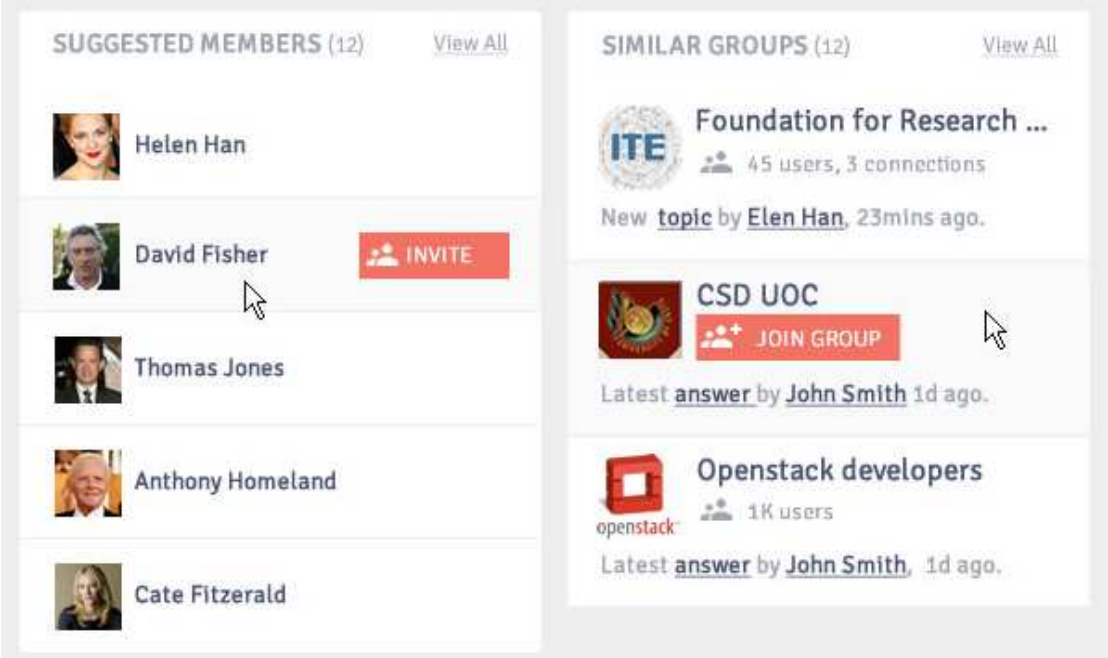

Figure 25 - Users are constantly motivated to participate in social activities

People with similar interests can connect and follow each others' updates. The formation of groups promotes a team spirit, exchange of opinions, and knowledge creation through the accumulation of experiences. Members are encouraged to participate in discussions by posting their questions and answers on topics of interest and rate or review the answers of others. To further contextualize discussions, the system offers the ability to associate posts with references to models and components (similar to attaching a file in an e-mail), while instant and automatically-generated hints are provided when possible by the network's knowledge base. Figure 10 depicts an example where two hints are automatically generated via simple queries (constructed based on the keywords "Amazon" and "SpecJEnterprise" found in the question body) to the CAMEL repository. They are meant to inform the user about the quantity of information that is possibly relevant to their question.

Community growth is stimulated by intelligent suggestions on connections or groups that may interest a user. Besides invitations to connect with other members or to join new groups, suggestions may also encourage individuals to endorse their connections for skills or even invite members to groups to strengthen community bonds. An adaptive approach has been employed to deliver and present the ideas: prompts are either displayed on dedicated areas (e.g., sidebars) or embedded in the main content area, mixed with existing notifications (e.g., updates in models in watchlist, new answers on topics that the user currently follows, etc.). To support new members, the system integrates advanced help facilities, including FAQ and Q&A sections, with intelligent mechanisms that discover and suggest similar questions or relevant answers.

6.3 Implementation Details

The social networking platform is implemented over the Elgg extensible social network framework [32]. Elgg is open-source software written in PHP, uses MySQL for data persistence and supports jQuery for client-side scripting. The Elgg framework is structured around the following key concepts:

- Entities, classes capturing concepts such as users, communities, application models, etc.
- Metadata describing and extending entities (e.g., a response to a question, a review of an application model, etc.).
- Relationships connecting two entities (e.g., user A is a friend of user B, user C is a contributor to an application model, etc.).

Elgg comprises a core system that can be extended through plugins. The core comes with a few basic entities (Object, User, Group, Site, Session, Cache) as well as other classes necessary for the operation of the engine. All Elgg objects inherit from Entity, which provides the general attributes of an object. Plugins add new functionality, can customize aspects of the Elgg engine, or change the representation of pages (examples are the Cart system or the handling of Application Models). A plugin can create new objects characterized (through inheritance of Entity) by a numeric globally unique identifier (GUID), owner GUID, and Access ID. Access ID encodes permissions ensuring that users has access only to data they have permissions on.

Figure 26 shows the model, view, and control parts of Elgg's architecture. In a typical scenario, a web client requests an HTML page (e.g., the description of an application model, Fig. 22). The request arrives at the Controller, which confirms that the application exists and instructs Model to increase the view counter on the application model object. The controller dispatches the request to the appropriate handler (e.g., application model, component handler, community handler) which then turns the request to the view system. View pulls the information about the application model and creates the HTML page returned to the web client.

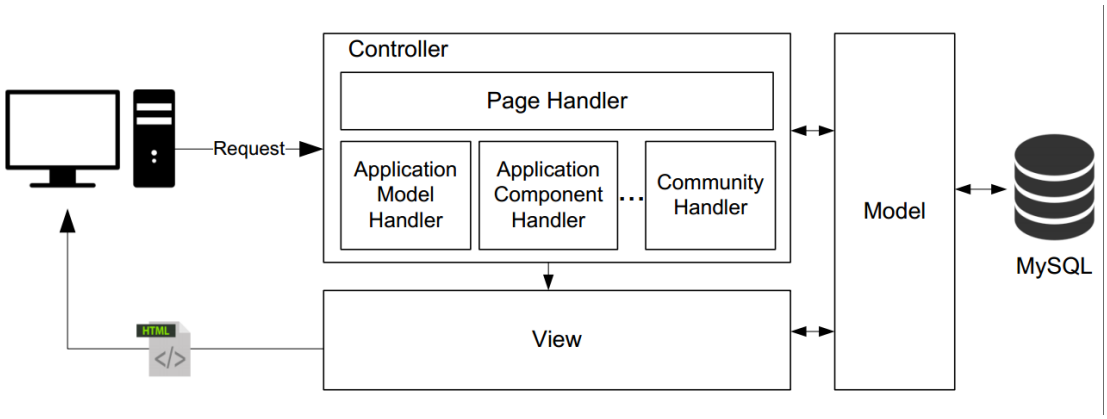

Figure 26 - Architecture of the Elgg Social Networking engine. Page handlers refer to functionality of implemented plugins

All plugins share a common structure of folders and php files. Folder actions includes the actions applied on application models (delete, save, or search). The views folder contains the php forms applied on application models and river events (live feeds).

Pages overrides elements of core Elgg pages. The js and lib folder provides javascript and php library functions. Finally, the vendors folders include third-party frameworks such as Twitter's bootstrap frontend [33], which lends its responsiveness, look and feel, and portability across Web browsers to the PaaSage professional network.

Social network relationships (friendship, group, ownership, etc.) are persisted in the Elgg back-end database. The execution history of deployments of application models and the description of those models is stored in the CAMEL information repository, which is implemented as an Eclipse CDO server. The exchange of information between the Elgg and CDO servers is implemented over network sockets.

7 MDDB Product Evaluation

7.1 CDO Evaluation

This section focuses on evaluating the capability of CDO to handle in a scalable way an increasing number of data and users. This evaluation has been performed for the two main types of a CDOStore, namely the DBStore and HibernateStore, combined with two different databases, namely MYSQL and HSQLDB. By following all possible 4 combinations of CDOStore types and underlying databases, we reach interesting conclusions about which combination to adopt in the end in order to fully realise the PaaSage prototype platform.

We need to underline here that the switching between stores of the same type is quite easy as there is no impact on the content of queries exploited at the main logic of any PaaSage component. However, the switch to a store of a different type, does have an effect which can lead to the re-engineering of the queries but not the actual (domain-specific) code. We do not advocate that this switch could be performed during the remaining limited lifetime of the project, especially as the focus is on delivering the promised functionality, but could definitely be performed by a potential exploiter of the PaaSage platform who wishes to create a prototype or product with an even better performance.

In the evaluation, we are also considering the results of the experiments conducted for the original database-based (MySQL) implementation of the MDDB in the context of D4.1.1. In this way, we are able to indicate what is the overhead introduced by adopting a layered solution which enables the manipulation at a higher layer of models instead of data at a lower layer.

Equivalently to D4.1.1, the same set of experiments were conducted with as much as possible the same underlying content produced. The three experiments are summarised as follows:

1. Evaluation of the size of the (underlying) database with an increased number of applications (whose description includes a certain amount of concrete deployments and measurements)
2. Evaluation of the answering time for a query which attempts to retrieve all the successful component deployments (on respective VM instances) with respect to the SLOs posed
3. Evaluation of the answering time for a query which attempts to retrieve the VM offering which is mostly instantiated across all application deployments.

The respective results for this experiment set are presented in the same order in the following sub-sections.

7.1.1 DB Size Experiment

The results of the experiment on the (underlying) database size are depicted in Figure 27. As it can be seen, the best implementation option maps to having just one layer, the DB one, as it leads to the minimum possible DB size (4 times less than the best CDO realization approach). However, we have to stress here the following main factors that contribute to the increase in the database size for the CDO layer-based approach:

- the CAMEL meta-model has evolved and now contains much more information with respect to the DB schema of the previous version of MDDB

- CDO creates additional tables storing CDO-specific (and not model-specific) information

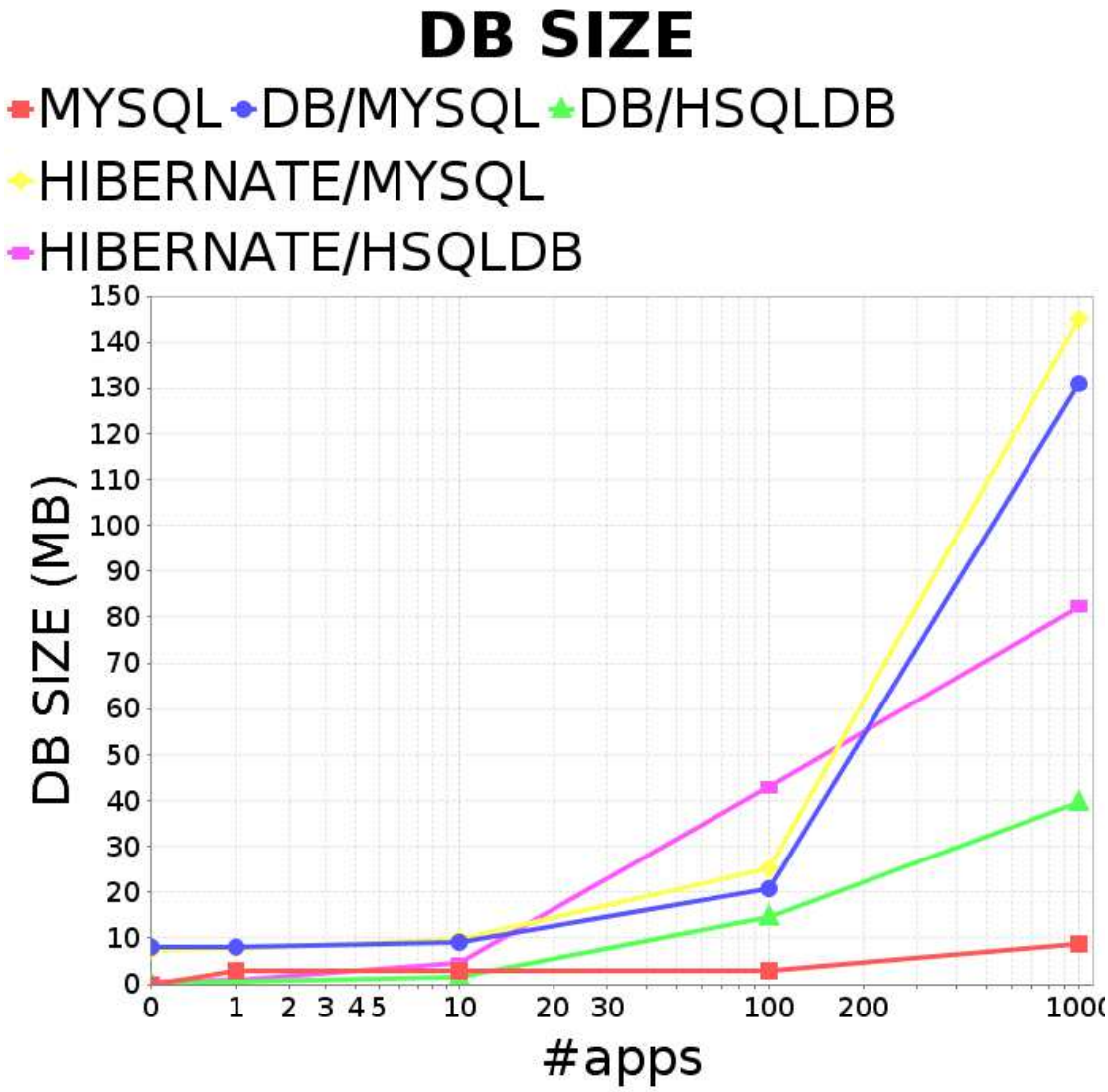

Figure 27 - Graph indicating the database size related to the amount of applications stored for different combinations of CDO store types and DBs as well as for the previous version of MDDB

To this end, the results depicted in Figure 27 do not bring about any surprise but they are more than expected.

The CDO combination that seems the best according to the DB size aspect is the one exploiting a HSQLDB for a DB type of store. In fact, the use of an SQLDB for any type of CDO store leads to a reduction in the database size due to the way SQLDB stores data (in a plain log file) with respect to a normal relational DB.

It can be also easily understood that the use of DBStore, leads to a smaller DB size with respect to the use of a HibernateStore for the same type of database. This seems to be normal as a DBStore adopts a simplified and straightforward (default) mapping of meta-model elements to database tables with respect to a HibernateStore. While a simplified mapping can be argued to lead to a greater DB size, this is not actually the case due to the fact that the complex mapping in the end requires the storage of additional data to accommodate for this complexity.

Finally, we should mention that the difference between these two types of stores is greater when a HSQLDB is exploited with respect to a MySQB DB. This should also be related to the way data are stored in a HSQLDB such that the savings can be multiplied when less amount of data have to be stored.

Based on the above results, it can be concluded that if the PaaSage developers need to adopt a higher-level manipulation of models via CDO technology, then clearly a DBStore with an underlying HSQLDB can lead to a smaller DB size than any other combination of CDO store type and DB. Otherwise, a direct manipulation over a relational DB should be the best possible choice due to the various advantages that are brought by the use of mature technology for relational database management.

7.1.2 First Query Experiment - Successful Deployments

For this and the next experiment, we attempted to reach a DB size which is close enough to the one with respect to the previous MDDB version in order to be fair with the comparison. This has led to the consideration of just 1000 applications for a CDO-based MDDB version compared to 100000 applications for the previous MDDB version. Then, we started performing the experiments but quite soon realized that the use of HSQLDB as a DB backend leads to major performance problems with such amount of applications. In fact, even one query performed by a single thread took a significantly unsuitable amount of time to complete. After doing some experimentation, we concluded that the DB(Store)/MySQL combination of CDO is the one which enables to perform the required testing due to its normal query answering time. To this end, only this combination was exploited in the end for the two remaining experiments along with of course the previous version of MDDB.

The experiment results over the first query for an increasing number of concurrent users, each with 1 second sleep time, are depicted in Figure 28. Here we can clearly see that the previous MDDB version is way faster. This can be justified according to the following reasons: (a) the respective query for the new MDDB version involves additional joins of tables; (b) the amount of data to be returned are way more for the new MDDB version due to the way the objects to be returned are represented; (c) the data in the case of the new MDDB version have to be transformed from the level of raw data to the level of objects (in models). In this respect, the new MDDB version can be considered to have better and acceptable performance only when the amount of concurrent users is less than 100 (good values are obtained for 50 users and even better for 10 concurrent users).

For both MDDB versions, the behaviour is linearly increasing which seems to be a logical and well-expected and accepted result.

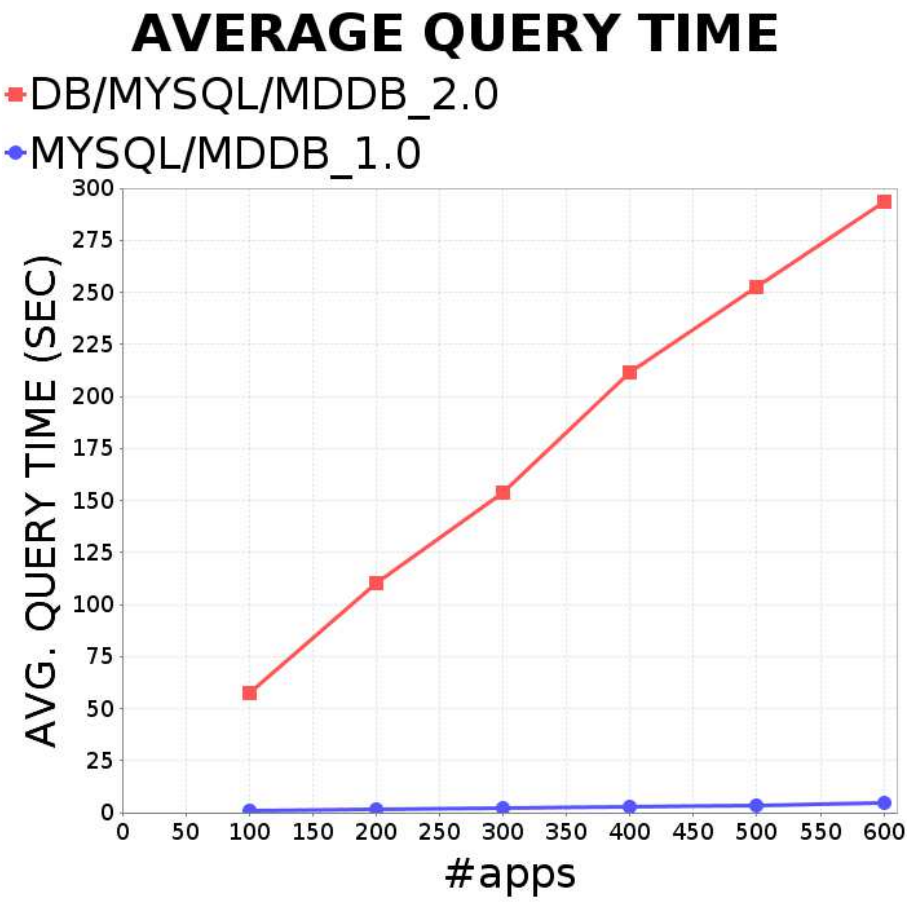

Figure 28 - First query experiment results for DB/MySQL and previous MDDB version

7.1.3 Second Query Experiment - Most instantiated VM Type

In the second query experiment, we were expecting that the overhead for the transformation of results would be lower as just one result has to be returned. Based on the rationale in the beginning of the previous sub-section, only the DB/MySQL CDO realisation of MDDB and MDDB's previous version were only considered. The respective experiment results are shown in Figure 29.

By comparing this query results with the previous one, we can clearly see that the gap between the new and the previous MDDB version is a little bit closed but not so much. This gap closing is mostly related to the fact that just one results has to be returned. A great gap closing could be achieved if the same amount of joins were performed for both MDDB versions which is not the actual case. In fact, by inspecting Section 6.1 in D4.1.1, we can clearly see that we need to access just one table for the previous MDDB version. On the other hand, for the new MDDB version, we have to join two tables.

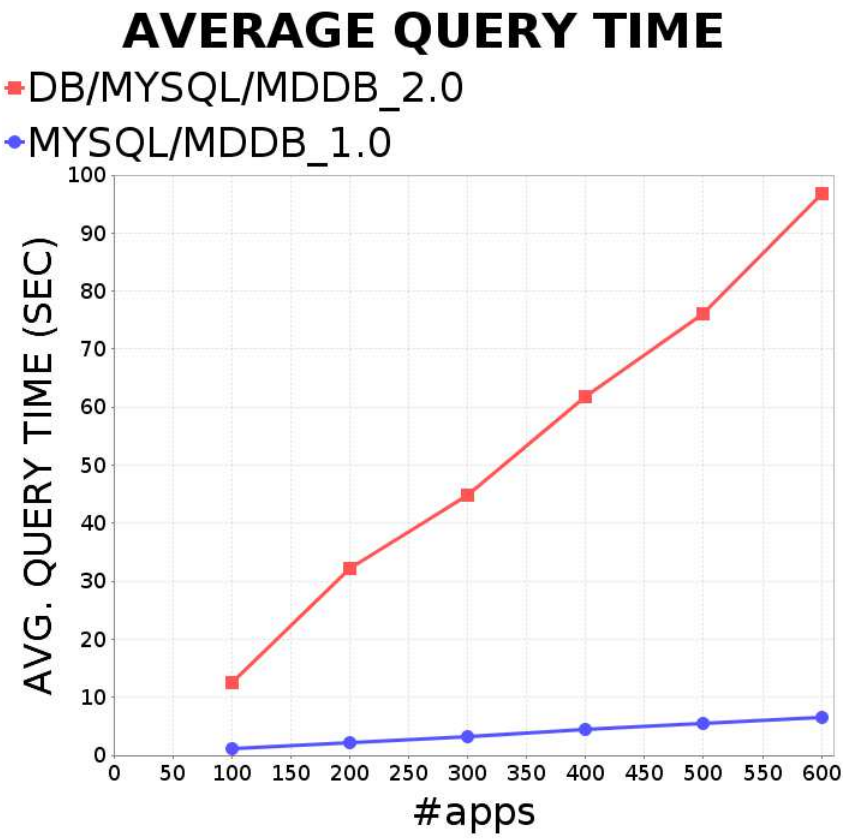

Figure 29 - Second query experiment results for DB/MySQL and previous MDDB version

7.1.4 Overall Conclusion

Based on the results of all the experiments, we can conclude that the best realisation option for the new MDDB version is the one which exploits a DBStore as well as MySQL as the underlying database. Although this realisation option leads to greater database size, it seems to be the sole alternative for a good query performance over a medium-sized MDDB/CDO Repository. To this end, if the PaaSage prototype is to offer good and acceptable performance levels supporting a great amount of customers and applications (i.e., to become a real product operating under real world circumstances), then the current realisation option in the form of HIBERNATE/HSQLDB should be replaced with the DB/MySQL one.

7.2 KB Evaluation

The purpose of this evaluation was to assess the new version of the REST-based implementation of the KB based on the same experiments that were conducted for the previous KB version. In this sense, it will be possible to compare the new assessment results with the previous ones in a fair manner in order to reach safe conclusions on whether the new KB version is better than the old one.

The same experiment was conducted over a different content of the MDDB each time. This experiment involves 100 concurrent users issuing 100 queries with a time space between each query request equal to 1 second. Each user issues 4 different types of queries to the REST-based KB service:

a. the first query involves getting the applications matching a certain application;

b. the second query involves getting the components matching a certain component;
c. the third query involves getting the best deployments for a certain application;
d. the fourth query involves getting the best deployments for a certain component.

The experiment was conducted in a small and medium-sized MDDB. The small MDDB configuration contained 5 applications with 4 execution contexts each while the big MDDB configuration contained again 5 applications but now with 400 execution contexts each.

The results of the experiment for the small MDDB configuration are shown in Figure 30. Compared to the respective results in D4.1.1 [1], we can clearly comprehend that:

- The query time is now much better. It went down to almost 1 second in contrast to the previous version which was around 1.8 seconds.
- There is no initial peak in response time as with respect to the case of the previous KB version. On the contrary, we can see a more or less stable behaviour with some obvious improvement in the end when the user load becomes lower. In fact, the behaviour of the previous KB version was different even in the very end as the query response time tended to become a little bit increased.

From the aforementioned presented results, we can conclude that the new KB version is better in case of a small-sized MDDB. It has a better behaviour and a better query response time. This can be justified by the fact that while the size of query results to return is small, the compact fact representation scheme of the new KB version leads to improving slightly the query results size with respect to the previous KB version. In this sense, while the actual time needed to perform the query at the server side will be similar, the actual transportation time for the results is reduced in the case of the new KB version. As we will see later on, the fact representation scheme leads to greater savings in transportation time in the case of a medium-sized MDDB.

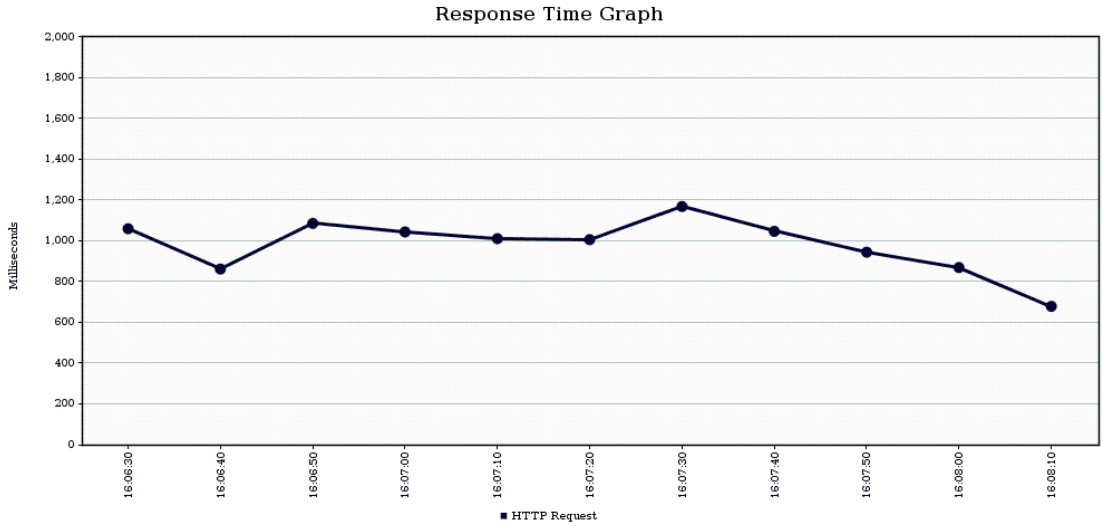

Figure 30 - Experiments results for KB over small-sized MDDB

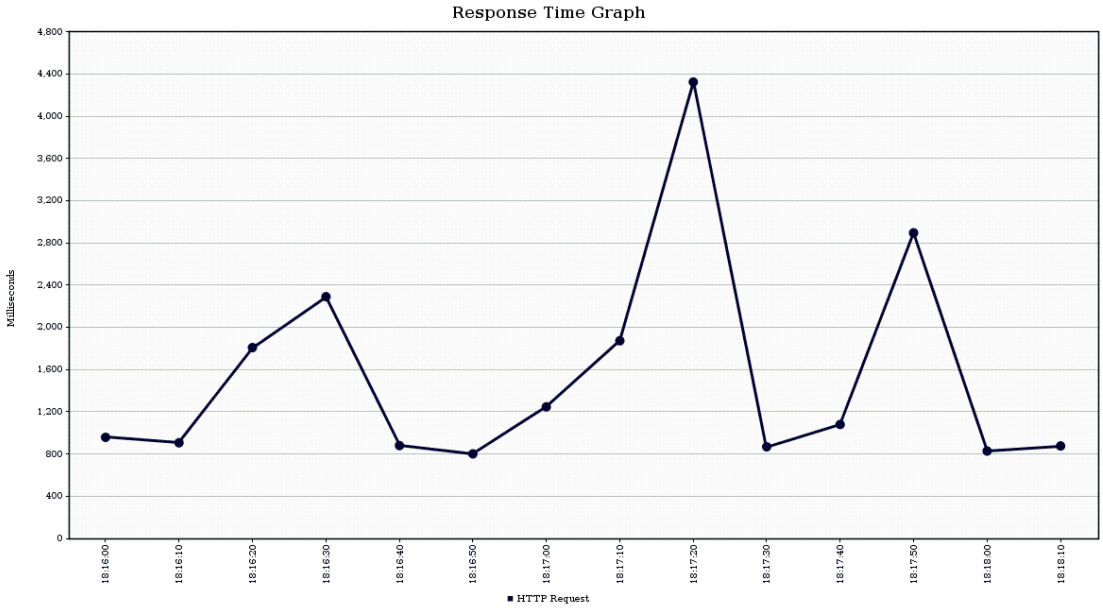

Figure 31 - Experiments results for KB over medium-sized MDDB

The respective results for the experiment on a medium-sized MDDB are shown in Figure 31. Now, the behaviour and performance of the new version of the KB not only improved but much better and higher, respectively, with respect to the one exhibited by the old version. The great savings on query time can again be attributed to the compact representation of facts which leads to a much shorter query result transportation time. In this sense, the new version of KB can eventually be characterized as far better than the previous ones, thus constituting a significant improvement. Its sole disadvantage is the fact that the user/developer has to perform separate queries on the CDO Repository in order to obtain the respective objects mapping to the CDOIDs returned as the original KB query results. However, we cannot deem this as critical due to the fact that, depending on the respective user/developer task, the user/developer might not need to expand the compact fact form or expand it only for a minimum amount of results. In this sense, the performance exhibited will always be better in the end and of course will enable the new version of KB to address an increased number of concurrent users.

7.3 Evaluation of PaaSage Social Networking Platform

In this section we describe the user evaluation of the PaaSage professional network, as well as the design and deployment of an application model using the platform. During the iterative process of designing the user interface several expert-based evaluations were carried out in group sessions. To obtain additional feedback from non-experts, two additional user-based evaluation experiments were designed and carried out involving potential users. This research work does not involve identifiable human material or data and user privacy is ensured in all cases.

7.3.1 Evaluation of interactive prototype through free exploration

This evaluation experiment aimed to collect subjective results rather than performance metrics. It involved a different set of 12 users who, after a brief introduction to the available facilities, were asked to use the interactive prototype [34, 35] exploiting the free exploration method of the Thinking Aloud protocol [36] and complete a

questionnaire in order to rate and comment their experience. Users were recruited through the PaaSage EU project consortium, based on the expertise criteria that all participants should be software developers with at least some expertise in social networking or cloud computing. The interactive prototype to be evaluated included all the necessary functionality for browsing through models and components, viewing detailed model information, and engaging in social activities, such as following and messaging a user, joining a group and posting questions to the group. The questionnaire comprised four sections:

a. background information (sex, age, expertise),
b. overall system usability according to the System Usability Scale (SUS) questionnaire [37],
c. assessment of specific system features (registration, finding a model, model information, community features) rated on a five point Likertscale and
d. additional information, where participants were asked to identify the three features that they liked most, the three features that they disliked most, and provide additional comments.

Statistics regarding the participants' gender, age, social network expertise, cloud computing expertise, and IT expertise are provided in Table 3.

Gender		
Male	8	66,7%
Female	4	33,3%
TOTAL	12	100,0%

Age		
<30	7	58,33%
30 - 40	5	41,67%
TOTAL	12	100,00%

Social network expertise		
Very high	2	16,6667%
High	6	50,0000%
Medium	2	16,6667%
Low	0	0,0000%
Very low	2	16,6667%
TOTAL	12	100,0001%

Cloud computing expertise		
Very high	3	25%
High	4	33,33%
Medium	5	41,67%
Low	0	0,00%
Very low	0	0,00%
TOTAL	12	100,00%

Software development expertise		
Very high	1	8,33%
High	6	50,00%
Medium	5	41,67%
Low	0	0
Very low	0	0
TOTAL	12	100,00%

Table 3 - Participants' statistics

As shown in Fig. 32, the overall SUS score for the professional network prototype (section B of the questionnaire) was 68.5 (a SUS score above 68 can be considered above average), while the SUS score from expert users was 77.8 and the SUS score from users with medium expertise was 55.5 (below average).

Rating of the individual network features (section C of the questionnaire), as shown in Fig. 33, indicated that users encountered difficulties in finding specific models and viewing their information. As explained by additional comments provided by some of the users, an important shortcoming was that only a small number of models were

included in the prototype and only one model included rich information for users to view.

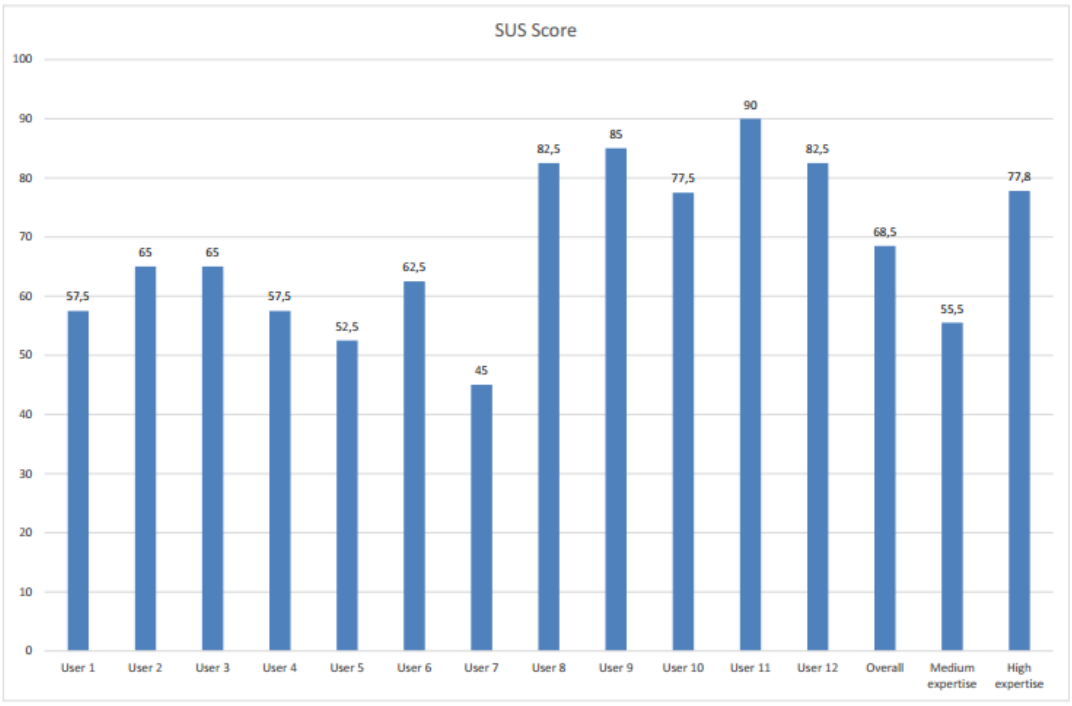

Figure 32 - SUS scores

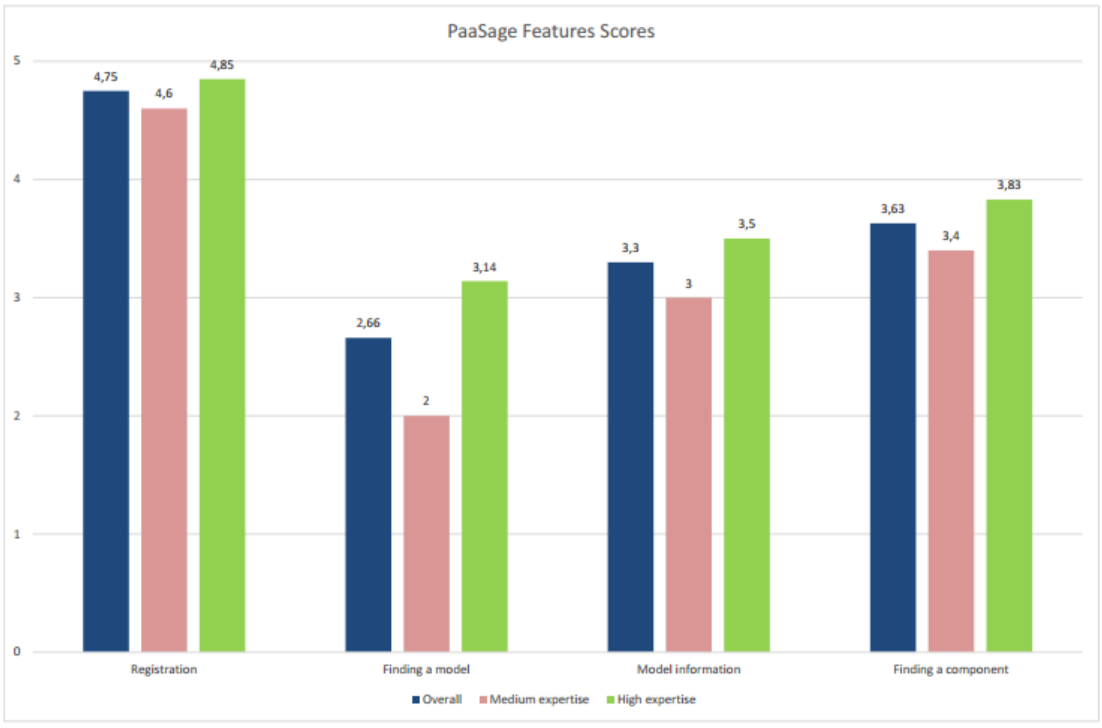

Figure 33 - Scores of the individual PaaSage features: registration, finding a model, model information, finding a component

Analysis of user response in section D of the questionnaire (most liked and most disliked features) revealed that users most commonly favoured the user interface of the network, the potential to share models and components with experts in the field and the ability to discuss with others and participate in groups of users with similar interests. On the other hand, users' responses regarding the features they disliked most indicated that medium expertise users would prefer an introductory video, a user guide, or tooltips in order to help them understand how the site works. Furthermore, most users commented on functionality that was missing or partially implemented, such as editing user profile, searching for friends, resetting password, and filtering model runs. Finally, users pointed out the lack of content as a problem that negatively affected their experience with the prototype. In summary, evaluation of the interactive prototype indicated that users liked the current form of the user interface as simple and professional and that they highly appreciated the facilities provided by the social network including model sharing, discussions and networking with users of similar interests.

Most of the negative comments and ratings are a result of the small size of the current content and evolving implementation. This was due to the nature of the experiment: aiming to receive as many comments and suggestions as possible for functionality that users desire to see included, they were asked to freely use the prototype, vocalize their thoughts and fill-in the evaluation questionnaire, rather than been taken through specific scenarios in an observation experiment

7.3.2 Evaluation of prototypes through scenarios and interviews

7.3.2.1 Methodology

The third evaluation experiment involved 15 participants who were guided through the interactive prototype of the PaaSage social network using specific scenarios. They were interviewed on their requirements and feedback following a semi-structured interview approach [38] that included both open-ended and close-ended questions. The evaluation session was carried out via Skype. Participants were recruited through European companies and organizations associated with the PaaSage EU project. They were either developers or operations staff, thus within the target user groups of the PaaSage social network. Participants were not users of the PaaSage platform; some however were familiar with the project's goals and objectives. Before the experiment, each participant was requested to fill-in a background information form and was sent an informed consent form, explaining all the recording and anonymity-ensuring procedures. A more detailed analysis of the participants' profile is presented in Table 4.

Gender			Age		
Male	12	80%	<30	6	40,00%
Female	3	20%	30 - 40	9	53,33%
TOTAL	15	100%	Undeclared	1	6,67%
			TOTAL	15	100,00%
DevOps expertise			**Social networking expertise**		
0-1 years	3	20,00%	0-1 years	1	6,67%
1-2 years	1	6,67%	1-2 years	0	0,00%
>2 years	11	73,33%	>2 years	14	93,33%
TOTAL	15	100,00%	**TOTAL**	15	100,00%
Professional expertise*			**Familiarity with PaaSage objectives**		
Software Engineer	13	86,67%	Limited	5	33,33%
Researcher	6	40,00%	Medium	4	26,67%
Solutions architect	2	13,33%	High	6	40,00%
Other	1	6,67%	**TOTAL**	15	100,00%

Table 4 - Participants' profile

Each interview session was structured as follows:

1. Introduction to the purpose of the evaluation experiment and the procedures that will be followed;
2. Video recorded consent to participate in the experiment;
3. Their requirements from DevOps platforms and social networking sites;
4. Use of the PaaSage social network through given scenarios;
5. Feedback regarding the PaaSage social network;
6. Debriefing.

Upon completing these sections, the interviewer read his/her notes to the participants and asked them (if needed) to clarify notes or add more comments prior to ending the session. All sessions were audio recorded. To ensure anonymity and data safety, each user received an ID with which he/she was referred to, while all documents and recordings of the participant were password protected.

7.3.2.2 User requirements

Questions targeting users from DevOps and social networking environments aimed at collecting information about which specific DevOps and social networking platforms each participant uses, how frequently they use them, features that they like or dislike in these environments, and features that are missing. When a participant was not a DevOps or social network user, they were asked why they were reluctant to use them. In all cases, the participants' willingness to try a new environment was explored both in general and in particular for an environment that would combine both DevOps and social networking features.

In relation to DevOps platforms, most of the participants were GitHub (80 %), GoogleCode (33.3 %), StackOverflow (33.3 %), or BitBucket (26.7 %) users. Users reported that the features they liked most about the DevOps environments they use

were: (1) related to developer activities: issue and activity tracking, bug reporting, viewing project statistics and following specific projects; and (2) relevant to community activities around the projects: user collaboration, ranking and rating, commenting. Features missing from these platforms include instant messaging, project management facilities, personalization characteristics (e.g., suggestion of possibly interesting projects according to each user's profile).

In relation to the social networking platforms, most of the participants were Facebook (80 %), LinkedIn (73.3 %) and Twitter (60 %) users. The majority of users reported that they use those platforms for personal and professional activities. Professional activities include information retrieval, connection with other professionals, seeking job opportunities, and promotion of oneself by making visible one's work. Features that users favored include communication with other people, fast information retrieval, linking with other individuals with similar interests and joining a large community of users. A major concern reported by nearly all participants is data privacy and security. Furthermore, some users reported that they were annoyed by e-mail pushing regarding activities carried out in the network.

Participants were asked if they would be willing to use a new platform that combines some DevOps features with social features, as well as what requirements they would have for such a platform in order to use it and prefer it in comparison to the other platforms that they use. All participants except one (who was not a developer) were positive to the idea of using such an environment. Several highlighted the fact that currently they have to use several different social networks to communicate with experts in their field. Features that the platform would have to deliver include privacy control, source code sharing and management options, good search and filtering mechanisms, user collaboration and communication facilities, as well as personalization features (such as the suggestion of potentially interesting people or projects according to the users' interests and current content that they are viewing).

7.3.2.3 Exploration through scenarios

Following the initial interview, participants were asked to explore the PaaSage network using a specific scenario that involved logging in, locating a specific application model (the SPEC jEnterprise2010 model), viewing its information, and selecting a specific execution of the model (the most cost-effective execution of the model on a specific cloud) to view its configuration. Each participant was then asked to visit the profile page of a specific model contributor, add him to the users he/she follows and view other models that this network user has contributed. Finally, each participant was asked to find a specific group in the PaaSage social network community, become a member, and post a question to the group.

During this process, participants were instructed to share their screen through Skype so that the interviewer could guide them through the scenarios and also observe their interaction with the system. The analysis of users' interaction with the system resulted in the identification of specific problems, which in most cases were also confirmed by the participants during their feedback elicitation process. For instance, issues that were identified included the visibility of certain UI elements, incomplete implementation of specific platform features (e.g., search), lack of integrated model editor, use of inappropriate labels in some cases, and lack of instructions or help for the more complicated pages.

7.3.2.4 User feedback

After going through the scenario, participants were interviewed regarding their opinion of the PaaSage social network. They were asked to rate on a scale of 1–10 how easy it was to find a specific model and if the model information was satisfying for them. For rates lower than 8, they were asked to identify what posed difficulties in their interaction and what they did not like about the specific network features. Model-finding received an average rate of 9 (standard error of the mean: 0.31), while model information was rated 8.4 on average (standard error of the mean: 0.34), as shown in Fig. 34.

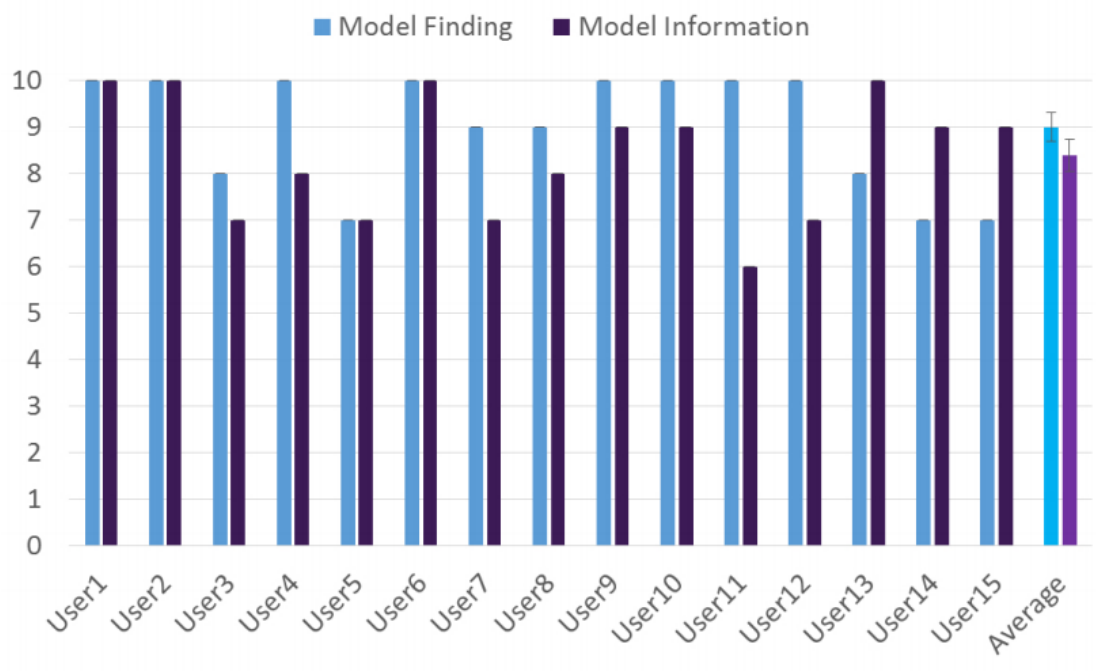

Figure 34 - Users' rating of model finding and model information

The lack of a fully implemented search facility was reported as the feature that prevented users from easily locating a specific model, since six of the users tried to locate the model through search and not through navigation on the available content. Additional information requested for the model page was: specific instructions on how to deploy a model, a graphical representation of the model, and to what percentage each contributor participates in the development of a model. Furthermore, some users found that due to the amount of information on the model page they needed additional time and instructions to view and fully understand it.

Next, users were asked to identify DevOps or social networking features that they would like to have seen in the network. DevOps features that were requested (Fig. 35) included code sharing, software project management facilities (e.g., activity tracking, versioning, milestones' management), instructions and scripts for model deployment, application binaries, and direct model execution. The few social networking features that were suggested were: messages with multiple recipients and attachments, chat facility, flagging of discussions as interesting to follow.

Figure 35 - User requests for DevOps features to be included in the PaaSage social network

Users were asked to identify up to three most-liked and three most-disliked features. Most-liked features included (Fig. 36): the integration of social features in a development environment; the use of charts and statistical information to represent data; the detailed model information that could be retrieved and the model execution histories; the automatically-generated hints provided by the network as replies in discussion topics; the concept of sharing one's models; the overall UI design; the direct connection between projects and users, and the user profile tags that allowed them to find users that would be interesting to connect with.

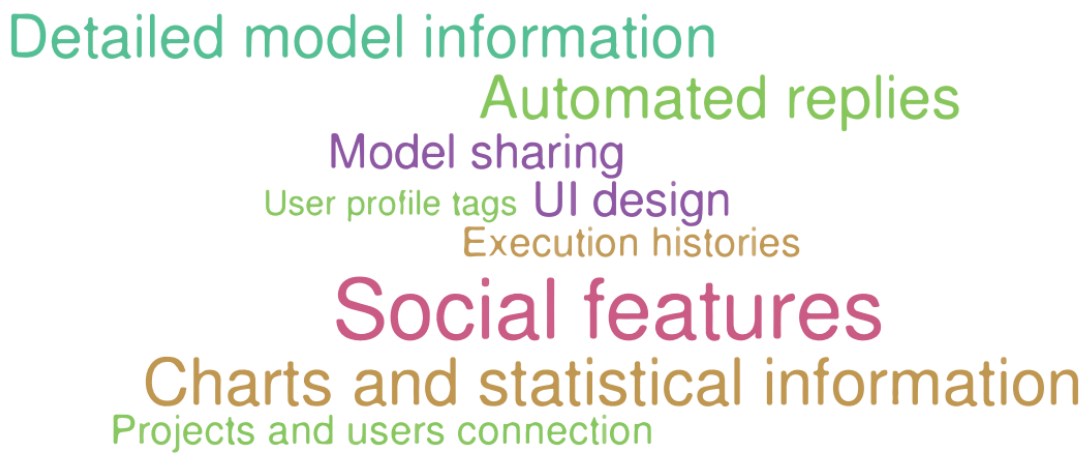

Figure 36 - PaaSage features that were mostly liked by the users

Most-disliked social networking platform characteristics included aesthetics issues (e.g., fonts, appearance of error messages, responsiveness); low visibility for certain components (e.g., model filters, suggested groups); extensive information in the model page; the lack of fully implemented search facility, and the current representation of models through XMI files. While there was mostly consensus between participants on most-liked features, most-disliked features were raised with low frequency (usually one -different- issue per participant), while several participants provided only two disliked features.

Wrapping up the interview, participants were asked if they would be willing to share their models in a social networking platform like PaaSage. For positive answers, they were further asked what benefits they thought it would bring to community members, as well as what kind of models they would like to see in such a platform. For negative answers, they were asked why they were sceptical. All participants provided a positive answer; some participants however expressed concerns regarding privacy

issues for sharing source code and application models, especially in the case of commercial projects. The foreseen benefits for the users relate to knowledge sharing, learning by example (e.g., starting from existing models and adapting them), finding resources related to one's interests through the concepts of similar models and model contributors, as well as network building with field experts. Regarding the type of models that users would like to find in the PaaSage social network most participants suggested well-known open source projects and applications, while some users suggested specific models related to their interests (e.g., from the IoT, Big Data area, and NoSQL datastore space).

A common request of most participants was support for code sharing. In most instances we had to explain that our platform managed application models not their software implementations, which can be hosted independently on platforms such as GitHub. That said, there is an inter-relationship between the model and the software implementation of an application that we must take into account: A specific version of the model corresponds to a specific version of the code, and a new software release may necessitate a change in the model representing the application. An important reason for such coevolution is that a specific execution history should refer to the code used in the deployment that produced it (as performance obviously depends on the application version used). There is thus a need for our platform to maintain model versions linked to references in code versions (when possible) in code-hosting platforms, which is a topic of ongoing work.

8 Discussion

8.1 Work planned for next year

While the core functionality of MDDB with respect to model management is considered as complete, there is still some work to be performed for the last year of the project within this WP according to the work plan that has been developed. This work is related to further support to standards, to the finalisation and cooperation of the SN with some PaaSage platform components, the promotion of the security solution [4] to the whole PaaSage platform and to the further updating of KB and its possible cooperation with other PaaSage components. These four directions of work are now analyzed in the following four sub-sections.

8.1.1 Support to Standards

With respect to the CERIF-to-CAMEL transformation tool, future work will focus on integrating this tool with the social network portal, to enable direct import of CERIF documents in CERIF XML exchange format into the MDDB, and further extending the scope of CERIF concepts covered by this transformation tool. In particular, concepts, such as DataCenter, Credentials, CloudCredentials, UserGroup, Resource and ResourceGroup, will be supported. Furthermore, the reverse transformation (CAMEL-to-CERIF) will be developed, which will allow generating a CERIF description for a certain organization from the respective CAMEL organisation model stored in MDDB so as to support use cases where an organization would like to model its structure in CAMEL and then generate CERIF models for dissemination or other purposes (e.g., create a CERIF repository with initial content to be later on enhanced).

Future work with respect to the XACML/policy editor will focus on its integration with the CDO Repository/MDDB in order to allow the generated XACML policies to be reflected on the organisation models stored in this repository. In this respect, the MDDB organisation models will always be up-to-date. To realize this integration, the XACML-to-CAMEL transformation code is planned to be realised, possibly in an ad-hoc in-memory way due to the simplicity of the mappings between the respective two meta-models.

The TOSCA transformation code based on ATL will be updated in order to cover all relevant aspects of TOSCA. It will focus mainly on the complete coverage of non-functional aspects which will lead to the further coverage of the requirement meta-model. As such, the capability to specify as well as transform all possible types of requirements (i.e., deployment, SLO and optimisation requirements) from TOSCA to CAMEL will be realized.

While WS-Agreement support has been planned, the resources available for the last year are limited. To this end, its realisation is currently on hold and resources are searched within the consortium to resume it.

8.1.2 Security

The security solution [4] analyzed in Section 4 has been already developed but its integration with the remaining PaaSage platform is missing. In addition, testing the solution according to particular scenarios, such as those provided in Section 4.4, needs to be conducted to possibly fix any type of security vulnerabilities that might be detected. To this end, the work planned for year 4 will focus on the following actions:

- integrate into the SN the authentication functionality to enable the proper authentication of the SN users
- integrate the solution with the platform API developed by CETIC - this requires further extending this API in order to exploit the PEP/PDP component towards authorizing user requests
- develop some short guidelines for PaaSage developers on how to exploit the security solution facilities in order to enable the secure management of models
- optionally support the external authentication of users via different platforms (e.g., facebook)
- extensive testing the security solution

It is important to stress here that the integration of the security solution in the platform is quite significant as it can provide an added-value feature to the PaaSage platform enable it to be distinguished from other similar or equivalent offerings.

8.1.3 KB

The re-engineering of KB has led to a significant improvement which was evidenced from the thorough evaluation results presented in Section 7.2. However, while the added-value knowledge that can be derived can be considered as satisfactory, we believe that we can further strengthen and showcase the KB's power through the design and specification of new rules which can provide even further support to one or more PaaSage components. To this end, the cooperation between components like the Reasoner in the Upperware module of PaaSage can be pursued. In the case of the Reasoner, the cooperation with the KB can actually lead to further filtering the provider space according to best deployments that have been found for similar or equivalent applications and components with respect to the application at hand, thus speeding up the solving time of the Reasoner.

Another candidate component for such a cooperation could be the Social Network. This cooperation could be incarnated through the derivation of useful facts with respect to the operation and exposed functionality of the SN. Examples of such additional facts that could be derived are: (a) discovery of successful applications, i.e., applications which always have their requirements satisfied; (b) identification of experienced modellers which provide suitable models for deployment which tend to satisfy the requirements posed on them; (c) discovery of successful scalability rules which always enable applications to bypass the problematic situations in which they have been entered - such rules could be exploited as templates when designing the camel model for a new application; (d) most reliable cloud platforms offering services which are always involved in successful application deployments.

8.1.4 Social Network

The Social Network can be considered as more or less complete. Due to decisions reached in the previous consortium meetings as well as by relying on the fact that the SN is one of the main entry points for users in the PaaSage platform, an additional functionality towards initiating the deployment of applications through this platform has been planned. The respective UI work has already been performed. What remains to be performed is the integration of the underlying deployment initiation code with the platform API developed by CETIC which is currently under development. Moreover, some UI improvements in the SN will be performed according to the feedback obtained from the evaluation.

9 Conclusion

This deliverable has reported the current status of the PaaSage MDDB and the Social Network and the respective work that has been performed in WP4 during the time period M19-M36. This report has indicated that the work plan indicated in D4.1.1 [1] for this 18 months period was faithfully followed with the sole exception of some tasks that have been abandoned due to the change of focus and the limited amount of resources left. This is evidenced by the fact that the roadmap for the remaining work during the last year of the project is limited and focusing on completing the promised functionality of a very small number of tasks. Moreover, this roadmap concentrates on providing some features which will provide some added-value to the project, such as cross-platform security and sophisticated integration between components. All this analysis indicates that the WP is not only on track but has also worked towards better promoting the PaaSage platform through the activation of new features.

10 BIBLIOGRAPHY

1. Kritikos, K., Korozi, M., Kryza, B., Kirkham, T., Leonidis, A., Magoutis, K., Massonet, P., Ntoa, S., Papaioannou, A., Papoulas, C., Sheridan, C., Zeginis, C.: D4.1.1 / D4.3.1 – Prototype Metadata Database and Social Network / Prototype of Metadata Integration Extension (March 2014).
2. The PaaSage Consortium. D1.6.1—Initial Architecture Design. PaaSage project deliverable. Oct. 2013.
3. Rossini, A., Nikolov, N., Romero, D., Domaschka, J., Kritikos, K., Kirkham, T., Solberg, A.: D2.1.2 – CloudML Implementation Documentation. Paasage project deliverable (April 2014).
4. T. Kirkham, B. Kryza, K. Kritikos and P. Massonet, “Security Enforcement for Multi-Cloud Platforms - the Case of PaaSage,” in CF ’15. Pisa, Italy: Elsevier, 2015.
5. J. Domaschka, K. Kritikos, and A. Rossini, “SRL: A Scalability Rule Language for Multi-Cloud Environments,” in *Proceedings of the 2014 IEEE International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM ’14. Washington, DC, USA: IEEE Computer Society, 2014.
6. Jeffery, K., Houssos, N., Jörg, B., Asserson, A.. Research information management: the CERIF approach. *IJMSO* 2014;9(1):5–14.
7. Cloud Select Industry Group (C-SIG), “Cloud Service Level Agreement Standardization Guidelines,” Technical Report, 2014.
8. Nielsen J (1993) Iterative user-interface design. Computer 26(11):32–41
9. Mayhew DJ (1999) The usability engineering lifecycle. In: CHI ’99 Extended Abstracts on Human Factors in Computing Systems. CHI EA ’99. ACM, New York, NY, USA. pp 147–148. http://doi.acm.org/10.1145/632716.632805
10. Ferreira J, Noble J, Biddle R (2007) Agile development iterations and ui design. In: Agile Conference (AGILE), 2007. IEEE. pp 50–58

11. Stone D, Jarrett C, Woodroffe M, Minocha S (2005) User Interface Design and Evaluation. Morgan Kaufmann
12. Von Krogh G, Spaeth S, Lakhani KR (2003) Community, joining, and specialization in open source software innovation: a case study. Res Policy 32(7):1217–1241
13. Begel A, DeLine R, Zimmermann T (2010) Social media for software engineering. In: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research. ACM. pp 33–38
14. DiMicco J, Millen DR, Geyer W, Dugan C, Brownholtz B, Muller M (2008) Motivations for social networking at work. In: Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work. ACM. pp 711–720
15. Brusilovsky P, Kobsa A, Nejdl W (2007) The Adaptive Web: Methods and Strategies of Web Personalization, Vol. 4321. Springer
16. Tam KY, Ho SY (2005) Web personalization as a persuasion strategy: An elaboration likelihood model perspective. Inf Syst Res 16(3):271–291
17. Sundar SS, Marathe SS (2010) Personalization versus customization: The importance of agency, privacy, and power usage. Hum Commun Res 36(3):298–322
18. Nielsen J (1999) Site design. In: Designing Web Usability: The Practice of Simplicity. New Riders Publishing, Indianapolis. pp 166–167
19. Krug S (2006) The first step in recovery is admitting that the home page is beyond your control: Designing the home page. In: Don't Make Me Think!: a Common Sense Approach to Web Usability. New Riders, Berkeley, CA. pp 94–121
20. Swan K (2001) Virtual interaction: Design factors affecting student satisfaction and perceived learning in asynchronous online courses. Dist Educ 22(2):306–331
21. Lazar J, Preece J (2002) Social Considerations in Online Communities: Usability, Sociability, and Success Factors
22. Liu IF, Chen MC, Sun YS, Wible D, Kuo CH (2010) Extending the tam model to explore the factors that affect intention to use an online learning community. Comput Educ 54(2):600–610
23. Deterding S, Sicart M, Nacke L, O'Hara K, Dixon D (2011) Gamification. using game-design elements in non-gaming contexts. In: CHI11 Extended Abstracts on Human Factors in Computing Systems. ACM, New York, NY, USA. pp 2425–2428
24. Antin J, Churchill EF (2011) Badges in social media: A social psychological perspective. In: CHI 2011 Gamification Workshop Proceedings (Vancouver, BC, Canada, 2011)
25. Konstas I, Stathopoulos V, Jose JM (2009) On social networks and collaborative recommendation. In: Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '09. ACM, New York, NY, USA. Pp 195–202. http://doi.acm. org/10.1145/1571941.1571977

26. Garrett JJ (2011) The Elements of User Experience : User-centered Design for the Web and Beyond. Voices that matter. Pearson Education, Berkeley, Calif. New Riders London
27. Nielsen J (2001) Converting Search into Navigation. http://www.nngroup.com/articles/search-navigation/. Retrieved October 6, 2014
28. Budiu R Search Is Not Enough: Synergy Between Navigation and Search. http://www.nngroup.com/articles/search not-enough/. Accessed 8/2015
29. Ames M, Naaman M (2007) Why we tag: Motivations for annotation in mobile and online media. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '07. ACM, New York, NY, USA. pp 971–980. http://doi.acm.org/10.1145/1240624.1240772
30. Hughes J (2011) iPhone and iPad Apps Marketing: Secrets to Selling Your iPhone and iPad Apps. Que Publishing
31. Ho KK, See-To EW, Chiu GT (2013) How does a social network site fan page influence purchase intention of online shoppers: A qualitative analysis. Int J Soc Organ Dyn IT (IJSODIT) 3(4):19–42
32. Engine ESN. http://elgg.org/ Accessed 8/2015
33. Spurlock J (2013) Bootstrap: Responsive Web development. O'Reilly Media, Sebastopol, CA
34. Virzi RA, Sokolov JL, Karis D (1996) Usability problem identification using both low- and high-fidelity prototypes. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '96. ACM, New York, NY, USA. pp 236–243. http://doi.acm.org/10.1145/238386.238516
35. Catani MB, Biers DW (1998) Usability evaluation and prototype fidelity: Users and usability professionals. In: Proceedings of the Human Factors and Ergonomics Society Annual Meeting. SAGE Publications Vol. 42. pp 1331–1335
36. Jordan PW (1998) An Introduction to Usability. CRC Press
37. Brooke J (1996) Sus-a quick and dirty usability scale. Usability Eval Ind 189(194):4–7
38. Gillham B (2001) The nature of the interview. Research Interview (Real World Research) 6

11 Appendix

11.1 Model Storage Guidelines

11.1.1 Rationale

We need to find a way to store models such that we avoid: (a) repetition of information and (b) naming conflicts which can hinder the commitment of models through CDO transactions. Moreover, we desire to avoid the creation of very big models. In addition, we opt for model retrieval alternatives, apart from queries, to obtain the required models/information according to the way they are stored. To this end, we have examined various solutions and selected the one that is analyzed below. The solution comes in terms of particular guidelines that dictate the way the models are stored in terms of the structure of the CDO Repository and the name of these models/resources. These guidelines must be followed by both developers and UI modellers/users in order to avoid the above problems from actually occurring.

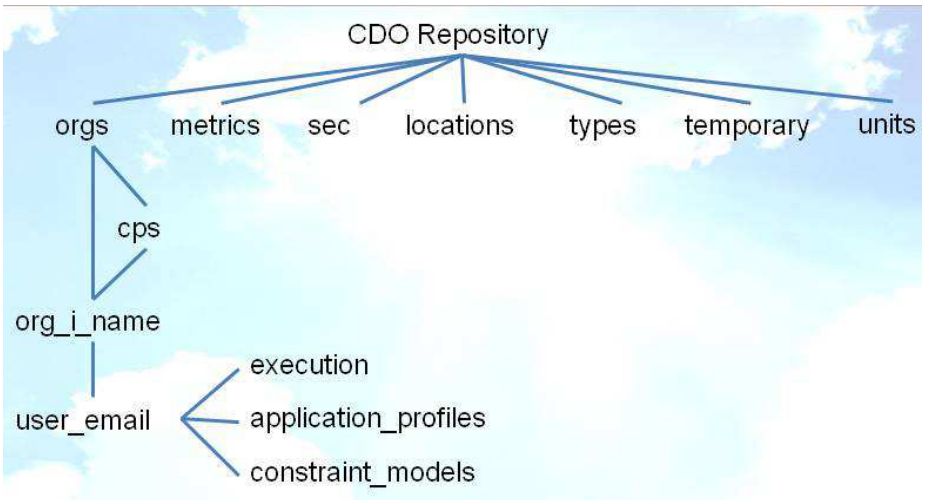

Figure 37 - The solution envisaged with respect to the structuring of the CDO Repository in resource folders

11.1.2 Solution

The solution, which is depicted in Figures 37 and 38, relies on (a) first identifying appropriate CDO folder paths (see *CDOResourceFolder* class) in order to store a model and (b) second identifying a proper name for the *CDOResource* containing the model. Both parts of the solution are interdependent as the scheme for the first filters the possible alternative schemes for the second. The solution envisaged dictates that a CDO repository should be organized in a particular way based on the type of information that needs to be stored. The respective sub-sections indicate the CDO repository organisation for each type of information.

11.1.2.1 Organisation / Cloud-Provider Specific Information

Here we consider all possible types of information that can be generated pertaining to a specific organisation, such as provider and organisation models. We should note that there is a differentiation between a cloud provider organisation and one that is not a cloud provider. Irrespectively of the organisation type, we envisage that there should

be a root CDO folder named as "orgs" which will contain the information related to all possible organisations independently of their type. In case that an organisation is not a cloud provider, then a respective CDO folder path must be created: "/orgs/org_i", where "orgs" denotes all organisations and "org_i" the name of a specific organisation in lowercase, which will contain all *CDOResource*s specific to this organisation. In case of a cloud provider organisation, the following CDO folder path must be generated: "/orgs/cps/org_i", where "cps" denotes all cloud providers. Then, for each particular organisation, we should create a *CDOResource*, to be stored in the generated organisation CDO folder path. This resource will store either an organisation model in case of a non-cloud provider organisation, or a CAMEL model which will contain the organisation and provider model of the considered cloud provider. Irrespectively of the organisation type, the name of the *CDOResource* should be equal to the name of the organisation at hand again in lowercase. Finally, we should also note that the Social Network (SN) is considered as another organisation with its own users (especially if those users do not belong to a specific organisation). To this end, we will create a *CDOFolderPath* with the following structure: "/orgs/sn" which will contain an organisation model indicating the users involved in the SN as well as information generated by these users (see next bullet).

Simple example: In case of AGH, we should create the CDO folder path "/orgs/agh" which will contain a *CDOResource* named as "agh" enclosing the organisation model of AGH. In case of Amazon, we should create a CDO folder path "orgs/cps/amazon" which will contain a *CDOResource* named as "amazon" enclosing a CAMEL model including the provider and organisation model of Amazon.

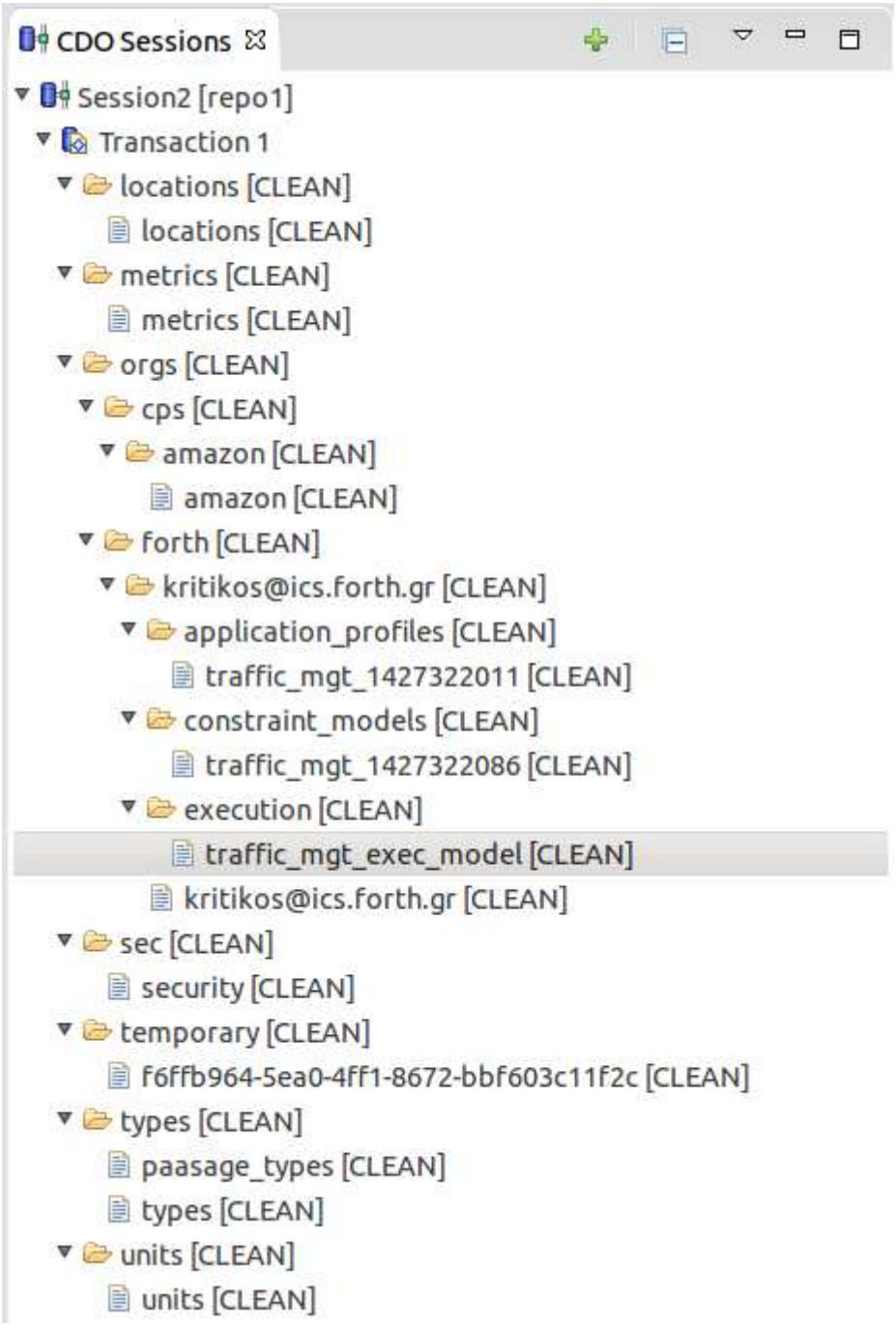

Figure 38 - A snapshot of the CDORepository showing both resource folders as well as respective resources contained in them

11.1.2.2 User Information

As we can have one or more users which generate information on behalf of an organisation (or the SN), we need to store this information, in terms of a set of models, in the CDO Repository at the appropriate place but below the *CDOResourceFolder* pertaining to this organisation. To this end, we need to create a sub-path per user on which these models should be stored. The complete path then should be: "/orgs/org_i/" + <user_email>. The models to be stored are the following:

1. *A CAMEL model*. This model should contain information about all the user applications, deployments, requirements, metrics and scalability rules. The name of the *CDOResource* to store the CAMEL model should be equal to the email of the user.
2. *Execution & Instance-based metric models*. As many executions can be performed per deployment model and many deployment models can be specified by a user, execution models will tend to be quite big, especially as they have to store various types of information, including measurements, SLO assessments and scalability rule triggers. Each execution context of an execution model should also be coupled with a metric model which will contain metric instances indicating specific metrics that are created in the context of the specific execution of the respective user's deployment model at hand to produce the respective measurements. To this end, it is dictated that a subfolder should be created inside the user's CDO folder which should be named as "execution ". In other words, this sub-folder will be available at the following CDO Folder Path: "/orgs/org_i/" + <user_email> + "execution ". This sub-folder will contain all user execution and instance-based models. As an execution model will be coupled with many metric models, we consider that one CAMEL model per execution model will be created which will contain this execution model and all instance-based metric models associated to it. This CAMEL model will be stored in a *CDOResource* named after the name of the encompassed execution model.
3. *Application profiles*. Each profile should be stored in a *CDOResource* named after the id of the application profile/configuration plus a timestamp. The *CDOResource* of the application profile should be placed in a subfolder named as "application_profiles" which should be available at the following CDO Folder Path: "/orgs/org_i/" + <user_email> + "application_profiles".
4. *Constraint Models*. Each constraint model should be stored in a *CDOResource* named after the name of the application on which the model has been generated plus a timestamp (here we have the case that many deployment requests can be requested for the same application so we need a timestamp information to distinguish between them). The *CDOResource* of the constraint model should be placed in a subfolder named as "constraint_models" which should be available at the following CDO Folder Path: "/orgs/org_i/" + <user_email> + "constraint_models".

Simple Example: Suppose that we need to accommodate for the storage of models for "user1" with email "user1@agh.pl" in AGH. Then, we will have to create the following CDO folder path: "/orgs/agh/user1@agh.pl". In this folder, we will put a *CDOResource* named as " user1@agh.pl " comprising a CAMEL model containing all applications, requirement models, metric models and scalability models of this user. In this folder, we will also create three main sub-folders, namely, "execution", "application_profiles", and "constraint_models". The two latter subfolders will store the models of the respective type (e.g., constraint model for the "constraint_model" sub-folder). On the other hand, the execution sub-folder will contain one or more

CAMEL models, each pertaining to an execution model and a set of instance-based metric models. For instance, an execution model named as "scalarm" will be put inside a CAMEL model along with its instance-based metric models and the *CDOResource* on which this CAMEL model will be stored will be named as "scalarm and will be situated in the following path: "/orgs/agh/user1@agh.pl/execution".

11.1.2.3 Metric Information

Here we put information that pertains to a basic taxonomy of properties and metrics. This will actually include the basic metrics that are by default supported by the platform. We do not consider here metric instance information which should be related to a particular execution model. The CDO folder path for this type of information should be the following: "/metrics". We also envisage that just one metric model should be stored in this path which should contain all this metric-related information. The name of the respective *CDOResource* containing this model could be equivalently named as "metrics".

11.1.2.4 Security Information

We consider here the storage of basic security information which can then be used for constructing security requirements and capabilities, including security controls, metrics and attributes. The CDO folder path envisaged is: "/sec". Again one (security) model will be stored here contained in a *CDOResource* named as "security".

11.1.2.5 Location Information

We should consider storing basic location information that can be exploited for, e.g., stating location requirements/constraints. We actually have here: (a) a taxonomy of physical locations that is being generated by following a standard for location description and (b) cloud-specific locations specific to a cloud provider which can be used via respective location requirements to couple a VM to a specific cloud. Cloud-specific locations are of course being updated for new cloud providers or for existing ones when there is a change in the internal location structure of a cloud provider. The CDO folder path envisaged in this case is: "/location". There one model will be stored containing all this location information. This model could be named as "location".

11.1.2.6 Type Information

We consider here information pertaining to value types and values which can be re-used and referenced by other CAMEL models, such as the case for the value type of a feature or metric or the value of positive or negative infinity. This information can also comprise types conforming to the paasage_types meta-model in WP3. The CDO folder path envisaged is: "/types". Two models should be stored pertaining to a value type repository and a paasage type model which should be contained in two *CDOResource*s named as "types" and "paasage_types", respectively.

11.1.2.7 Unit Information

Units are being used in the context of various CAMEL DSLs, including the metric and provider ones. To this end, as the taxonomy of units defined in CAMEL is not very big, it is envisaged that all possible units, to be re-used in other CAMEL models, must be defined just once in one particular unit model to be stored in a *CDOResource* named as "units" situated in a specific folder in the CDO Repository with the following path: "/units".

11.1.2.8 Temporary Information

Based on the current way PaaSage components communicate, we have the case that one component creates an output in terms of a model which is consumed as input by the next component in the execution order. In this sense, we can have two different types of models here: (a) normal: e.g., mapping to final models, such as deployment models and (b) temporal: mapping to incomplete models which need additional information to be filled in. The first type of models is already addressed by all the above cases. However, the second type is not. To this end, we propose a special CDO folder path for the storage of such models, named as "temporary" where we assume that the naming of the models to be stored is based on UUIDs to avoid name clashes (of course any other way which can avoid this problem is welcome). We should note here that it is the responsibility of the appropriate responsible component to finalize a model, store it in the appropriate CDO folder path with the appropriate name and delete the respective temporal model. We do not want to fill up the whole space devoted for the CDO Repository with the storage of these temporary models which does not enclose very important information for further processing or analysis.

11.1.3 Technical Details

A CDO (folder) path can be developed by constructing a tree of *CDOResourceFolder*(s). There are two ways a tree folder path can be constructed:

1. CDOResourceFolder folder = trans.getOrCreateResourceFolder(<folder_name>). In this case, we can construct both a root CDO folder as well as its constituing sub-folders. The trick here is the name of the folder. If it does not contain the "/" symbol, then the folder is put at the root level. On the other hand, if the name comprises one or more "/" symbols, then the folder will be placed at the tree hierarchy level matching the number of "/" symbols in the folder name. For instance, a name equal to "A/B/C" will create a folder C (level 2) and put it inside folder B (level 1) which is constrained in folder A (level 0 - root level).
2. CDOResourceFolder innerFolder = folder.addResourceFolder(<folder_name>). Here we explicitly generate a folder with name <folder_name> inside the CDOResourceFolder identified by the folder object. So, if the parent folder is root and named as "A", then if the generated folder's name is "B", this means that we have created a CDO folder path of the form: "A/B".

Each folder can contain also one or more *CDOResource*(s) where which should map to a particular model/EObject. There are again two ways to create a *CDOResource* inside a folder:

1. CDOResource res = folder.addResource(<resource_name>). Here we explicitly create the resource inside a particular folder. This means that the resource will hang from the respective CDO folder path pointed by the respective folder. For instance, if a folder points to the CDO folder path "A/B" and the name of the resource to be created is "ResA", this means that the full CDO path to this resource will be "A/B/ResA". This latter path can pave the way for using the next alternative:

CDOResource res = trans.getOrCreateResource(<full_resource_name>). Here we do not only create a resource but we also hang it at the appropriate CDO folder path. So following the previous example, if the full resource name is "A/B/ResA", then the resource will be inserted in folder "B" contained in root folder "A".

11.2 KB Technical Documentation

11.2.1 Introduction

The FORTH team has realized two different types of APIs that expose the functionality required to manage Knowledge Bases. The first API is in the form of a thick client called *StandaloneClient* which interacts directly with an internal KnowledgeBase Management Engine, called Drools, and as such takes the burden of executing locally all methods of the API. The second API has taken the form of a REST service which provides remote and authorized access to the respective methods. This second API is accompanied with a lightweight/thin client, called *RestClient*, which comprises particular methods corresponding in a one-to-one manner to the remote ones and actually leading to their execution. Both APIs comprise the same set of methods whose signature varies based on the technical differences between the two solutions. Such technical differences are mainly due to the need to pass information across the Internet for the second solution. As this information needs to be marshalled and unmarshalled, special classes had to be introduced wrapping the information to be transferred in an appropriate way. Moreover, JAXB annotations were actually exploited in order to specify the exact way and which part of the overall information will be encoded and decoded in the two main formats supported by the REST technology, mainly XML and JSON.

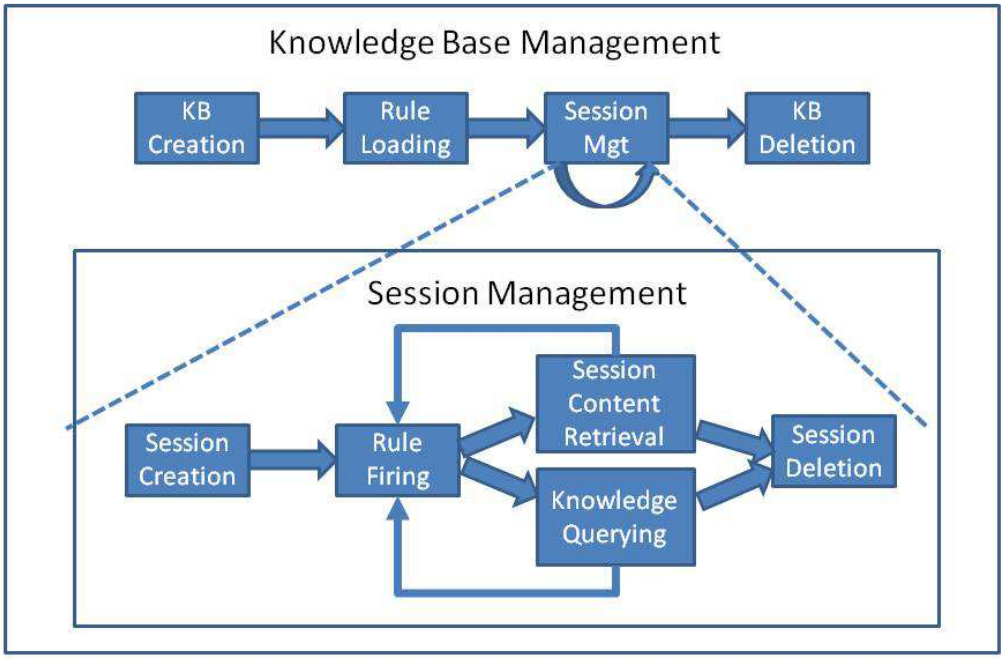

Figure 39 - The lifecycle of a KnowledgeBase and its sessions

11.2.2 KnowledgeBase Lifecycle

Both APIs share exactly the same functionality spanning the whole life-cycle of Knowledge Bases and of the respective stateful knowledge sessions that are created from them. As such, through these APIs, new knowledge bases can be created, existing ones can be updated with the addition of rules as well as be deleted when they are not needed any more. Moreover, stateful knowledge sessions can be created from knowledge bases facilitating the actual handling of rules, including their firing as a whole or in an individual manner leading to the production of new facts, the addition of facts to make particular rules enabled for triggering, and the exporting of all facts as Java objects. Such sessions can then be deleted when they are no more needed. A typical lifecycle of a knowledge base includes its creation, the loading of rules, the management of sessions and its final deletion. Thus, the lifecycle of a knowledge base includes internally a lifecycle of one or more stateful knowledge sessions. The latter lifecycle includes the creation of a session, the firing of rules, the addition and the retrieval of facts and the deletion of a session. The two lifecycles and their interplay can be seen in Figure 39.

The inclusion of the session lifecycle in the KnowledgeBase lifecycle designates that sessions are bound to the KBs that create them and cannot live independently from them. So, when a KB is deleted, then actually all the respective information is deleted, including all corresponding sessions, basic & user-provided rules and derived knowledge (bound to this KB). On the contrary, when a specific session is deleted, then just the knowledge derived from it is also deleted and the respective KB (including its loaded rules) remains untouched.

11.2.3 Exploitation Paths

The KnowledgeBase Management APIs offered can have multiple exploitation paths. We can outline some of the paths in the following:

- offering of design hints to modellers with respect to the deployment of components or whole applications (e.g., what were the best or successful deployments for a component & which were the bad that should be definitely avoided)
- explication of the performance variation for application or component deployments
- derivation of frequency of scaling for particular deployments
- determination of deployments with no scaling performance
- explication of performance variation across different deployments for a certain application based on its history or the history of similar applications
- discovery of similarity relations between applications and between components
- determination of the frequency of a component across applications of the same type e.g. indicating that for instance higher-frequency components should be always parts of such applications

- derivation of problematic VM offerings which lead to violation of SLOs most of the time (this is slightly different with respect to a bad deployment as in this case a problematic VM offering is consistently participating in bad deployments while a VM offering participating in just one or two bad deployments does not necessarily mean that it is problematic)

- derivation of availability variation for VM offerings (useful for the selection of offerings in deployment reasoning)

The above paths can be exploited by a Reasoner in order to further filter bad offerings or bad deployments and to fix a part of the problem by mapping it to the best deployments derived from the KnowledgeBase (of course this is meaningful mainly for components - if this is done for applications, then we actually have a complete solving of a problem through the selection of the best deployment according to the execution history of the application at hand or other equivalent or similar applications to this one). A model editor could also be enhanced through the KnowledgeBase (KB) as it could provide useful hints to modellers in order to guide them in the specification of deployment models for application (by, e.g., indicating which are the most suitable component candidates for an application or which are the recommended deployments for particular application components) or even requirements models (by, e.g., indicating what are those SLOs which are meaningful and cannot be always violated).

We should also highlight here that the knowledge currently derived by the KnowledgeBase relies on a set of basic rules. Such knowledge can be certainly boosted through the addition of rules by users of the social network derived from their experience such that a sophisticated KnowledgeBase for cloud computing can be finally produced. Obviously, users can also benefit individually by the offered API as they can also create their own private KnowledgeBases which can function over the execution history of their applications.

11.2.4 Usage Scenario

According to KB lifecycle, before adding rules or issuing queries, a user should make a respective KB on which he/she can load basic (those created by the FORTH team to produce derived, added-value knowledge out of the MDDB[15]) and user-specific rules. After this, then the user can create sessions from this KB through which he/she can add objects (e.g., ApplicationMatch(es), BestApplicationDeployment(s), etc.), fire all rules, obtain the derived knowledge in the form of a set of objects and run queries over this derived knowledge to filter the currently available results/knowledge space.

Based on the above analysis, a particular scenario that can be considered as the one that will be commonly and most often applied in the context of the PaaSage project is the following:

- The PaaSage user first creates a KB and provides it with a particular name (e.g., “myKB”). The user will load only basic rules to this KB as well as user-specific ones

[15] These rules are loaded a priori in the case of the Rest-ful API for each KnowledgeBase created. In case of the standalone API, these basic rules will be stored in a particular directory of the main code distribution and the user will be responsible of loading those rules (basic or user-specific) that interest him/her in the current exploitation context.

pertaining to the specification of queries focusing on the retrieval of the derived knowledge from any sessions created out of this KB (e.g., let's consider a Drools file which has a query named "applicationMatch" which will be used for querying the KB's session(s) to obtain matches for a particular application).

- The PaaSage user creates a session named "mySession" out of the "myKB" knowledge base. The user does not need to add any objects to this session as all the low-level knowledge that is requested from the basic rules is fetched from the MDDB (this capability to retrieve MDDB information will be analyzed in a later sub-section).

- The PaaSage user fires all rules in order to derive new knowledge via the session "mySession" previously created.

- Finally, the user issues the "getApplicationMatches" query on the "mySession" session by providing the respective query input comprising the CDOID of the application for which similar/equivalent applications are searched and then he/she obtains back the respective results.

This scenario is actually reflected on the following two Java Code listings which rely on the use of two main client Java classes aforementioned, i.e., the RestClient and the StandaloneClient. All the (client) code and the accompanying (domain) classes (some of which will be analyzed in the sequel) is included in the src sub-directory of the main directory in the respective git repository of the PaaSage project.

```
//Set the marshalling context for those classes whose instances are to be exchanged between the REST server and client
JAXBContext context = null;
try{
	context = JAXBContext.newInstance(eu.paasage.mddb.kb.domain.KBContent.class, eu.paasage.mddb.kb.domain.MySet.class, CDOIDSet.class,eu.paasage.mddb.kb.domain.MyQueryParameter.class, BestApplicationDeployment.class, BestComponentDeployment.class, ApplicationMatch.class, ComponentMatch.class, SuccessfulApplicationDeployment.class,SuccessfulComponentDeployment.class,Wildcard.class);
}
catch(Exception e){
	e.printStackTrace();
}
//Set the name of the KB and StatefulKnowledgeSession to be created
String sessionName = "mySession";
String kbName = "myKB";
//Create the REST KB client
RestClient ec = new RestClient();
//Create a new KB - Basic rules are automatically loaded
ec.createKB(kbName,true);
//Create a new session for this KB
ec.createSession(sessionName,kbName);
//Fire all rules for this session
ec.fireRules(sessionName);
//Add rules to the KB mapping to the particular query to be issued
ec.addRules(kbName,"queries.drl");
```

```
//Get all objects derived from the session
try{
	String s = ec.getObjects(sessionName);
	if (s != null){
		s = s.trim();
		logger.debug("Unmarshalled KBContent: " + s);
		Unmarshaller um = context.createUnmarshaller();
		um.setEventHandler(new
javax.xml.bind.helpers.DefaultValidationEventHandler());
		KBContent mc = (KBContent)um.unmarshal(new InputSource(new
StringReader(s)));
		if (mc == null) logger.debug("No objects were returned");
		else {
			//Do something with the content
			logger.debug("Marshalled KBContent:");
			for (Object o: mc.getElements()){
				Node node = (Node)o;
				Object o2 = um.unmarshal(node);
				if (o2 instanceof BestApplicationDeployment){
					BestApplicationDeployment bad =
(BestApplicationDeployment)o2;
					logger.debug("Got
BestApplicationDeployment with: " + bad.getId() + " " + bad.getApplication()
+ " " + bad.getDeploymentModel() + " " + bad.getMetrics());
				}
				else if (o2 instanceof BestComponentDeployment){
					BestComponentDeployment bad =
(BestComponentDeployment)o2;
					logger.debug("Got BestComponentDeployment
with: " + bad.getId() + " " + bad.getComponent() + " " +
bad.getHostingComponent() + " " + bad.getMetrics());
				}
				else if (o2 instanceof ApplicationMatch){
					ApplicationMatch match =
(ApplicationMatch)o2;
					logger.debug("Got ApplicationMatch with: "
+ match.getFirstApplicationID() + " " + match.getSecondApplicationID() + " "
+ match.getApplicationMatching());
				}
				else if (o2 instanceof ComponentMatch){
					ComponentMatch match = (ComponentMatch)o2;
					logger.debug("Got ComponentMatch with: " +
match.getFirstInternalComponentID() + " " +
match.getSecondInternalComponentID());
				}
				else if (o2 instanceof
SuccessfulApplicationDeployment){

	SuccessfulApplicationDeployment sad =
(SuccessfulApplicationDeployment)o2;
					logger.debug("Got
SuccessfulApplicationDeployment with: " + sad.getApplication() + " " +
sad.getDeploymentModel() + " " + sad.getExecutionContexts() + " " +
sad.getSlos());
				}
				else if (o2 instanceof
SuccessfulComponentDeployment){
```

```
        SuccessfulComponentDeployment scd = (SuccessfulComponentDeployment)o2;
                                    logger.debug("Got
SuccessfulComponentDeployment  with:  "  +  scd.getComponent()  +  "  "  +
scd.getHostingComponent()  +  "  "  +  scd.getExecutionContextID()  +  "  "  +
scd.getSlos());
                                }
                            }
                        }
                }
        }
}
catch(Exception e){
        e.printStackTrace();
}
//Issue a particular query to the session, obtain back the results and print
them
List<MyQueryParameter> content = new ArrayList<MyQueryParameter>();
MyQueryParameter                param1                =                new
MyQueryParameter();param1.setCdoID(CDOUtils.getObject("APP1","Application"))
;
content.add(param1);
MyQueryResults qr = ec.runQuery(sessionName,"applicationMatch",content);
printQueryResults(qr);
//Delete the session
ec.deleteSession(sessionName);
//Delete the KB created
ec.deleteKB(kbName);
```

From the first listing pertaining to the use of the *RestClient*, we will focus mainly on the specifics involved in setting the appropriate marshalling context, in the unmarshalling of returned output from the method calls and the wrapping of parameters to be sent as input to a API method call. First of all, the user has to specify the marshalling context which includes those domain classes whose instances are to be transferred from the REST client to the server and the opposite such that they are marshalled and unmarshalled accordingly. Secondly, in case that the user desires to obtain the output of a method call, such as the *getObjects* one used to obtain the content of a stateful knowledge session in the context of a specific KB, he/she has to first create an Unmarshaller based on the initially specified context, then to unmarshall the output retrieved to obtain the respective object (*KBContent* for the current method) and finally for each part of the object to again unmarshall it to obtain the actual objects contained (this last action is optional depending on the type of output returned). The wrapping of complex input parameters (mapping to domain objects and not objects of basic Java classes) for calling the REST API method is performed via objects of the MyQueryParameter class. In the particular use case and respective listing, the user creates a list of MyQueryParameters and includes in it one such parameter which pertains to the CDOID of the application that needs to be matched. To call the respective *runQuery* method, the user should also specify the name of the query to be called and of course the name of the affected session. The result of the method call is encapsulated in an instance of the MyQueryResults class. (!!!This looks like a point of optimisation for the *getObjects* method!!! - The user does not have to unmarshall the content of the result as this is performed internally by the respective REST library exploited called jersey by just explicating the class of the

object to be retrieved from the method call - see actual code realizing the *runQuery* method)

```
//Set the name of the KB and StatefulKnowledgeSession to be created
String sessionName = "mySession";
String kbName = "myKB";
//Create the KB client
logger.info("Creating StandaloneClient");
StandaloneClient sc = new StandaloneClient();
//Create a new KB
boolean ok = sc.createKB(kbName);
if (!ok) logger.info("Something went wrong with the creation of the KB: " +
kbName);
//Create a new session for this KB
ok = sc.createSession(sessionName,kbName);
if (!ok) logger.info("Something went wrong with the creation of the session:
" + sessionName + " on KB: " + kbName);
//Add rules to the KB mapping to particular queries
ok = sc.addRules(kbName,"input");
if (!ok) logger.info("Something went wrong with the addition of rules in KB:
" + kbName);
//Set globals for the session
ok = sc.setGlobal(sessionName,kbName,"view",sc.view);
if (!ok) logger.info("Something went wrong with the setting of global 'view'
for session: " + sessionName + " in KB: " + kbName);
ok = sc.setGlobal(sessionName,kbName,"threshold",new Double(0.5));
if (!ok) logger.info("Something went wrong with the setting of global
'threshold' for session: " + sessionName + " in KB: " + kbName);
//Fire all rules for this session
ok = sc.fireRules(sessionName, kbName);
if (!ok) logger.info("Something went wrong with the firing of rules in
session: " + sessionName + "of the KB: " + kbName);
//Get objects from session
Object[] objects = sc.getObjects(sessionName,kbName);
if (objects != null){
      for (Object o: objects) logger.info("Got object: " + o.toString());
}
else logger.info("No objects were obtained or something went wrong with
session: " + sessionName + " of KB: " + kbName);
//Issue a particular query to the session, obtain back the results and print
them
objects = new Object[1];
objects[0] = CDOUtils.getObject("APP1","Application");
QueryResults                                qr                                =
sc.runQuery(sessionName,kbName,"applicationMatch",objects);
printQueryResults(qr);
//Delete the session
ok = sc.deleteSession(sessionName,kbName);
if (!ok) logger.info("Something went wrong with the deletion of session: " +
sessionName + "of the KB: " + kbName);
//Delete the KB created
ok = sc.deleteKB(kbName);
if (!ok) logger.info("Something went wrong with the deletion of KB: " +
kbName);
System.exit(1);
```

As it can be seen from the second listing, the execution of the method calls leads to a smaller code size as there is no need to wrap parameters into particular instances of classes in order to transfer them over the Internet. In addition, there is no need to specify any marshalling context. Finally, we should highlight the same set of methods that is used (with different signature) as well as the common way to formulate parameters for queries by neglecting their wrapping. We will come back later on this as the use of the CDO technology has lead to specific restrictions on the information that is being exchanged and the way it is specified.

11.2.5 Domain Models

Any stateful knowledge session of any KB manipulates objects that belong to particular java classes. This creates the necessity to generate a domain model, called domain knowledge model, which will indicate those classes whose instances are to be used for storing facts in the KB as well as for the exchange of input/output between clients and servers. As the drawing of added-value knowledge will rely on the information manipulated and stored by the PaaSage platform in the MDDB, a natural candidate for the population of this domain model or at least for cross-referencing are the classes of the CAMEL DSL[16]. However, this part of the domain model is not enough as it does not capture the added-value knowledge to be derived and stored in the KB. As it can be seen, there is a clear separation between basic information and added-value knowledge drawn from this information. It can be argued that CAMEL could be extended in order to include the modelling of the added-value knowledge. However, this should be avoided for the following two reasons: (a) as the added-value knowledge relies on the basic information stored, any change of the latter information will require the updating of the added-value knowledge - this requires a component which constantly polls the MDDB for such changes and then fires the corresponding rules, thus leading to a substantial overhead which is not actually needed as the added-value knowledge should be retrieved on demand; (b) the decoupling of knowledge from information facilitates users to independently extend the knowledge domain model with their own classes pertaining to the rules that they define thus catering for an individualized and customized exploitation of the KnowledgeManagement API offered.

Based on the above rationale, a particular domain knowledge model has been constructed which is depicted in Figure 40. This knowledge model comprises classes pertaining to the knowledge produced for the basic rules which cover the equivalence/similarity between applications and between components as well as successful and best deployments for applications and their components along with particular helper classes including utility and wrapping ones for marshalling reasons focusing on including as their content results of queries, query parameters, or list of objects derived in the context of a particular stateful knowledge session. This domain model is included in both the client and server code, due to the two way communication performed in the context of calls to the REST API methods between the REST server and client. It must be noted that wrapping classes are not really

[16] We assume that the user is acquainted with the documentation of CAMEL so there is no need to explain the different packages and classes and relationships that are involved.

needed during the exploitation of the StandaloneClient as no network-based communication is involved with any server.

Before analyzing each class of the knowledge model individually, we need to explicate here a very important restriction. As a KB draws information from MDDB, it has to exploit the CDO technology in order to perform the appropriate queries to retrieve it. Fortunately, Drools provides a particular mechanism through which access to a DB can be achieved. This comes with the setting of global external variables that are bound to stateful knowledge sessions. Such variables are considered as static in nature and are evaluated once when a rule is first loaded in a KB. This means that all queries that are to be performed in a context of a rule specification are issued a priori by Drools through exploiting a CDOView configured to be a global variable for a session. Of course, this has an effect on the way rules are to be exploited. In particular, rules should be fired in an on demand basis as they cannot really sense changes in the underlying CDO Repository.

However, the use of CDO technology, apart from restricting the way rules are to be fired, also has an affect on what information is to be stored in a KB in conjunction with the restrictions brought by the REST technology. In particular, the objects of CAMEL classes cannot be immediately used in an individual manner or through their composition into knowledge classes but only a reference to them should be provided. This is due to the fact that there should be some special handling on the way the CAMEL classes are generated from the CAMEL meta-model (to extend the java.io.Serializable interface) as well as an individual handling of classes through JAXB annotations to restrict the way they are marshalled and especially to avoid marshalling cycles due to cross-referencing between CAMEL objects. Added to this, there were two main problems that lead us to follow this restriction: (a) it was observed in the experiments performed in the context of the work in WP4 (which were analyzed in Deliverable D4.1.1) that it is better to rely on lightweight representations of the knowledge as a full and heavy representation imposes a significant overhead on the query time and especially with respect to the case of the REST-based interaction; (b) we wanted to avoid any duplication with respect to what is stored in the KB and what is stored in MDDB especially by considering the fact that objects retrieved in the context of a CDOView cannot be reused after this view has been closed - this of course has the effect that once there are connection problems or the REST service is stopped (but not deleted), the knowledge objects which have full copies of CAMEL objects will be invalid and an exception will be thrown once it is attempted to retrieve them. Based on the above analysis, it was decided that any knowledge object will only reference the *CDOID* of a CAMEL object already stored in the CDO Repository. CDOID is a special class introduced in CDO which is persistent and uniquely identifies an object stored in a CDO Repository. Thus, it constitutes a perfect candidate for object referencing, especially as it is independent of any class-specific identifiers that might have identical values for objects stored in different (CAMEL) models or across different CDO repositories.

Taking into consideration the above analysis and restriction, we now focus on the analysis of the main classes of the knowledge domain model. As it can be seen from Figure 40, there are 24 classes in this model that are now analyzed in more detail. Classes *ComponentMatch*, *ApplicationMatch*, *BestApplicationDeployment*, *SuccessfulApplicationDeployment*, *BestComponentDeployment* and *SuccessfulComponentDeployment* are the main outcome of the basic rules and represent the added-value knowledge produced through the firing of these rules.

ComponentMatch and *ApplicationMatch* represent the matching between two (internal) components and two applications, respectively. The matching information of two applications, apart from referring to the CDOIDs of the applications themselves also contains information about the type of matching (full or partial) as well as the set of CDOIDs of the (common) containing components that have been matched. A partial matching between two applications occurs when a specific percentage below a certain threshold (set as a global variable in the respective session) of components of the first application has been matched with those of the second application. A full matching occurs when all components of the first application have been matched. Please take care of this latter statement as it leads to the induction that the equivalence/full matching between application is not symmetric. The matching of components relies at the moment on their name but could rely on other aspects, if they are specified by the user, such as the information-retrieval-based matching of the component descriptions or the similarity between their code.

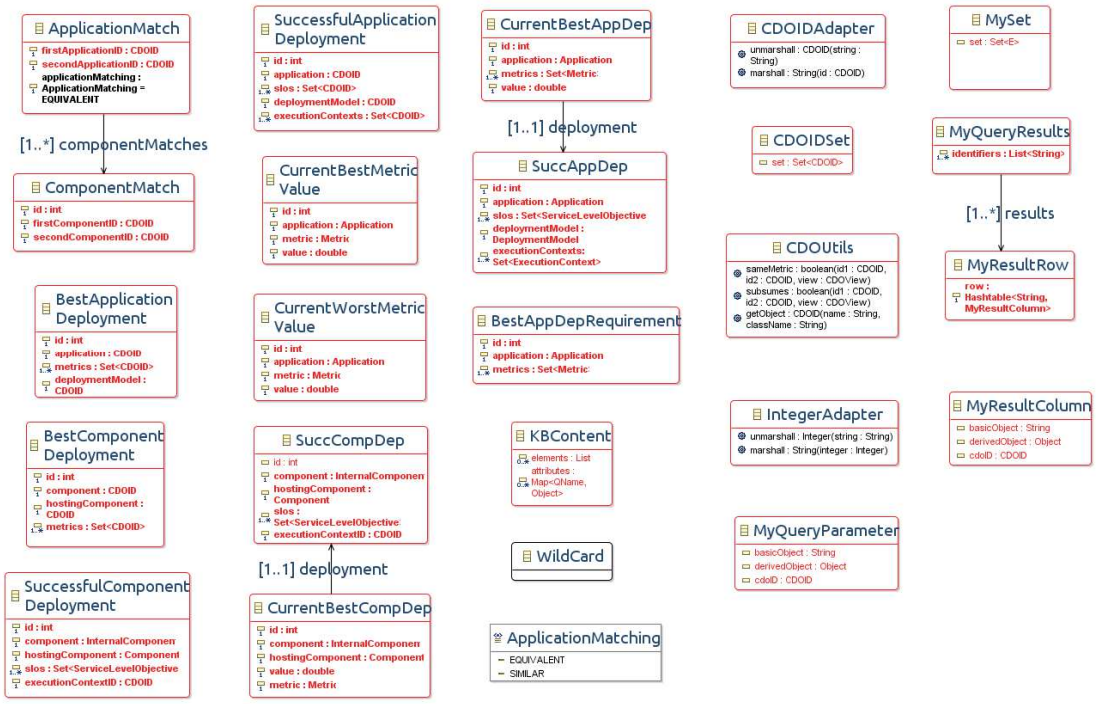

Figure 40 - The class diagram of the domain knowledge model

The remaining (derived knowledge) classes from the aforementioned set refer to best and successful deployments of applications and components. Each class, apart from referring to basic information (CDOIDs of components/applications and respective deployments), is also linked to the some other related CAMEL objects. In particular, successful deployments are associated to the set of CDOIDs of the SLOs that have been respected and of the execution context(s) involved. On the other hand, best deployments, as derived from successful deployments, do not require to reference SLOs but particular metrics involved at the top-level in these SLOs. Another rationale for this relates to the fact that a best deployment should be the most optimal one irrespectively of any SLO imposed on a particular metric. Thus, a deployment is best as it leads to the best value of a metric which of course satisfies even the most demanding SLO posed for this metric in the context of a specific application. It should also be highlighted that best deployments are associated to a set of metrics and not just one metric based on the fact that if a deployment is the best and same for two

or more metrics, then it is better to gather the references to these metrics into a single object rather than having only individual best deployments for each metric that are equivalent. This also facilitates the posing of queries that take into account multiple metrics and desire to discover those deployments that are the most optimal ones according to the conjunction of these metrics.

The *CurrentBestAppDep*, *CurrentBestCompDep*, *CurrentBestMetricValue*, *CurrentWorstMetricValue* classes are used for the representation of intermediate facts that lead to the final generation of the added-value knowledge. The first two represent the current best deployment that has been identified so far during the progress in the firing of the rules for applications and components, respectively. The latter two represent the best and worst metric value that has been so far seen for a particular application and metric such that these values can then be compared with the remaining successful application deployments in order to finally derive the best one.

The SuccAppDep and SuccCompDep are convenience classes which are equivalent to those of the SuccessfulApplicationDeployment and SuccessfulComponentDeployment classes, respectively. The main rationale for their generation is that they lead to less requirements to the posing of queries to the CDO Repository as they map to full copies of the underlying CAMEL objects and not just CDOID references. This is an essential move as we do not desire to have rules whose firing delays a lot and leads to substantial load in order to be able to handle CDO Repositories of even greater size.

The classes related to the REST-based I/O comprise the following:

- *KBContent*: represents the information derived/stored in a particular session of a specific KB (which can be obtained by calling the *getObjects* API method). Objects of this class can also be regarded as potential KB content to be stored so they can be used as input for the respective API method (*addObjects*).

- *MyQueryParameter*: represents query parameter information that can be passed for running a particular query to a stateful knowledge session. Special care must be given to the way the information in this class is specified as there are three types of objects that can be used: (a) basic Java objects which should be mapped to a String (see basicObject) mapping to the representation of e.g. names or any other kind of attribute pertaining to a certain class that is expected to be used as input parameter to a query; (b) CDOIDs to indicate a required cross-reference to a particular CAMEL object; (c) complex Java objects which could be related to particular knowledge domain classes that have to be passed as input parameters.

- *MySet*: used to represent a set of any domain object that can be passed as input or output to the API methods. For instance, it can be used to represent the content of a query parameter (*MyQueryParameter*) in the third object type case as explained previously.

- *MyQueryResults*: represents the resulting information from a particular query to a stateful knowledge session. It comprises row-based information that is represented by another class called *MyResultRow* along with a

particular set of column identifiers that can be used to fetch the respective column value of a row-based result. The columns of a result row are represented by the *MyResultColumn* class which has similar content with respect to a query parameter.

- *CDOIDAdapter*: As a CDOID is just an interface, we need to define an extension to XMLAdapter in order to be able to map the content of a CDOID into a String during marshalling and back from String to CDOID during unmarshalling.

- *Wildcard*: It is used to represent a parameter with any possible value to cater for the posing of generic queries which should give back as a result all objects that match the query focus (e.g., the best deployment for all applications if no concrete application or metric is provided as input to the query). It should be noted that any expected input parameter to a query can be represented in this way catering for fully generic (all parameters are generic) or partially generic queries (some parameters are generic).

- *CDOIDSet*: This is a special class used to represent a set of CDOIDs. This is required in cases that a knowledge object refers to a set of CDOIDs and not just to a single CDOID, as it is the case with respect to the *BestApplicationDeployment* class which refers to a set of CDOIDs of the respective metrics for which this application deployment is the best.

Finally, the *CDOUtils* is a utility class which includes the specification of static functions which can be used in the context of rules specifications in order to facilitate the compact representation of the rules by encapsulating functionality that is repeated across rules in the form of a method call. For instance, whenever a rule desires to inspect in the antecedent part whether two SLOs map to the same metric, it can call the *sameMetric* method of the CDOUtils class in order to obtain the respective boolean evaluation result. The same functions or even additional ones (as it is the case for *getObject*) can be used also in the context of the main method of the clients or any other main method which involves the usage of any of the two clients offered in order to facilitate the interaction with CDO in order to fulfil a certain functionality (such as the retrieval of a CDOID for an CAMEL object which is identified by a particular value with respect to its name as it is the case of the functionality exposed by the *getObject* method).

11.2.6 Implementation / Technical Details

11.2.6.1 Implementation Details

The API code has been implemented in Java and has exploited the Jersey java library (version 1.14). The enunciate library (version 1.27) was also exploited in order to produce on-line documentation that is provided by the same end-point from which the REST API methods can be called. The Drools Expert module (6.2.0.Final) was utilized in order to be able to manage the KBs and the sessions created out of them. As already indicated both the CDOClient (to open CDOSessions and create the

respective CDOViews required to pose queries on the CDO Repository) and the CAMEL domain model are those libraries that are essential for the proper functioning of both types of clients exposed (and of the REST service of course).

Spring security has been used in order to secure the access to the API methods. Thus, in order for a user to run any API method, basic authentication information needs to be passed as part of any call. A simple user is for the moment expected and needs to be created in the context of Tomcat (or any other container) with the "paasage":"paasage" login and password credentials. Thus, this basic authentication information will need to be passed in the REST API method calls (see also configuration information indicated later on). In the next version of the REST API, the single-site security solution realized in the context of WP4 will be exploited in order to have a correspondence between the actual users and their respective information stored in the CDO Repository such that these users can then be authenticated according to their credentials. Concerning authorisation, we expect that each user will be able to access only the information pertaining to the roles assigned to it. It will be explicated how this will be performed in the context of the single-site security solution once it is finally realized.

11.2.6.2 Technical Usage Details

The client code has been realized as a maven project which can be easily imported/loaded into any java development environment (e.g., SpringSource) in order to be compiled and executed. The code can be compiled by just executing "mvn clean install" either in a terminal/execution command or through the java development environment of your choice. It can be run by executing the following command: "mvn exec:java (-Deu.paasage.configdir=<path_to_properties_dir>)?". As it can be seen, the code is either executed without any JAVA VM directive or with a specific one. In the first case, it is expected that an environment variable called PAASAGE_CONFIG_DIR has been set from which the path to the properties directory can be obtained while in the second case, this path is provided through the VM directive which sets a particular system property called "eu.paasage.configdir" which points to this path.

Two configuration files are needed: (a) eu.paasage.mddb.cdo.client.properties mapping to the properties file expected to be read by the CDOClient exploited by the two clients offered and (b) eu.paasage.mddb.kb.client.properties mapping to the properties file expected to be read by the RestClient in order to properly connect to the REST server. As the content of the first file is covered in the documentation of the CDOClient, we only analyze the content of the second configuration file.

In the eu.paasage.mddb.kb.client.properties configuration file four properties can be mainly specified:

- *host*: The IP of the host on which the REST service resides
- *port*: The port on which the REST service listens (actually the container hosting this service). Default value is "8080".
- username: The username of the user given for authentication purposes.
- password: The password of the user given for authentication purposes.

It must be noted that the username and password values are fixed for the moment, as indicated previously.

Contact Information: For any inquiries concerning the REST service or the two types of clients offered, please send an email to: kritikos@ics.forth.gr

11.2.7 API Methods Signature Analysis

We now provide a signature analysis for the methods of the REST API as this is more complicated due to the remote technology used. We do not cover the signature analysis for the methods of the StandaloneClient as these method signatures are quite similar to the ones analyzed and quite straightforward (the interested reader can of course resort to the code listing as the main code of this client which includes all possible interactions with a knowledgebase and its respective stateful knowledge sessions). In the sequel, we consider that the IP of the host on which the REST service resides is "localhost", i.e., the service runs locally. In addition, the signature analysis is split between the two main lifecycles and relies on a particular structural declarative approach dictating the method type, description, I/O parameters, exceptions raised and short usage example in the form of a curl-based call.

For KnowledgeBase management

- *Name*: **createKB**, *URI*: http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/createKB .
    - *Method Type: POST*
    - *Description*: This method is used for creating KBs by specifying their name as well as indicating whether the basic rules will be loaded on them. Please note that KBs are manipulated by their names, so they have to be unique for each user. In other words, one user cannot have two KBs with the same name but two different users can create KBs with the same name.
    - *Input Parameters*:
        - **kbName** : The name of the KB to create as a String. Type: *QueryParam*
        - **basic** : Indicates whether the basic rules (for application/artifact matching and successful/best application/artifact deployment) should be loaded on the KB to be constructed. Default value is true. Type: *QueryParam*
    - *Output*: A successful message indicating that the KB has been created
    - *Exceptions*:
        - **BadRequestException**: User has either provided a name which is already used by another, previously created KB or no KB name has been actually provided.

        - **InternalServerErrorException**: An internal error has been produced which prevented the service from creating the KB requested by the user.

    - *Usage*: Suppose that a KB with name "test" must be created on which the basic rules will be loaded. Then, the curl command to execute the method will be:

"curl –i -X POST –u paasage:paasage http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/createKB?kbName=test"

- *Name*: **addRules**, *URI*: http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/addRules .

    - *Method Type: POST*

    - *Description*: This method is used for loading particular rules in a specific KB that is owned by the requesting user. The rules are specified according to the textual rule description format of Drools in a drl file and provided as a *FormDataParam* to the user request. Please note that the user should avoid providing different definitions of the same rule as this may lead to undesirable results.

    - *Input Parameters*:

        - **kbName**: The name of the KB, on which the user rules will be loaded, as a String. Type: *FormDataParam*

        - **drlFile**: The input drl file containing the Drools rules. Type: *FormDataParam*

    - *Output*: A successful message indicating that the KB has been updated with the new rules

    - *Exceptions*:

        - **NotFoundException**: The user did not provide a valid KB name

        - **BadRequestException**: The user did not provide a KB name or any drl file.

        - **UnsupportedSyntaxException**: The drl file provided has a wrong syntax.

        - **InternalServerErrorException**: An internal error has created that has prevented the service from fulfilling this request.

    - *Usage*: Suppose that the user desires to update his/her KB with name "test" with a drlFile called "". Then, the following curl command to execute the method could be issued:

"curl –i -X POST -F kbName=test -F drlFile=@<path to drl file> –u paasage:paasage http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/addRules?kbName=test"

- *Name*: **deleteKB**, *URI*: http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/deleteKB .
    - *Method Type: POST*
    - *Description*: This method can be used for deleting a KB with a particular name. Please note that by deleting a KB, the respective rules loaded on it as well as the sessions created out of it with their respective knowledge are also deleted.
    - *Input Parameters*:
        - **kbName** : The name of the KB, to be deleted, as a String. Type: *QueryParam*
    - *Output*: A successful message indicating that the KB has been deleted.
    - *Exceptions*:
        - **BadRequestException**: The user did not provide any KB name.
        - **NotFoundException**: The user did not provide any valid KB name.
        - **InternalServerErrorException**: An internal error has occurred that prevented the service from fulfilling the user request.
    - *Usage*: Suppose that the user desires to delete his/her KB with name "test". Then, the following curl command to execute the method could be issued:

"curl –i -X POST –u paasage:paasage http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/deleteKB?kbName=test

For StatefulKnowledgeSession management

- *Name*: **createSession**, *URI*: http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/createSession .
    - *Method Type: POST*
    - *Description*: This method is used for creating a session with a particular name out of an existing KB owned by the user.
    - *Input Parameters*:
        - **sessionName**: The name of the session to be created as a String. Type: *QueryParam*
        - **kbName** : The name of the KB to be used for the creation of the session. Type: *QueryParam*

- *Output*: A successful message indicating that the session has been created
- *Exceptions*:
  - **BadRequestException**: The user did not provide any name for the session or the KB or a session with the same name already exists.
  - **NotFoundException**: The user did not provide a valid KB/session name.
  - **InternalServerErrorException**: An internal error has occurred that prevented the service from fulfilling the user request.
- *Usage*: Suppose that a KB with name "test" must be used for creating a session named as "testSession". Then, the curl command to execute the method will be:

"curl –i -X POST –u paasage:paasage http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/createSession?sessionName=testSession&kbName=test"

- *Name*: **fireRules**, URI: http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/fireRules .
  - *Method Type: POST*
  - *Description*: The method is used for firing all the rules of a particular session.
  - *Input Parameters*:
    - **sessionName**: The name of the session on which the rules will be fired, as a String. Type: FormData*Param*
  - *Output*: A successful message indicating that the rules have been fired
  - *Exceptions*:
    - **BadRequestException**: The user did not provide any name for the session or a session with the same name already exists.
    - **NotFoundException**: The user did not provide a valid session name.
    - **InternalServerErrorException**: An internal error has occurred that prevented the service from fulfilling the user request.
  - *Usage*: Suppose that all rules for session named "testSession" must be fired. Then, the curl command to execute the method will be:

"curl –i -X POST –u paasage:paasage http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/fireRules?sessionName=testSession"

- *Name*: **addObjects**, URI: http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/addObjects
    - *Method Type: POST*
    - *Description*: Through this method the user can add objects to a particular session which can be of course exploited in the subsequent firing of the corresponding rules.
    - *Input Parameters*:
        - **sessionName**: The name of the session on which the objects will be added, as a String. Type: *FormDataParam*
        - **list**: An object of type KBContent must be passed for this parameter containing the objects to be added. Type: *FormDataParam*
    - *Output*: A successful message indicating that the objects have been added
    - *Exceptions*:
        - **BadRequestException**: The user did not provide any name for the session or a session with the same name already exists. Thrown also when no object is given.
        - **NotFoundException**: The user did not provide a valid session name.
        - **InternalServerErrorException**: An internal error has occurred that prevented the service from fulfilling the user request.
    - *Usage*: Suppose that the user packs the desired objects into a KBContent object "obj1" and desires to add them to session named "testSession". Then, the curl command to execute the method will be:

"curl –i -X POST –u paasage:paasage -F sessionName=testSession -F list=<json:serialization of list object> http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/addObjects"

or

"curl –i -X POST –u paasage:paasage -F sessionName=testSession -F list=<path to file containing the xml serialization of list object> http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/addObjects"

- *Name*: **getObjects**, URI: http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/getObjects .
    - *Method Type: POST*
    - *Description*: Through this method, the user can obtain all the objects that have been derived/stored in a particular session.

- *Input Parameters*:
  - **sessionName**: The name of the session where the objects have been created/derived and need to be retrieved, as a String. Type: *QueryParam*
- *Output*: A KBContent object containing the objects that have been derived from the user session
- *Exceptions*:
  - **BadRequestException**: The user did not provide any name for the session or a session with the same name already exists.
  - **NotFoundException**: The user did not provide a valid session name.
  - **InternalServerErrorException**: An internal error has occurred that prevented the service from fulfilling the user request
- *Usage*: Suppose that the user wants to obtain all derived objects from a session named "testSession". Then, the curl command to execute the method will be:

"curl –i -X POST –u paasage:paasage http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/getObjects?sessionName=testSession"

Please note here that the result will be serialized in XML or JSON, so you need to deserialize it into a Java Object in order to exploit it (see client code).

- *Name*: **runQuery**, URI: http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/runQuery .
  - *Method Type: POST*
  - *Description*: This method is used in order to run a query over a particular session. If input query parameters are given by the user, then the results obtained will be restricted according to this input. Otherwise, all possible results will be returned.
  - *Input Parameters*:
    - **sessionName**: The name of the session, on which the query will be run, as a String. Type: *FormDataParam*
    - **queryName**: The name of the user query as a String. Type: *FormDataParam*
    - **parameters**: A list of *FormDataParam* objects that are going to be passed as input parameters to the query
  - *Output*: A MyQueryResults object containing the results of the user query.

- *Exceptions*:
  - **BadRequestException**: The user did not provide any name for the session or a session with the same name already exists or wrong parameters were given.
  - **NotFoundException**: The user did not provide a valid session name.
  - **InternalServerErrorException**: An internal error has occurred that prevented the service from fulfilling the user request
- *Usage*: Suppose that the user wants to run the query "testQuery" on a session named "testSession" with no input parameters . Then, the curl command to execute the method will be:

"curl –i -X POST –u paasage:paasage -F sessionName=testSession -F queryName=testQuery http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/runQuery"

- *Name*: **deleteSession**, URI: http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/deleteSession .
  - *Method Type: POST*
  - *Description*: This method is used for deleting a particular session. This means that all objects derived from this session will also be removed.
  - *Input Parameters*:
    - **sessionName** : The name of the session, to be deleted, as a String. Type: *QueryParam*
  - *Output*: A successful message indicating that the session has been deleted.
  - *Exceptions*:
    - **BadRequestException**: The user did not provide any name for the session or a session with the same name already exists.
    - **NotFoundException**: The user did not provide a valid session name.
    - **InternalServerErrorException**: An internal error has occurred that prevented the service from fulfilling the user request
  - *Usage*: Suppose that the user desires to delete his/her session with name "testSession". Then, the following curl command to execute the method could be issued:

"curl –i -X POST –u paasage:paasage http://localhost:8080/ksession-manager-1.0-SNAPSHOT/rest/knowledge/deleteSession?sessionName=testSession"