PaaSage

Model Based Cloud Platform Upperware

Deliverable D4.1.1 / D4.3.1

Prototype Metadata Database and Social Network / Prototype of Metadata Integration Extension

Version: 1

D4.1.1 / D4.3.1

Name, title and organisation of the scientific representative of the project's coordinator[1]:
Mr Tom Williamson Tel: +33 49238 5072 Fax: +33 4 92385011
E-mail: tom.williamson@ercim.eu

Project website[2] 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:	31/10/13
Document	
Period covered:	
Deliverable number:	D4.1.1 / D4.3.1
Deliverable title	Prototype Metadata Database and Social Network / Prototype of Metadata Integration Extension
Contractual Date of Delivery:	31/03/2014 (M18)
Actual Date of Delivery:	
Editor (s):	Kyriakos Kritikos
Author (s):	Kyriakos Kritikos, Maria Korozi, Bartosz Kryza, Tom Kirkham, Asterios Leonidis, Kostas Magoutis, Philippe Massonet, Stavroula Ntoa, Antonis Papaioannou, Christos Papoulas, Craig Sheridan, Chrysostomos Zeginis
Reviewer (s):	Arnor Solberg, Jörg Domaschka
Participant(s):	Alessandro Rossini, Daniel Romero
Work package no.:	4
Work package title:	Communications Hub
Work package leader:	Kostas Magoutis
Distribution:	
Version/Revision:	1.0
Draft/Final:	
Total number of pages (including cover):	

[1] Usually the contact person of the coordinator as specified in Art. 8.1. of the grant agreement

[2] The home page of the website should contain the generic European flag and the FP7 logo which are available in electronic format at the Europa website (logo of the European flag: http://europa.eu/abc/symbols/emblem/index_en.htm ; logo of the 7th FP: http://ec.europa.eu/research/fp7/index_en.cfm?pg=logos). The area of activity of the project should also be mentioned.

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

The Metadata Database (MDDB) is an important component of the overall PaaSage platform architecture, fulfilling mainly two roles: (a) storing the information required for the correct functioning of the other PaaSage components and (b) is leveraged as an indirect way of communication between two PaaSage components. The MDDB stores all information modeled by the Domain-Specific Languages (DSLs) exploited in the context of PaaSage project, and includes additional aspects such as execution histories of applications, descriptions of organizations, cloud providers and roles, and provisioning of triggering rules, among others. A key objective for the MDDB is to ensure the linkage of these additional aspects to the concepts in the respective DSLs.

The social network infrastructure described in this deliverable is another important component of the overall PaaSage platform architecture, forming an interface between the PaaSage system and its users, and fostering a community of stakeholders that interact with PaaSage by providing and retrieving knowledge in the area of cloud software engineering.

This deliverable describes the first MDDB prototype architecture and implementation of the respective components, including the social network infrastructure. It describes the mapping between the information modeled in the DSLs and the MDDB schema and the corresponding tables and columns by providing particular visualizations of the mappings. It also provides examples of how particular models conforming to these DSLs are mapped to MDDB content. Finally, it presents results from the evaluation of specific components to highlight different aspects of their performance.

This deliverable describes in detail the current development status of the MDDB prototype and the components that comprise it, the status of the internal integration effort, and the remaining work to be performed through Month 36 of the project. The MDDB prototype offers the overall functionality planned to date, comprehensively covering the information needed to enable the project use-cases. The path towards integration between the various components of the MDDB relies on the principle of clear separation of responsibilities between components.

Overall, the consortium has put significant effort in realizing the MDDB prototype with satisfactory results. While more work lies ahead, the current status forms a solid basis for the project. The implementation plan offers a clear path to further enhancing and finalizing the development of the MDDB and social network prototype.

Intended Audience

This is a public document intended for readers with expertise in cloud computing and relational database modelling. The reader is also referred to the description of the overall PaaSage architecture presented in Deliverable D1.6.1 [D1.6.1]. 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 D4.1.1, the reader is introduced to the functionality implemented in the M18 prototype of the MDDB and the social network infrastructure and the next steps in completing the features and functionality planned for M36.

Contents

1 Introduction

The Metadata Database (MDDB) is an important component of the overall PaaSage platform architecture designed to store the information required for the correct functioning of the other PaaSage components and acting as an indirect way for PaaSage component communication. The MDDB integrates a number of Domain-Specific Languages (DSLs) exploited in the context of PaaSage. It further integrates the DSLs with other information such as execution histories of applications, descriptions of organizations, cloud providers and roles, and provisioning of triggering rules. A key concern for the MDDB is to ensure that all information is appropriately inter-related and cross-linked in a consistent manner.

The intent of this deliverable is to provide a comprehensive and detailed exposition of the development status of the MDDB prototype, including design principles, its architecture, an analysis of the functionality of the respective components, and an initial evaluation. This deliverable also provides a roadmap based on which features and functionality planned for the project period up to M36 will be realized.

In Section 2 the main architecture of the MDDB prototype is presented along with a high level description of the functionality provided by the respective MDDB components. Sections 3-5 are dedicated to a detailed exposition and analysis of the design rationale and functionality of the main components of the MDDB prototype, which are the relational database, knowledge base, and social network infrastructure.

Section 3 discusses the principles and methodology used to design the MDDB schema. It describes the main information aspects covered by this schema and how different parts of it map to corresponding DSL languages. Examples are provided focusing on a scenario involving the SensApp application and describing how the information from models complying to the DSLs is mapped to the respective MDDB content. Next, Section 3 discusses issues concerning the enforcement of a security layer over the MDDB and analyzes in detail the respective security components and their implementation. Moreover, there is a discussion on how the DSL-to-MDDB mappings can be realized according to the type of mappings induced and the current technology available. Finally, Section 3 discusses methods to achieve scalability and high availability in the MDDB.

Section 4 presents the Knowledge Base, a component that enables various PaaSage components and modules to derive additional, added-value facts from the MDDB. The analysis concentrates on the design rationale, the criteria for selecting particular implementation technologies, and the design and realization of a particular API through which the PaaSage components can interact and obtain the derived facts.

Section 5 focuses on the Social Network (SN), one of the most important components of the MDDB, constituting one of the main information entry points for the users as well as the place for knowledge-sharing and exploitation of the platform's main functionality. Section 5 describes the details of the SN's back-end design and implementation. Next, the design of the SN's UI is analyzed by describing the design rationale and the main UI mockups. Finally, the section focuses on detailing the main mechanisms for enforcing authentication as well as a secured access to the functionality exhibited by the SN.

Section 6 presents a preliminary experimental evaluation of different aspects of the performance of the MDDB components in different scenarios and discusses the results produced. Some of these results provide valuable insight into the need for slight design adjustments of some components so as to optimize them. Finally, Section 7 supplies a roadmap for organizing and realizing the remaining work towards the final MDDB prototype, while Section 8 concludes this deliverable.

2 Architecture

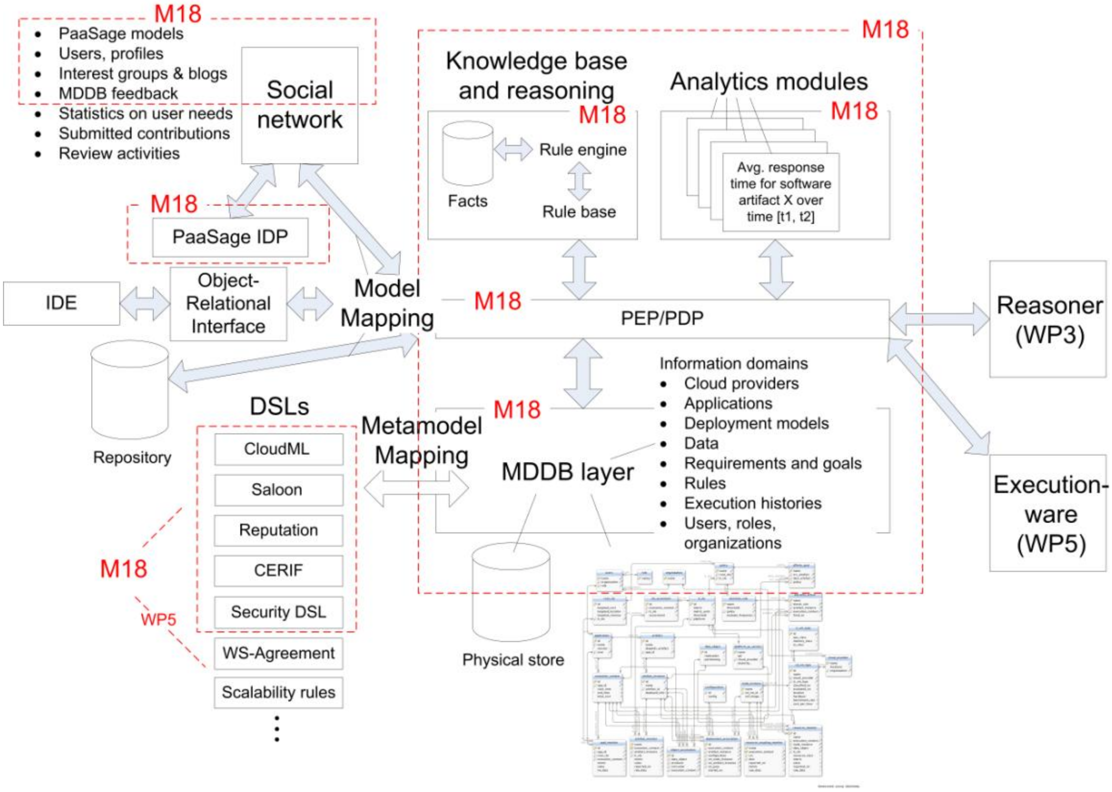


Figure 1 - The architecture of the MDDB System

The architecture of the MDDB can be seen in Figure 1. It comprises various components which exhibit particular functionality and which cooperate with each other in order to achieve common goals. The MDDB components are the following:

- The social network (SN) which is responsible for the sharing of knowledge between PaaSage users and the PaaSage platform. The SN is actually the main point of interaction between users and the PaaSage platform where users can create application models, issue queries as well as populate the MDDB, deploy applications in the Cloud, add rules to the Knowledge Base (KB) and issue queries for retrieving high-level knowledge. Apart from this, users can perform usual tasks in SNs such as join groups, participate in discussions in blogs, send messages to each other as well as manage their profiles.
- Attached to the SN is the Identity Provider (IDP) which is a module responsible for the management and identification of users and their information. IDP is a realization of Single-Sign On (SSO) functionality which enables users to authenticate themselves and obtain authentication tokens which can then be passed to another security layer (PEP/PDP) which is

responsible for the authorization of users based on their requests and the policies imposed on the required resources.

- The Policy Enforcement and Decision Points (PEP/PDP), as already indicated, are responsible for the authorization of user requests. PEP will be responsible for obtaining the user request and transforming it into the appropriate format that can be properly processed by the PDP as well as enriching it through obtaining additional user information from the MDDB. On the other hand, PDP will be responsible for identifying the appropriate resource policies based on the user requests and evaluating them in order to assess whether access to the resources will be granted. PEP/PDP is the main security layer through which any component (internal to MDDB or a PaaSage component) can interact with the MDDB components in order to obtain access to the appropriate functionality. For instance, as it can be seen from Figure 1, three main PaaSage components (Reasoner, ExecutionWare and Profiler) interact with the MDDB components (namely KB, Analytics Module and MDDB layer) through PEP/PDP.
- The Knowledge Base (KB) is the model responsible for deriving new, high-level knowledge from the information stored in the MDDB layer which can assist in performing particular tasks, such as application matching that can be exploited for instance in order to obtain information for new applications from the information already stored for old applications that match them. Such information is exploited both by internal MDDB components, such as the SN (for e.g., assisting in the design of CloudML models), as well as external PaaSage components/modules, such as the Profiler (e.g., for obtaining useful profile information for new applications from old ones).
- The Analytics Module (AM) is responsible for performing analytics over performance data stored in the MDDB layer concerning the execution history of cloud-based applications deployed through PaaSage. Such analytics can involve the statistical processing of huge amount of information in order to provide insights about the performance of an application in terms of high-level metrics. This information can be of great value to many components and modules of the PaaSage platform, such as the Profiler (e.g., enrich the profile of an application) or the Reasoner (e.g., exploit this information for better matching performance requirements to capabilities).
- The MDDB layer is the place where all of the cloud-based application management information is stored and can be accessed through standard SQL queries or the object-relational interface. Such information can, then, be exploited in order to perform various application management tasks, such as the deployment and adaptation of cloud-based application. Moreover, it can be used for the (indirect) communication between PaaSage components/modules where one component can store information which the other component can exploit in order to perform its required functionality/tasks. At the conceptual level, MDDB exploits a particular database schema which is responsible for integrating as well as mapping the various DSLs and the main CAMEL language used in the project. At the physical layer, the information stored based on this schema is physically distributed into various database stores which are integrated and synchronized with each other.

One of the main benefits of the MDDB system is that it offers four different ways through which different types of PaaSage users can interact with it:

1. Through UI interfaces of the Social Network. This way is more appropriate for application developers/modellers who would like to model their application, have access to the PaaSage's main functionality, input content to the system (e.g., organization description and policies) and share knowledge.
2. Via plain SQL queries. This way is appropriate both for application developers/modellers who would like to examine in more detail the content of the MDDB and for PaaSage developers in the realization of their components by posing particular SQL queries which would be used as valuable input for the proper support of a component's particular functionality.
3. Through the KnowledgeBase which builds upon the content of the MDDB to derive added-value facts. This way is appropriate for both application developers/modellers, as they obtain through the Social Network important support for their modelling and deployment tasks as well as for the PaaSage developers who obtain support for realizing the functionality of particular components, such as the Reasoner through obtaining the knowledge of best deployments for particular applications.
4. Via an object interface through which PaaSage users can perform different types of queries (HQL or SQL), obtain the respective information in terms of domain-specific objects, manipulate and update this information and store it back to the MDDB. Such an interface is beneficial for PaaSage users and developers which tend to exploit Ecore-based DSLs, such as CloudML or Saloon. Currently, this interface relies on the Hibernate technology which enables a one-to-one mapping between the MDDB and the respective domain objects. This will be modified through exploiting hibernate or JPA annotations in order to enable a different mapping that is required between the DSLs and the MDDB. An analysis of technologies which could be used to realize the required DSL-to-MDDB mappings is provided in Section 3.1.8, while an evaluation of the query performance of the 1-1 hibernate-based mapping is supplied in Section 5.2. Section 7 as well as Deliverable D2.1.2 [D2.1.2] explain what decisions have been made for selecting the particular mapping technology that will be used in the course of this project.

Based on the above analysis, it is apparent that each MDDB component has a specific role and functionality to offer to the MDDB module/prototype and that it securely cooperates or draws content from other MDDB components via the exploitation of a sophisticated security layer. The latter layer is also responsible for ensuring that appropriate authentication and authorization activities are pursued to secure the access of MDDB components for other components/modules requests. The distribution at the physical layer of MDDB also ensures that the vast MDDB content is timely accessed by any component of the PaaSage platform and that it is distributed in such a way that no physical MDDB store/DB becomes a bottleneck.

3 Metadata Database Design & Implementation

3.1 Relational Database – Core Design

The MDDB schema is regarded as the place where all PaaSage incoming or produced information is stored and the medium through which PaaSage components and modules can indirectly communicate. In addition, this schema is considered to be able to capture all the information from the main DSLs used in the project, thus actually being a superset of all these DSLs.

The decision of choosing a relational database relied on the fact that relational database management systems have been used across many decades and constitute quite mature technology with quite satisfactory and scalable performance. Moreover, by exploiting object-to-relational solutions, we will be able to easily realize the required mapping between the DSLs used in PaaSage and the MDDB.

The design of the MDDB schema was performed in such a way that not only all the necessary information is captured but also there is no redundancy and that any type of information from any DSL is captured by it. The relationships between cloud entities were also reviewed so as to create the respective association tables and avoid any wrong modelling of information as well as data redundancy and duplication. An iterative model for producing the schema was followed which relied on the fact that each time a different information aspect was on focus but also on being able to handle any changes that had occurred for particular, evolving DSLs, such as CloudML. Each time a particular information aspect was covered in the schema, a cooperative action with respective WP2 partner organizations (responsible for the corresponding DSL) took place in order to validate it and ensure that all required information was covered, especially in terms of the various use-cases that will be supported in the project.

When a particular information aspect was more or less finalized, then the mapping from/to the respective DSL was also designed in a visual way as another validation step to ensure that not only all DSL information is covered but that it can also be easily mapped. The realization of these mappings is on-going but it will be based on the decisions made and the subsequent selection of the appropriate tools which will ensure that they are performed in a proper and correct way.

Based on the above methodology, a set of MDDB incomplete schema versions were produced and circulated to all PaaSage partners and a first, stable and complete MDDB schema release was finally developed before M18 which can now be tested as well as used for the storage of all related PaaSage information and the indirect communication between PaaSage components. As already indicated previously, some of the DSLs still evolve, so this means that the first stable MDDB schema version will be modified in the near future when new versions of the DSLs will be produced. As we expect that no serious modifications will take place (just small modifications or extensions) for these DSLs, then it will be a quite easy task to update the respective MDDB schema part. The corresponding mapping specifications will also be updated, if needed. Based on our previous expectation, this will be again an easy task as only the parts of the mapping that correspond to the modified information at the DSL level will need to be adjusted.

Information Aspect	DSL
Users, roles and organizations	CERIF[3] [CERIF]
Applications specification & deployment	CloudML [CloudML]
Cloud providers & their offerings	Saloon [Saloon]
User requirements and execution/monitoring/assessment information	WS-Agreement [WS-Agreement]
Security requirements, capabilities and policies	(under-design) Security DSL, XACML
Scalability rules	Scalability & event DSL

Table 1 - The information aspects covered by MDDB schema and the respective DSLs for each aspect

The main aspects of information that are captured by MDDB and particular DSLs (covered in detail in the remaining sub-sections), are depicted in Table 1 and are shortly analyzed as follows:

- *users, roles and organizations*: Here CERIF plays the major role as a DSL capable of capturing the required information. Through CERIF, not only information about the main entities (i.e., users, roles and organizations) is covered but also cloud-related information, such as which data centre belongs to a cloud provider, as well as information about the roles played by users or user groups and the related domain-specific policies in terms of resource usage.
- *applications specification & deployment*: CloudML is the main DSL to cover this information through its cloud-provided independent specifications which are able to model applications and their constituting software components as well as the dependencies between these application components. It is also able to map application components to configuration information as well as to the cloud provider services (VMs, PaaS) on which they will be deployed. The MDDB schema models this information in such a way that it can be re-used in the specification of other applications as well as to correlate user requirements with the deployments that they have driven.
- *cloud providers and their offered services*: Saloon is the DSL which is able to describe rich and quite expressive feature models that are ideal for expressing the variability offered in terms of services for cloud providers. The MDDB schema, although not being able to employ the same structure for describing this information, it is able to cover all the necessary information as well as include additional, added-value one mainly concerning a fair and uniform rating of the cloud services offered according to one or more aspects, such as CPU, memory, IO and network.
- *user requirements and monitoring/assessment information*: User requirements are mainly expressed through IT SLOs that are usually part of SLAs. Here WS-Agreement is envisioned as the DSL which is able to cover all the

[3] www.eurocris.org/cerif

related information, such as the parties involved, the agreed service levels in terms of SLOs and the penalties incurred when these SLOs are violated. As no monitoring/assessment information is actually covered by WS-Agreement, the MDDB schema was designed to also cover the missing information in terms of the metrics that are involved in the SLOs as well as the monitoring information produced at run-time during application execution and respective assessment information in terms of evaluating the agreed SLOs between the application user and cloud provider.

- *security requirements, capabilities and policies*: As security requirements and capabilities are not actually covered by any DSL, MDDB schema was designed such that the respective information was covered in terms of which high-level security controls are required and provided by cloud users and providers, respectively. A new security DSL is also designed according to this purpose to also cover this information but its design will be finalized after M18. While CERIF is the main point where policies can be extracted from organizations, there is a need for a DSL that is actually used for the enforcement of these policies. To this end, XACML is exploited and the MDDB schema again has guaranteed that all respective policy information is covered in a flexible way such that one policy can be related to more than one actions that can be performed by a particular role on more than one resources.
- *scalability rules*: As generic rule languages can be exploited to express scalability rules, the PaaSage project can select one of them for this purpose with the main exception that a rich event meta-model/DSL is required so that complex events can be really expressed as well as all the required event details. To this end, a concatenation of a rule and event DSL is being proposed in Deliverable D2.1.2 [D2.1.2] and MDDB schema again has been realized to cover all the respective information as well as information concerning the triggering of rules by concrete events and the actual actions performed.

In the following, Section 3.1.1 is dedicated to analyzing the CAMEL language under design which is expected to be a superset of all the DSLs exploited. Next, Sections 3.1.2 to 3.1.7 focus on shortly analyzing all the DSLs, with greater detail attributed mainly to the ones with which integration with MDDB is expected to have matured more by M18. Each sub-section also analyzes the way the MDDB schema covers the required information, even the one missing from the respective DSL, and finally presents a visual proposition for mapping at the schema level the MDDB with this DSL. Finally, the last sub-section (Section 3.1.8) is dedicated to reviewing mapping techniques that could be used for realizing the proposed mappings between the DSLs and the MDDB.

3.1.1 CAMEL

CAMEL (Cloud Application Modelling Language) is an ontology-based schema that is designed to be a superset of the DSLs used in PaaSage. CAMEL is formed at run time and spans the main Cloud lifecycle phases of Configuration, Deployment and Execution. CAMEL is a reflection of the live state of a PaaSage application and works as an envelope grouping data from specific DSLs throughout the lifecycle.

The main PaaSage architectural components are the Profiler, Reasoner and Adaptor. Each component handles and transforms CAMEL at each lifecycle phase. At the Profiler level, user requirements are expressed as DSLs in PaaSage. The Reasoner populates the CAMEL DSLs with potential infrastructure/deployment solutions and the Adaptor selects a specific target infrastructure/deployment solution and adds DSL information specific to application execution in that environment.

The DSLs are explained in more detail within this document as well as in D2.1.1 [D2.1.1] and the architecture document D1.6.1 [D1.6.1]. The main DSLs will be mapped at the schema and content level with MDDB, and as CAMEL is a superset of these DSLs, the live data from CAMEL will be eventually integrated with MDDB. The latter will enable CAMEL data that are stored in the MDDB to be exported and exploited by the various PaaSage components. For instance, CAMEL application data from previous executions would be analyzed at the DSL level, thus allowing decision making in the Reasoner component where (execution) data from previous applications is used to Reason and Simulate potential solutions for new or existing applications.

While CAMEL is being designed and realized, we believe that by being able to map the main DSLs to CAMEL, we will be also able to map CAMEL data to MDDB as the latter will require just adjusting the current mappings as well as producing new ones just for the information that might be not captured by the DSLs but only by CAMEL. To this end, we leave the integration/mapping of CAMEL to MDDB as a future task that will be implemented after M18 and will be documented in the next deliverable of WP4.

3.1.2 User, Role and Organization Aspect - CERIF

The main goal of Task 4.3 in PaaSage is to enhance the metadata collection by integrating additional third-party information sources with Metadata Database. This activity was introduced with the extension of PaaSage. The first prototype focused on integrating CERIF (Common European Research Information Format) in order to improve user and organization modelling within MDDB to better support social network platform, accounting and security features of entire PaaSage platform.

3.1.2.1 Extension and Integration of MDDB with CERIF

CERIF is a modelling framework for describing organizations, people, projects and other aspects related to research domain. It is a EU recommendation[4] for information systems related to research databases, in order to standardize research information and foster research information exchange. Within Task 4.3 of PaaSage, a selected subset of CERIF has been integrated with MDDB for the purpose of improved user and organization modelling.

3.1.2.1.1 CERIF Overview and Advantage for PaaSage

CERIF is a conceptual model with some meta-modelling features, allowing for very detailed description of organizations, users, publications, facilities and research projects. It consists of a data model, a metadata model and an XML exchange model.

[4] http://cordis.europa.eu/cerif/

The data model is defined using an entity-relationship model in a highly normalized manner, which enables efficient representation and storage in relational databases. Figure 2 presents the general overview of CERIF concepts and their relations. The relations in CERIF are represented using separate tables (called link entities), not only to allow many-to-many relationships, but to also allow for adding provenance information to relations such as annotating relations with custom types or adding temporal information (start date and end date) to them.

The metadata model, called CERIF Semantic Layer, provides means to annotate entities and relations with custom or third-party taxonomies. The taxonomy entries can only be added to link entities (both unary and binary) through the special cfClass attribute that allows identifying both the class term as well as the taxonomy identifier, which allows mixing several taxonomies in a single CERIF model.

Furthermore, in order to enable easier information exchange, CERIF provides an XML exchange model[5], which allows encoding of subsets of CERIF repositories in XML for importing and exporting CERIF data between repositories in a feasible way. The main difference between relational and XML models is that in XML the entity relations can be represented by embedding tags within the relations' parent entities.

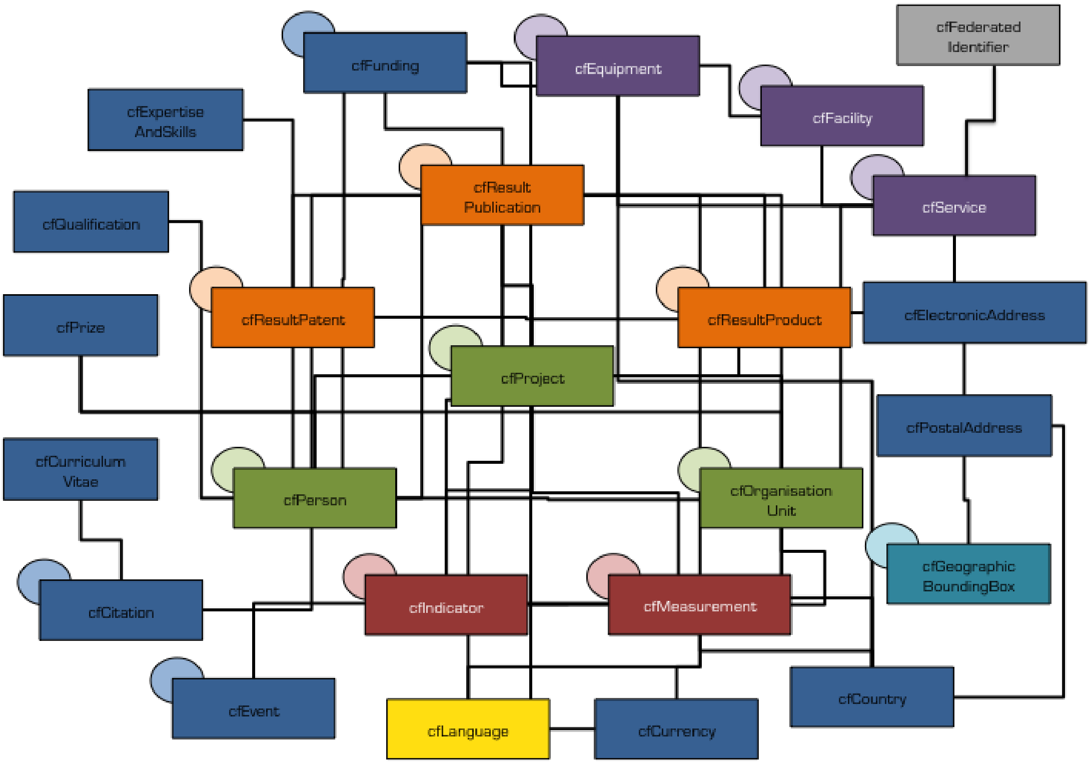


Figure 2 - High level view of CERIF entities[6]

From the point of view of PaaSage, several use cases for improving metadata representation in the project using CERIF have been analysed including:

- Advanced users/organization modeling

[5] http://www.eurocris.org/Uploads/Web%20pages/CERIF-1.6/CERIF_1.6_2.xsd
[6] http://cerifsupport.org/cerif-in-brief/

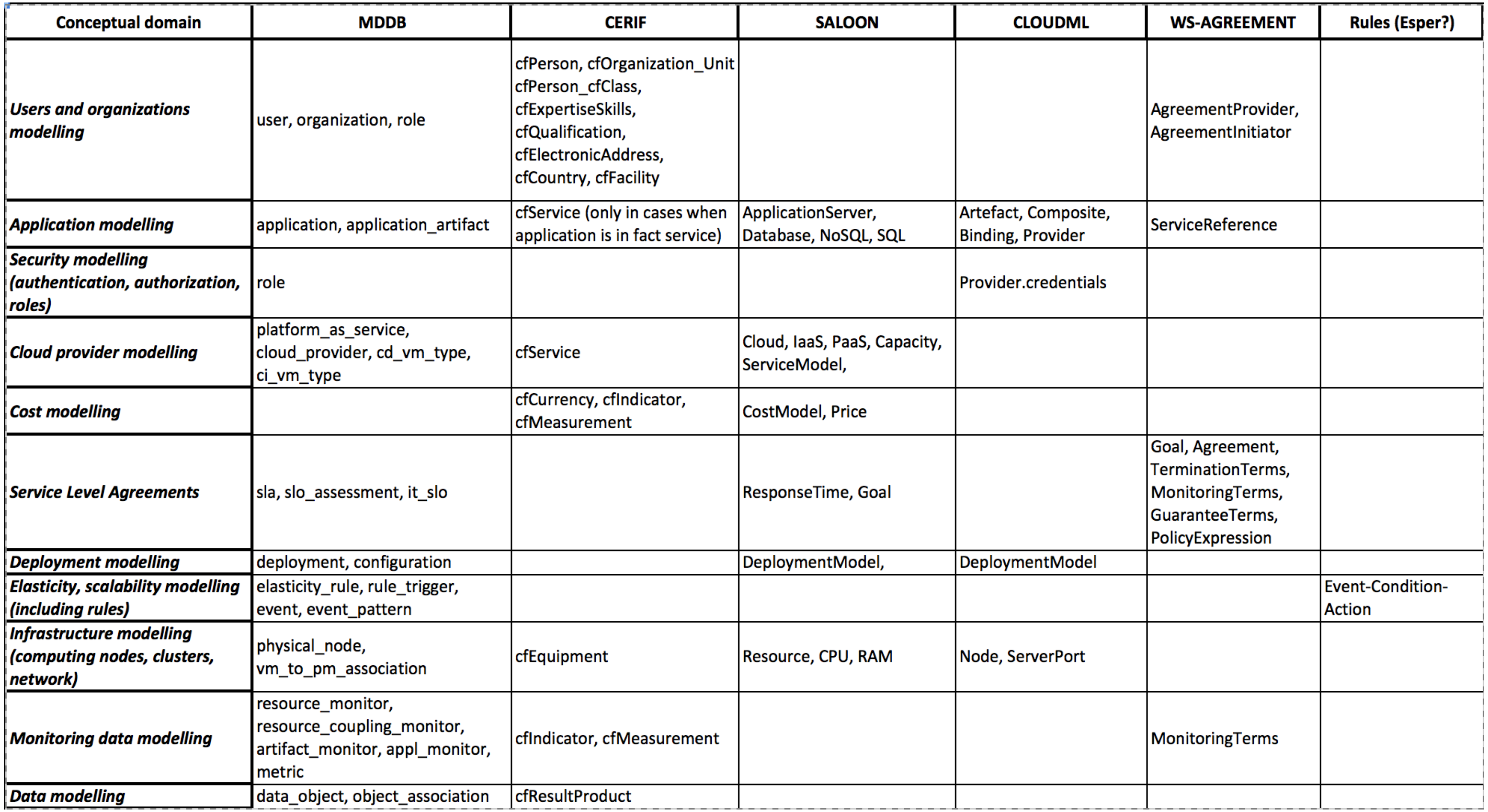
- Trust, reputation, roles
- Accounting, billing
- Provenance tracking, usage statistics
- Resource discovery

In particular, the following CERIF entities were identified as potentially useful for PaaSage:

1. cfCountry – identification of countries
2. cfClass, cfClassification – provide means for adding or referencing existing classification schemes with entities in CERIF (for instance we can have some classification for services, organisations or people which is not part of the CERIF metamodel), the core of the CERIF Semantic Layer
3. cfCurrency – useful for accounting/billing
4. cfDublinCore – provides means for adding Dublin Core metadata to CERIF entities
5. cfEAddr – electronic address
6. cfEquip – equipment information, could be used for information about organisations hardware resources
7. cfFacil – organisations facilities
8. cfFedId – federated identifier, provides the means for creating globally unique ID's for entities (which otherwise are only unique within a single DB)
9. cfIndic – indicator entity (e.g. ratio of some sort)
10. cfMeas – abstract measurement entity
11. cfMetrics – metrics entity
12. cfOrgUnit – entity for modelling organisations and organisation units
13. cfPAddr – postal address, relates to organisations and persons
14. cfPers – person entity
15. cfQual – qualification attributes of a person, can be used for trust/reputation
16. cfSrv – service entity
17. cfResultProduct – data sets produced by applications

The most important of these entities for now correspond to the modelling of users and organizations. In particular, it has been decided that the MDDB schema will be extended with additional entities for organization and user modelling inspired by CERIF schema and XML exchange format will be used to import CERIF data into MDDB.

In order to identify formally how CERIF model overlaps and complements the MDDB, an analysis of the MDDB (and CAMEL in general) with respect to CERIF has been performed. It is summarized in Table 2. As it can be seen, CERIF provides extensive means for describing user/organization modelling aspects, but also some other conceptual domains, such as services, measurements, indicators, and products, which, if necessary, could be added in the future by creating the appropriate mapping against the MDDB schema.



Conceptual domain	MDDB	CERIF	SALOON	CLOUDML	WS-AGREEMENT	Rules (Esper?)
Users and organizations modelling	user, organization, role	cfPerson, cfOrganization_Unit cfPerson_cfClass, cfExpertiseSkills, cfQualification, cfElectronicAddress, cfCountry, cfFacility			AgreementProvider, AgreementInitiator	
Application modelling	application, application_artifact	cfService (only in cases when application is in fact service)	ApplicationServer, Database, NoSQL, SQL	Artefact, Composite, Binding, Provider	ServiceReference	
Security modelling (authentication, authorization, roles)	role			Provider.credentials		
Cloud provider modelling	platform_as_service, cloud_provider, cd_vm_type, ci_vm_type	cfService	Cloud, IaaS, PaaS, Capacity, ServiceModel,			
Cost modelling		cfCurrency, cfIndicator, cfMeasurement	CostModel, Price			
Service Level Agreements	sla, slo_assessment, it_slo		ResponseTime, Goal		Goal, Agreement, TerminationTerms, MonitoringTerms, GuaranteeTerms, PolicyExpression	
Deployment modelling	deployment, configuration		DeploymentModel,	DeploymentModel		
Elasticity, scalability modelling (including rules)	elasticity_rule, rule_trigger, event, event_pattern					Event-Condition-Action
Infrastructure modelling (computing nodes, clusters, network)	physical_node, vm_to_pm_association	cfEquipment	Resource, CPU, RAM	Node, ServerPort		
Monitoring data modelling	resource_monitor, resource_coupling_monitor, artifact_monitor, appl_monitor, metric	cfIndicator, cfMeasurement			MonitoringTerms	
Data modelling	data_object, object_association	cfResultProduct				

Table 2 - Conceptual correspondence between CERIF, MDDB and CAMEL

3.1.2.1.2 Specification of MDDB Schema Extensions

The MDDB extensions based on CERIF are focused on improving the user and organization modelling aspects in PaaSage. At the centre of the extended schema is the concept of ‘user’ which represents any user which has access to the PaaSage system, either through the social network, Eclipse-based editors or any other way (e.g., the object-relational interface for PaaSage developers).

An overview of MDDB extensions include:

- Additional external identifiers can be assigned to users (e.g., SAML2, OpenID, Social logins, etc.)
- Support for different types of external identifiers through a separate table: *user_identifier_type*
- Users can belong to multiple organisations with different roles
- User can be organized in groups such that these groups can also be assigned to specific roles for a particular organization
- Data centres can have locations
- *cloud_provider* is a subtype of organisation, i.e., it’s primary key is inherited from table *organisation* and is used as relationship id for *physical_node* and *platform_as_service*
- VM types (mapping to *cd_vm_type* table) are now associated to a data centre owned by the respective cloud provider. This is a design choice selected for two main reasons: (a) different performance has been demonstrated for the same VM types in different cloud provider data centres and (b) different pricing is induced for the same VM type in different data centres of the same cloud provider.
- A separate *location* table is used to list data centre locations.
- *cloud_provider* can have several boolean attributes depending on the type of services offered: public, IaaS, PaaS, SaaS which can be useful for classifying providers
- Multiple roles per user are possible through *user_has_role* relationship
- Basic RBAC (Role Based Access Control) through tables *role*, *permission*, and *action*
- Permissions assign to a role the right to perform one or more actions (see also *permission_action* table used to associate a permission to more than one actions) on one or more resources and are issued by organizations
- Resources can be organized in groups such that we can create permissions also for these groups. To this end, resource groups are also (a special type of) resources. Thus, all entities for which permissions can be created have been made sub-entities of the resource entity/class.
- Table *allowed_actions* has been created in order to list all types of actions that can be allowed on a particular resource. This table can be used as a guide for developing permissions but also as a validation tool in order to discard invalid permissions in terms of the actions that they allow on the designated resources.

A detailed mapping between the considered subset of CERIF and MDDB is defined in Table 3. Several of the attributes required by MDDB (for instance role name or cloud provider type) are not directly represented in CERIF but can be inferred if CERIF entities are annotated with a proper taxonomy. Figure 3 presents the entity-relationship diagram of the new tables introduced in the MDDB.

MDDB Table	**Attribute**	**CERIF property**
user	Lastname	cfPers::cfPersName::cfLastNames
	Firstname	cfPers::cfPersName::cfFirstNames
	Email	cfPers::cfPers_EAddr::cfEAddr::cfURI *(select proper attributed based on cfClass)*
	www	cfPers::cfPers_EAddr::cfEAddr::cfURI *(select proper attributed based on cfClass)*
	Login	-
organization	Name	cfOrgUnit::cfOrgUnitName
	www	cfOrgUnit::cfOrgUnit_EAddr::cfEAddr::cfURI
	postal_address	cfOrgUnit::cfOrgUnit_PAddr::cfPAddr
	Email	cfOrgUnit::cfOrgUnit_EAddr::cfEAddr::cfURI
cloud_provider	Public	*(decide based on cfClass if available)*
	Paas	*(decide based on cfClass if available)*
	Iaas	*(decide based on cfClass if available)*
	Saas	*(decide based on cfClass if available)*
location	City	cfPAddr::cfCityTown
	Country	cfPAddr::cfCountryCode
	contry_code	cfPAddr::cfCountryCode
	Latitude	cfPAddr_GeoBBox::cfGeoBBox::cfSBLat cfPAddr_GeoBBox::cfGeoBBox::cfNBLat
	Longitude	cfPAddr_GeoBBox::cfGeoBBox::cfWBLong cfPAddr_GeoBBox::cfGeoBBox::cfEBLong
	Address	cfPAddr::cfAddrline1, …, cfPAddr::cfAddrline5
data_center	Name	cfFacil::cfFacilName
	location_id	cfFacil::cfFacil_PAddr
user_identifier_type	Name	cfFedId::cfClass::cfURI
	Description	cfFedId::cfClass::cfClassDescr
user_external_identifier	external_identifier	cfFedId::cfInstId
role	Name	cfPers::cfPers_Class::cfClass

		(identify role depending on taxonomy)
	issued_by	-
user_has_role	Start	cfPers::cfPers_Class::cfStartDate
	End	cfPers::cfPers_Class:cfEndDate
user_group	Name	cfPers::cfPers_Class::cfClass *(identify role depending on taxonomy)*
user_group _has_role	Start	cfPers::cfPers_Class::cfStartDate
	End	cfPers::cfPers_Class:cfEndDate

Table 3 - CERIF to MDDB detailed mapping

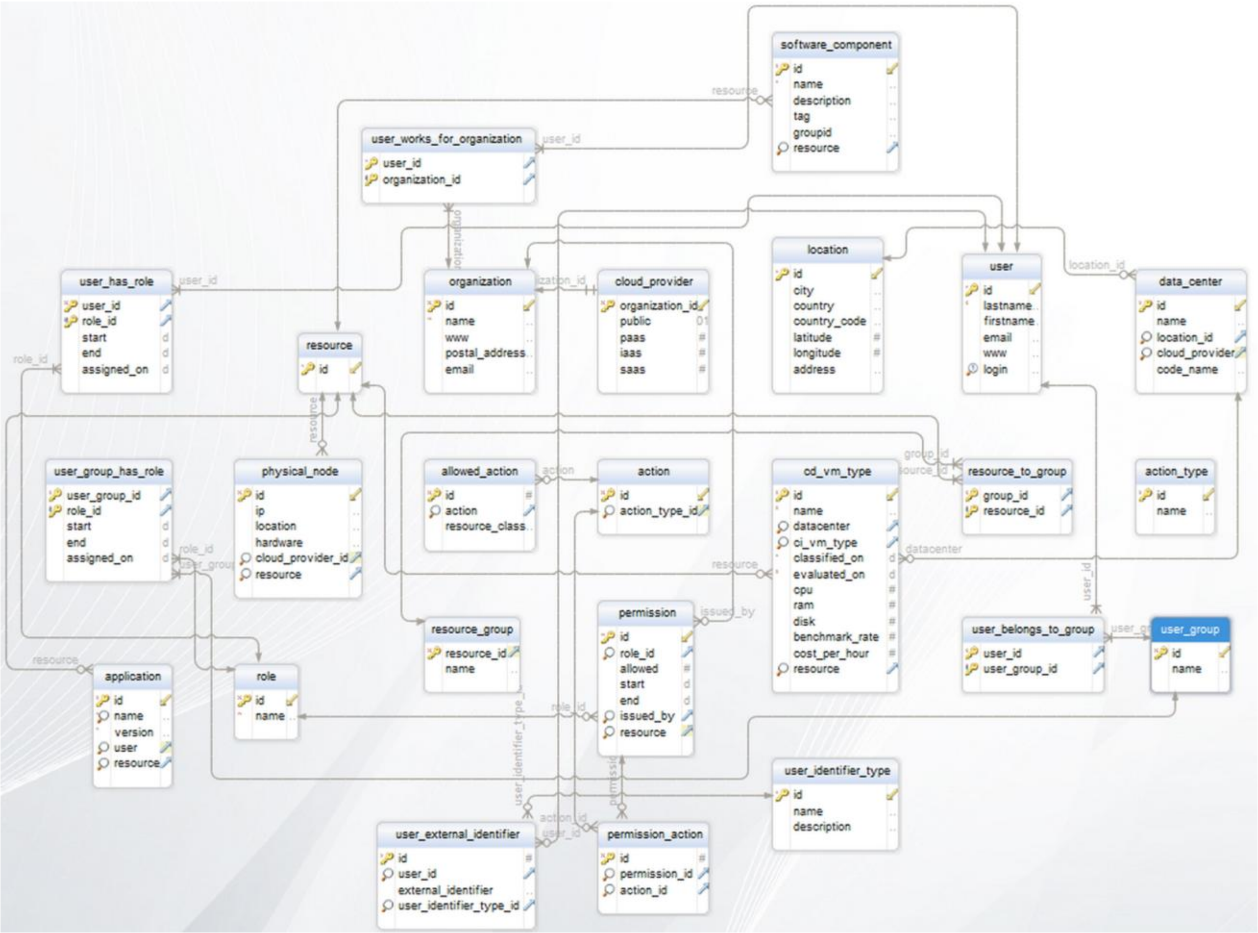


Figure 3 - User and organization modelling entities in MDDB

3.1.2.2 Prototype integration of Metadata Database with CERIF repositories

The prototype mapping from CERIF to MDDB has been realized in the form of a command line tool develop in the Clojure language executable through Java Virtual Machines. The tool allows to import user and organizational models represented in CERIF XML exchange format with optional CERIF taxonomies for inferring additional information (such as organization types or user roles) during data conversion process. An example mapping from a simple user model (Figure 4) and taxonomy (Figure 5) to the respective SQL code to be executed over MDDB (Figure 6) is provided to showcase the functionality of the tool.

```
…
<cfPAddr>
        <cfPAddrId>131.1</cfPAddrId>
        <cfCountryCode>PL</cfCountryCode>
        <cfAddrline1>Nawojki 11</cfAddrline1>
        <cfPostCode>30-095</cfPostCode>
        <cfCityTown>Krakow</cfCityTown>
</cfPAddr>
…
<cfOrgUnit>
        <cfOrgUnitId>123</cfOrgUnitId>
       <cfAcro>CYFRONET</cfAcro>
        <cfOrgUnit_Class>
                <cfClassId>CAMEL.IaaSCloudProvider</cfClassId>
                <cfClassSchemeId>CAMEL</cfClassSchemeId>
        </cfOrgUnit_Class>
        <cfOrgUnit_PAddr>
                <cfPAddrId>131.1</cfPAddrId>
                <cfClassId>aaa</cfClassId>
                <cfClassSchemeId>bbb</cfClassSchemeId>
        </cfOrgUnit_PAddr>
</cfOrgUnit>
…
<cfPersName>
        <cfPersNameId>persname-id1</cfPersNameId>
         <cfFamilyNames>Fontnot</cfFamilyNames>
         <cfFirstNames>Todd</cfFirstNames>
</cfPersName>
<cfPers>
         <cfPersId>pers-id1</cfPersId>
         <cfGender>m</cfGender>
         <cfPers_Class>
             <cfClassId>CAMEL.Administrator</cfClassId>
             <cfClassSchemeId>CAMEL</cfClassSchemeId>
         </cfPers_Class>
         <cfPersName_Pers>
             <cfPersNameId>persname-id1</cfPersNameId>
         </cfPersName_Pers>
         <cfPers_EAddr>
             <cfEAddrId>ToddMFontenot@dayrep.com</cfEAddrId>
             <cfClassId>35d43364-2160-4b6c-a487-5019458321e8</cfClassId>
             <cfClassSchemeId>05cc5ff9-bc58-4743-ab59-46e5013e0039</cfClassSchemeId>
         </cfPers_EAddr>
         <cfPers_OrgUnit>
             <cfOrgUnitId>123</cfOrgUnitId>
             <cfClassId>ebd55ab0-1cfc-11e1-8bc2-0800200c9a66</cfClassId>
             <cfClassSchemeId>e9616dbd-0d38-4b7d-a6cd-3c4df1e95462</cfClassSchemeId>
             <cfStartDate>2012-06-01T00:00:00</cfStartDate>
        </cfPers_OrgUnit>
```

Figure 4 - Example organization description in CERIF

```
…
<cfClassScheme>
<cfClassSchemeId>CAMEL</cfClassSchemeId>
<cfName cfLangCode="en" cfTrans="o">CAMEL</cfName>
<cfDescr cfLangCode="en" cfTrans="o">PAASAGE CAMEL schema in CERIF.</cfDescr>
<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>
<cfClass>
        <cfClassId>CAMEL.Developer</cfClassId>
        <cfTerm cfLangCode="en" cfTrans="o">Developer</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>
<cfClass>
        <cfClassId>CAMEL.CloudProvider</cfClassId>
        <cfTerm cfLangCode="en" cfTrans="o">CloudProvider</cfTerm>
        <cfDef cfLangCode="en" cfTrans="o">Cloud provider type of
organization</cfDef>
</cfClass>
<cfClass>
        <cfClassId>CAMEL.IaaSCloudProvider</cfClassId>
        <cfTerm cfLangCode="en" cfTrans="o">IaaSCloudProvider</cfTerm>
        <cfDef cfLangCode="en" cfTrans="o">IaaS Cloud provider type of
organization</cfDef>
        <cfClass_Class>
                <cfClassId2>subClassOf</cfClassId2>
                <cfClassSchemeId2>RDFS</cfClassSchemeId2>
                <cfClassId>CAMEL.CloudProvider</cfClassId>
                <cfClassSchemeId>CAMEL</cfClassSchemeId>
        </cfClass_Class>
</cfClass>
```

Figure 5 - Excerpt form CERIF taxonomy representing CAMEL entities

```
…
INSERT INTO `role`
       VALUES (25,'CAMEL.Administrator'),(26,'CAMEL.Developer');
INSERT INTO `organization`
       VALUES (11,'CYFRONET',NULL,'Nawojki 11, 30-095, Krakow',NULL);
INSERT INTO `user`
       VALUES (11,'Fontnot','Todd','ToddMFontenot@dayrep.com',NULL,'pers-id1');
INSERT INTO `user_has_role`
       VALUES (11,25,NULL,NULL,NULL);
INSERT INTO `user_works_for_organization`
       VALUE (11,11);
…
```

Figure 6 - Excerpt of SQL updates on MDDB based on example CERIF description

3.1.3 Application Specification & Deployment Aspect - CloudML

CloudML (Cloud Modeling Language) is a domain-specific language (DSL) for modelling and enacting the provisioning and deployment of multi-cloud applications. With CloudML we can specify the topology of virtual machines and applications components.

CloudML proposes two levels of abstraction for the provisioning and deployment models of applications. The Cloud Provider-Independent Model (CPIM) expresses the provisioning and the deployment of a multi-cloud application in a cloud (provider)-independent way. On the other hand the Cloud Provider-Specific Model (CPSM) refines the CPIM in order to express the provisioning and deployment of the application in a single or multi-cloud setting (for one or more specific cloud providers).

The CloudML language includes the following concepts:

- *Cloud*: a collection of virtual machines offered by a cloud provider
- *Virtual machine type* and *virtual machine instance* where the latter represents an instance of the former. A virtual machine type refers to a generic description of a virtual machine, while an instance of a VM type maps to a specific instantiation of a VM including specific configuration information.
- *Application component* which represents a reusable type of application component, while an *application component instance* represents an instance of an application component. Similarly to the specification of VM information, the description of an application component stays at a generic level while the specification of its respective instances involves particular configuration information.
- *Port* which represents an interface of a feature of an application component. This interface may be offered or required by an application component.
- *Relationship* representing a relationship between ports of two application components (which can be of communication or containment type).

The MDDB schema is capable of capturing all of the above information as it includes a set of tables which can be used to capture information about cloud providers and the virtual machines that they offer, the application characteristics, requirements and structure as well as the resources used by an application according to its deployment plan. The next sub-section demonstrates the mapping between CloudML and the MDDB schema.

3.1.3.1 Running/Mapping example

In this section we demonstrate the usage of CloudML and the mapping of its data in the MDDB using the CloudML running example of SENSAPP described in deliverable D2.1.1 [D2.1.1]. This example will be used across the whole Section 3.1 to demonstrate the mappings also for the other DSLs that are going to be used by the project.

In Figure 7, we can see the software components constituting the SENSAPP application, as described with the CPIM visual syntax of CloudML. These components are the servlets Dispatcher, Notifier, and Admin and MongoDB. The servlets are hosted by generic servlet containers, which in turn are contained in generic virtual machines. The Dispatcher depends on MongoDB. The communication between these components is mandatory. In addition, the Dispatcher and MongoDB have to be deployed on the same virtual machine. Finally, the Dispatcher communicates with Notifier, while the Admin communicates with MongoDB. The latter communications are optional.

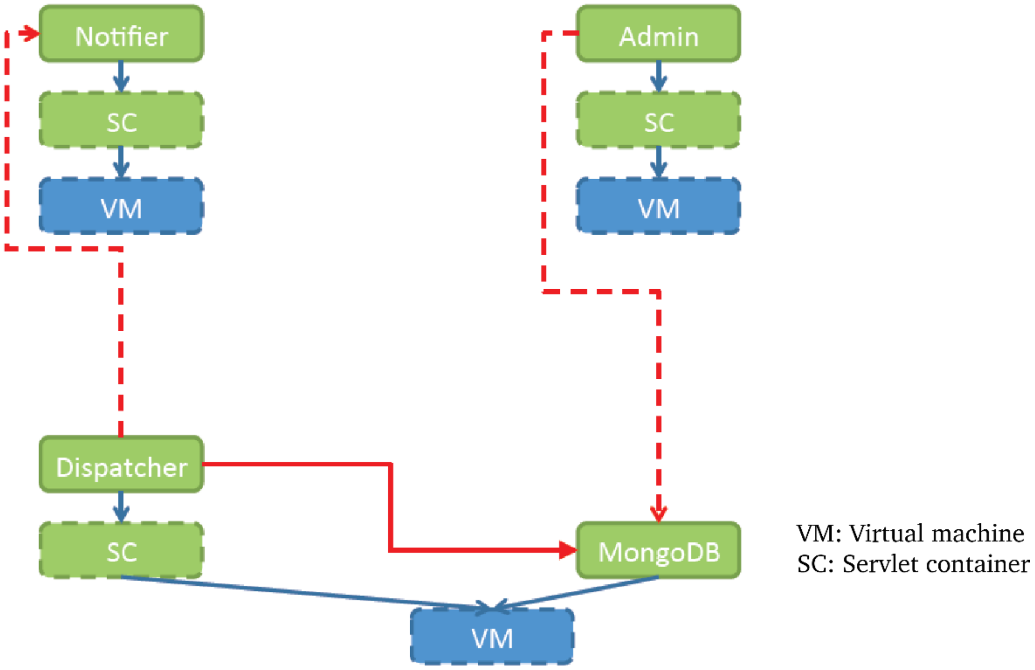


Figure 7 - Sample CPIM model of CloudML for SENSAPP

Figure 8 shows a subset of the MDDB tables that can store the information described previously with the CloudML visual syntax. In particular, the generic abstract structure of the application can be modelled using the tables on the first rectangle at the upper left corner of the figure. These tables are the *software_component* (refers to a more abstract description of the application's software components, e.g., a generic servlet component) and *application* ones which are used to model applications and their corresponding components, respectively. A software component is characterized by its id, name, (textual) description, and tag (could map to a taxonomy node for software component types). On the other hand, an application has an id, name and version, while it belongs to a certain user. Based on the latter modelling, we capture the versions of a particular applications in order to have a clear view of how an application evolves and what are the main benefits of this evolution in terms of performance with respect to specific deployment solutions. Apart from these two tables, there is also the *application_components* table which associates software components to applications as there can be an M-to-N relationship between these two entities. This also enables the re-use of components across different applications.

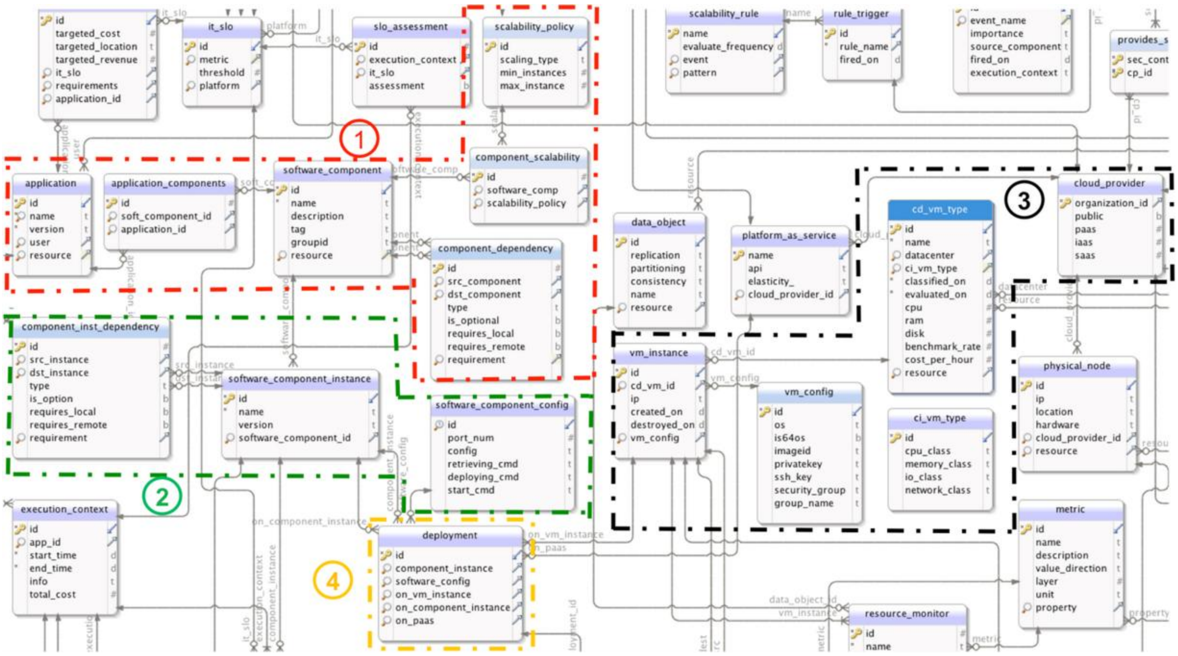


Figure 8 - Part of the MDDB schema used to capture CloudML information

In the considered (running) example, the software components of the application include the Dispatcher, Notifier, and Admin servlets as described in the CPIM of CloudML. Each software component may depend on or require other software components. As these are types of component dependencies, we have decided to model this relationship with the table *component_dependency* which specifies if the component dependency is mandatory (see *is_optional* attribute of the table) signifying that the required component has to be deployed before the component that requires it. In addition, properties *requires_local* and *requires_remote* of the table specify if the components should or should not be contained on the same virtual machine. Please have in mind that a false value on one of these two properties does not map to a true value for the other. Thus, if property *requires_local* has a false value, this means that the application user does not require that the respective components should be deployed on the same virtual machine, so the PaaSage reasoner is free to take any possible decision concerning the placement of these components. As different types of dependencies may exist (e.g., communication and containment), a particular column (*type*) has been inserted in the *component_dependency* table to capture a dependency's type. Please bare in mind that the component containment dependency type is considered as a mandatory requirement which must lead to the collocation of the related components in the same VM. Another important feature of a software component regards its scalability policy. For instance, in the running example, we could have two scalability policies: (a) one dictating that a particular component (i.e., the Dispatcher and the MongoDB) can scale horizontally from 1 to 8 instances, (b) and another one signifying that the Notifier component must scale vertically using VMs with 2 to 4 virtual cores, respectively. To capture this information in the MDDB, we use the table *scalability_policy* and associate it with a specific component of the application though *component_scalability* table, where the first table includes all the appropriate scalability policy information, such as the scaling type (horizontal or vertical) and the minimum and maximum amount of instances. This design enables specifying scalability rules/policies irrespectively of any software component and then linking them together via a specific association, thus properly capturing this M-to-N relationship between scalability rules and software components.

According to CloudML, the CPIM model has to be refined into CPSM in order for the application to be properly deployed. This CPSM specifies the provisioning and deployment in a cloud provider-specific way. Figure 9 shows a particular deployment of the SENSAPP application specified graphically via CloudML. The executionware deploys the application according to the depicted CPSM model, creating instances of each software component.

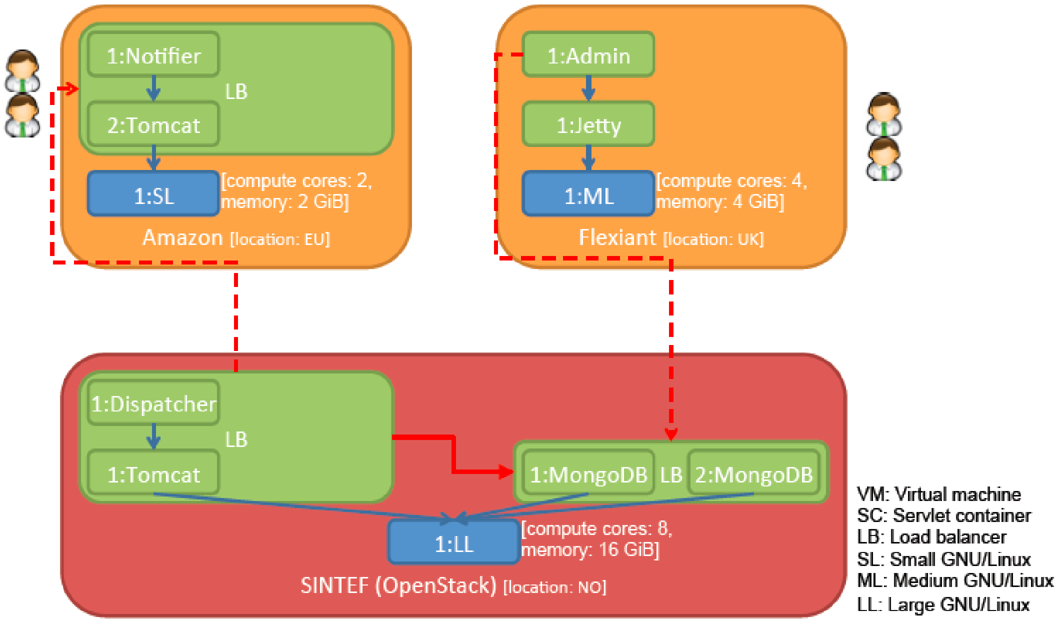


Figure 9 - A snapshot of the CPSM at run-time

These instances have more specific description and configurations than the ones able to be specified in the generic *software_component* table. To this end, the MDDB schema includes the tables *software_component_instance* and *software_component_config*, depicted in the second (green) rectangle of Figure 8, which are dedicated to modelling information regarding software component instances and their configurations. Through the latter table, we also assist the user (e.g., the one that does not own the application but intends to run it) in re-using configurations for components in application deployment and not creating them from scratch. Each software instance realizes a specific abstract software component (e.g., a Tomcat application server realizes a generic servlet container). As a result it inherits its properties as well as its (component) dependencies. However, there can also be dependencies between component instances that have not been captured in the higher-level of the respective components. To this end, a new table has been created, named as *component_inst_dependency*, with identical content with respect to the *component_dependency* table with the sole exception that this table references component instances and not components. We should also mention that we consider component and component instance dependencies as application requirements and thus a specific column mapping to a foreign key to the requirements table has been created.

CloudML also specifies the VMs on which the application is deployed along with the cloud providers that offer them. On the right part of Figure 8, in the third rectangle, the respective MDDB tables are depicted which are used to fully include this type of information. The *cloud_provider* table stores information about cloud providers regarding whether they offer public or private cloud services. The table includes a reference to the respective organization (as a cloud provider is usually an organization) which models additional information, such as the organization's name and location. Information about a virtual machine instance, such as its name, IP address and lifetime is stored in the *vm_instance* table, while the VM instance configuration properties are modelled through the table *vm_config*. Each node/VM instance can belong to particular cloud-dependent and cloud independent (VM) types (see tables *cd_vm_type* and *ci_vm_type*, respectively, where "cd" stands for cloud dependent and "ci" for cloud independent). The *cd_vm_type* table specifies real-world VM types offered by a Cloud provider (actually one or more data centres of the provider) (see Section 3.1.4), including the description of the various possible VM properties. On the other hand, *ci_vm_type* table stores provider-independent classifications for a particular VM (see Section 3.1.4).

Finally, MDDB table *deployment* (the fourth rectangle) is used to store information about the mapping between an application software component instance and a corresponding VM instance or other software component instance on which it is deployed (e.g., an application component instance can be deployed directly on a VM or on a application container). It is also associated to the configuration of the software component for the deployment to occur.

Below we present a snapshot of the data contained in the respective MDDB tables that models the structure, the requirements and the deployment of the SensApp application according to the above description. With blue colour, we denote new MDDB table content, while with grey colour pre-existing MDDB table content. Please note that this colour convention is used throughout Section 3.1 when the mapping of the models of other DSLs to MDDB schema is showcased via the SensApp example.

Table Application

id	name	version	user	resource
1	SensApp	1.0	4	1

Table *user*:

id	lastname	Firstname	email	www	login
4	User4	CC1	CC1.User4@email.com	...	...

Table software_component

id	name	description	tag
1	Notifier	…	…
2	Admin	…	…
3	Dispatcher	…	…
4	MongoDB	…	…
5	Servlet-Container	…	…

Table application_components

id	software_component	application_id
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1

Table component_dependency

id	src_component	dest_component	type	is_optional	req_remote	req_local	requirements
1	3	4	Communication	false	false	true	
2	3	1	Communication	true	false	false	
3	2	4	Communication	true	false	false	
4	1	5	Containment	false	false	true	
5	2	5	Containment	false	false	true	
6	3	5	Containment	false	false	true	

Table scalability_policy

id	scaling_type	min	max
1	Horizontal	1	8
2	Vertical	2	4

Table component_scalability

id	software_comp	scalability_policy
1	3	1
2	4	1
3	1	2

Table software_component_instance

id	name	software_component
1	Notifier-1	1
2	Admin-1	2
3	Dispatcher-1	3
4	MongoDB-1	4
5	MongoDB-2	4
6	Tomcat-1	5
7	Tomcat-2	5
8	Jetty-1	5

Table cloud_provider

organization_id	public	paas	iaas	saas
1	true	true	true	false

Table *data_center*:

id	**name**	**location_id**	**cloud_provider**	**code_name**
1	DC Ireland	1111	1	DCI

Table *ci_vm_type*:

id	**cpu_class**	**memory_class**	**io_class**	**network_class**
111	medium	small	small	low
112	medium	medium	medium	medium
113	high	medium	medium	high

Table *organization*:

id	**name**	**www**	**postal_address**	**email**
1	Amazon	www.amazon.com	...	...

Table *cd_vm_type*:

id	**name**	**datacenter**	**ci_vm_type**	**classified_on**	**evaluated_on**
11	SL	1	111	...	...
12	MM	1	112	...	...
13	LL	1	113	...	...

Table *cd_vm_type* (*cont.*):

id	**cpu**	**ram**	**disk**	**benchmark_rate**	**cost_per_hour**	**resource**
11	2	2	100	...	0.5	7
12	4	4	400	...	0.7	8
13	8	16	600	...	0.9	9

Table vm_instance

id	name	cd_vm_type	ip	created_on	destroyed_on	vm_config
1	SL-1	11	…	…	…	…
2	MM-1	12	…	…	…	…
3	LL-1	13	…	…	…	…

Table deployment

id	component_ instance	software_ config	on_vm_ instance	on_component instance	on_paas
1	1	…		6	
2	2	…		8	
3	3	…		7	
4	4	…	3		
5	5	…	3		
6	6	…	1		
7	7	…	3		
8	8	…	2		

3.1.4 Cloud Provider Profiling Aspect - Saloon

Here we provide first a high level description of the Saloon meta-model, then we identify the respective MDDB schema part and finally provide visually the mapping between the Saloon meta-model and the MDDB schema along with a particular example validating it.

DSL Description

Saloon is a model-based framework (see Deliverable D2.1.1 [D2.1.1]) through which cloud requirements can be matched with cloud offerings so as to identify particular cloud services which can be used for the deployment of a specific cloud application. This framework is accompanied by a particular cloud ontology which can be used to fix the vocabulary between the cloud offerings and requirements and thus enabling a better matching of them. The framework offers a generic feature meta-model used for describing cloud-provider offerings. Actually this meta-model is so generic that can be used even in the context of other fields or domains. The expressivity of the meta-model is quite powerful as it is able to describe different combinations of offerings for particular cloud providers as well as introduce constraints which can be used as a guide for the matching and the deployment of applications. This generality and expressivity, however, should be accompanied by more domain-specific constructs so as to enable the mapping of feature models to the domain of the cloud and to the respective elements and attributes that can be used to describe/annotate features.

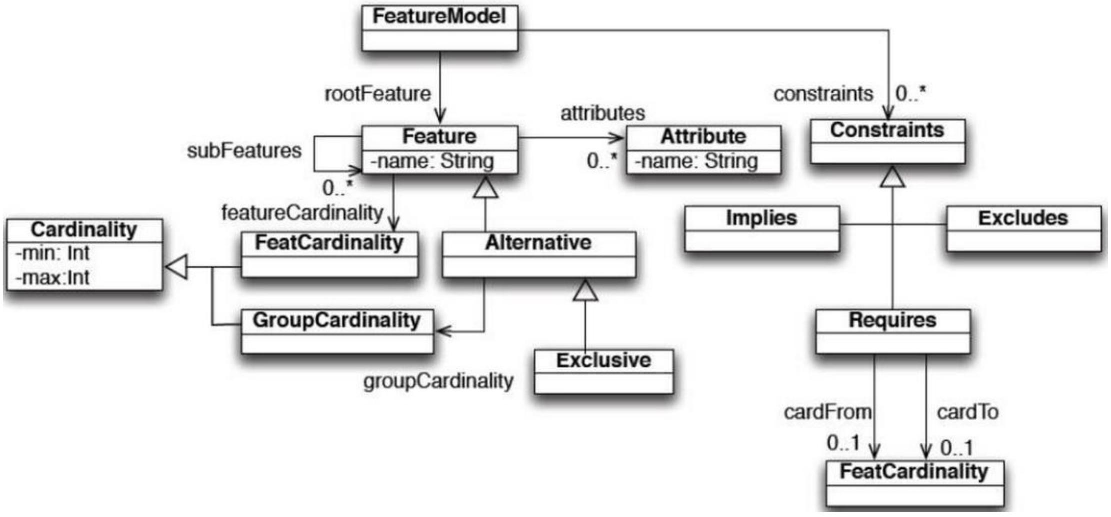


Figure 10 - The Saloon feature meta-model

The feature meta-model of Saloon is depicted in Figure 10. As it can be seen, any feature model requires the appearance of a root Feature which can be mapped to a specific cloud provider. Then, the sub-features of this root feature denote the cloud services that the respective provider offers. A Feature is related to a set of attributes which attempt to characterize its main properties/characteristics. For instance, a particular VM type offered by a cloud provider might be characterized by the size of the main memory and the frequency of the CPU. The feature-model is quite expressive allowing the specification of different types of (sub-)feature combinations:

- a combination of alternative sub-features covers the case where the main feature can be realized/configured through the selection of one or more features. For instance, a cloud provider might offer a Database cloud service which can be realized in two different ways (mapping to the respective sub-features of the Database feature): an SQL or a NoSQL database.
- a xor (exclusive or) combination of alternative sub-features covers the case where only one (sub-feature) configuration/realization of a feature can be actually selected. For instance, concerning the SQL database feature, only one from all possible SQL realizations can be selected (e.g., MySQL or PostgreSQL).
- a simple (and) combination of sub-features means that the main feature comprises particular sub-features that characterize it. This case can be separated into two sub-cases: (a) the sub-features can be offered as standalone services and (b) all sub-features should be considered as a uniform combination which is required for the proper realization of the feature.

Features also have cardinalities and constraints. Cardinalities indicate the amount of instances for a feature for a particular application/product configuration, while constraints usually indicate inter-feature or inter-attribute dependencies, i.e., requirements for other features imposed for the proper offering of the feature or constraints restraining the combination of values that the attributes of a feature can take. For instance, a constraint might indicate that four individual instances of Tomcat require the existence of a load balancer.

As it can be understood for the above analysis, the Saloon's feature meta-model has important modelling capabilities which allow flexibility in the specification of the features or services offered by a particular cloud provider and their combination. However, it has the following disadvantages that need to be covered:

- domain-specific terms need to be entered and considered which could be regarded as belonging to a lower modelling level. The Saloon ontology could be used for this purpose but this ontology is just a simple hierarchy of terms attempting to fix a specific vocabulary for particular cloud terms which is not enough as, e.g., the values of the feature attributes should have a particular domain to which they should conform. This would enable the proper mapping to the cloud domain as well as the capability to check the correctness of the models derived from the feature meta-model.
- there is a lack of particular information which is highly-required in order to enable the proper matching of cloud requirements and capabilities. This information includes the time validity of a particular offering, its pricing as well as particular means for rating such an offering. For the latter, it has been already proposed and proved that the matching of VMs solely based on their capabilities is not enough and correct as even VMs with the same characteristics might have a different behaviour in different platforms. To this end, direct rating information according to a particular VM dimension (e.g., CPU frequency) could be highly beneficial for the matchmaking process as it would lead to a more fair ranking of cloud provider offerings. Such information could be produced through benchmarking.

Description of Respective MDDB Schema Part

The overall, related to Saloon tables in the MDDB schema and their relationships are depicted in Figure 11. Two main tables are used for the description of the two types of offerings that a cloud provider can supply: *cd_vm_type* which maps to the description of virtual machine types (IaaS) and *platform_as_service* which maps to the description of platform as a service offerings. Virtual machine types are characterized by the usual information of number of CPUs and amount of RAM and disk storage. Moreover, their location is specified as well as a reference to the description of a fair ranking for them described by a specific row of the *ci_vm_type* table. The latter row indicates the uniform and fair ranking of a VM according to different VM dimensions, such as some from those indicated (CPU, RAM) above plus the I/O and network one (mapping to particular latency ratings). Apart from these, the description of a VM includes the rate value for a particular benchmark, the cost of the VM per hour as well as the cloud provider which offers the particular VM. Concerning the description of platform as service offerings, the information that can be specified currently includes a textual description of the PaaS API and some elasticity capabilities.

Apart from these two tables, two other MDDB tables are quite important and need to be considered. The first one is the *cloud_provider* which maps to a root node of a feature model and the second one is the *data_center* which maps to the data centres owned by the cloud provider for which particular VM types are offered. The existence of the latter table and its association to the *cd_vm_type* table unveils the necessity to include data centre information in feature models. Such information could be provided in two alternative ways: (a) a cloud service is offered in a particular data centre (so the reference to the data centre could be a particular feature attribute) and (b) a root node (cloud provider) is first associated to the data centres which are associated in turn to the particular cloud services that they offer and support.

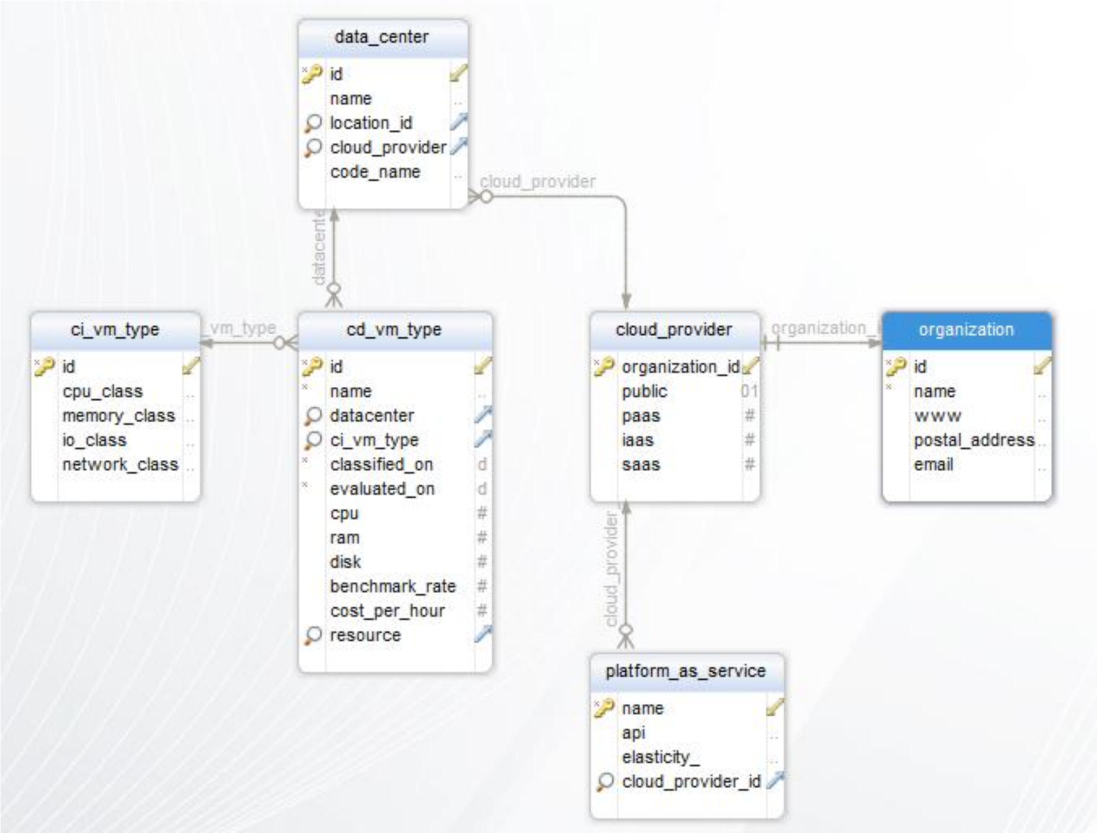


Figure 11 - The respective MDDB part capturing cloud provider offerings

From the above analysis, it is apparent that the MDDB schema provides information that is either missing or can be regarded as missing with respect to the Saloon feature meta-model, such as cost and rating information. On the other hand, the Saloon feature meta-model is richer in describing the way service offerings can be combined as well as enable the introduction of particular constraints which restrain the context under which the respective cloud services can be selected and used. Thus, the two schemata need to be closely aligned with each other in order to have both of them capable of describing the same amount and type of information. This would, of course, facilitate the mapping of models from one schema/meta-model to the other. Figure 12 shows the current mapping of the two considered schemata where, as it can be easily deduced, the fact that some information inherent in one schema is missing from the other and vice versa is more than apparent. Moreover, it is also clear that the mapping is not so straightforward and certainly requires some procedural logic as the same element in the feature model can be mapped to the content of different MDDB tables depending on the information that it actually caries (e.g., a root feature maps to a cloud provider and a non-root feature maps to a cloud service).

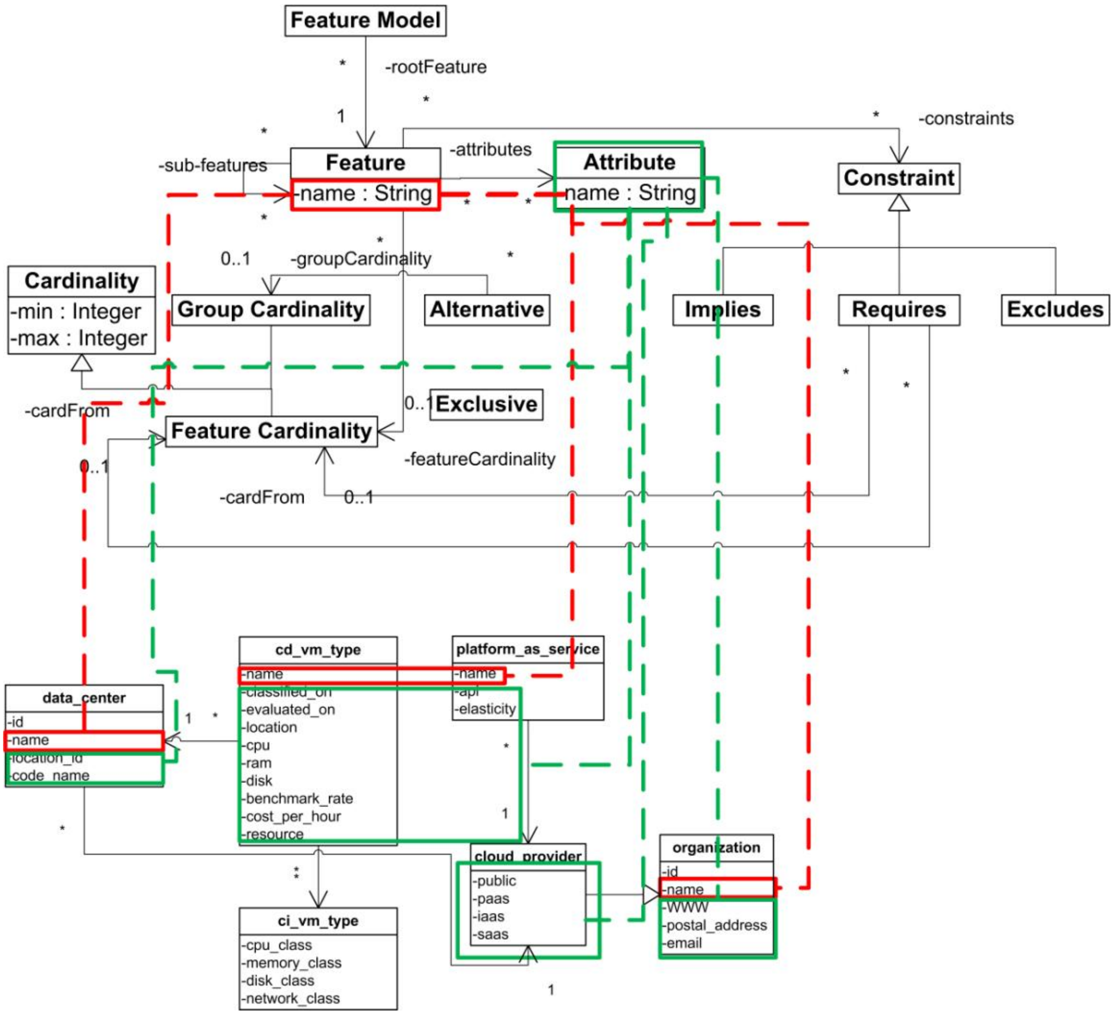


Figure 12 - The mapping of Saloon feature meta-model to MDDB Schema

Mapping Example

Based on the current alignment between the Saloon feature meta-model and the respective part of the MDDB schema, we continue the running example of SENSAPP where we indicate a Saloon feature model for Amazon, conforming to the Saloon feature meta-model, which shows the IaaS offering of this cloud provider for a specific data centre that has been considered for the deployment of the Notifier (and the tomcat container): a particular VM type with the name "SL" which has the following characteristics: CPU: 2 cores, RAM: 2 GB, DISK: 100 GB, and cost: 0.5 $ per hour. Further suppose that based on particular benchmarking classifications, this VM type is rated as (CPU: "medium", RAM: "small", IO: "small", network: "low"). The Saloon feature model is depicted in Figure 13, while the respective MDDB content to be produced is shown in the following tables representation:

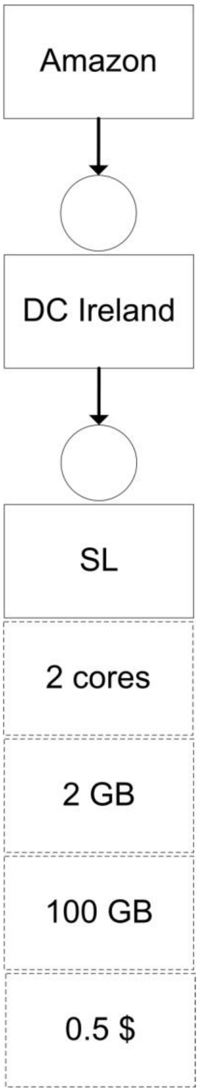


Figure 13 - The Saloon feature model for the running example

Table *cd_vm_type*:

id	name	datacenter	ci_vm_type	classified_on	evaluated_on
11	SL	1	111	...	...

Table *cd_vm_type (cont.)*:

id	cpu	ram	disk	benchmark_rate	cost_per_hour	resource
11	2	2	100	...	0.5	7

Table *data_center*:

id	name	location_id	cloud_provider	code_name
1	DC Ireland	1111	1	DCI

Table *ci_vm_type*:

id	cpu_class	memory_class	io_class	network_class
111	medium	small	small	low

Table *cloud_provider*:

organization_id	public	paas	iaas	saas
1	...			

Table *organization*:

Id	name	www	postal_address	email
1	Amazon	www.amazon.com	...	...

3.1.5 Scalability Rule Aspect

DSL Description

MDDB exhibits particular tables that map to exploiting the "Event-Condition-Action" (ECA) method of capturing (quality) property/metric violations and mapping them to specific actions. This ECA method is the one used for representing scalability rules (elasticity rules in the MDDB schema) defined via a specific DSL which is proposed in Deliverable D2.1.2 [D2.1.2] and can be mapped to that of particular rule languages that normally assort a particular rule engine. The rationale for introducing this scalability rule DSL was the following. As most of the DSLs for representing (scalability) rules lack a rich meta-model for describing events, we have created an extended rule DSL encompassing a sub-DSL to cover the missing information concerning the event aspect (based on the work in [Zeginis et al., 2013]). The scalability rules are triggered when an event is emitted by either the Monitoring engines of the Executionware or MDDB and/or KB. Their main target is virtual computing (e.g., CPU cores or memory of a virtual machine) and storage (e.g., the size of virtual storage devices and databases) resources. Four are the main adaptation actions of PaaSage as already defined in [D2.1.1]: scale up (i.e., increase the size of compute core and memory on a virtual machine or the size of a virtual storage device) and scale out (i.e., add more virtual machines or more virtual storage devices), as well as the inverse adaptation actions scale down and scale in, respectively. If more adaptation actions will be realized in the near future in the context of this project, it will be ensured that the proposed DSL can represent all the required information.

The scalability rule DSL has a particular meta-model, part of which is depicted in Figure 14, which comprises various concepts as well as their relationships. The main concept is of course a scalability rule which is related to an event to be detected as well as the set of actions to be executed when the rule is actually triggered. Such events are actually part of a particular event meta-model for Service-based Cloud applications proposed in [Zeginis et al., 2013]. This model is generic enough and extensible to incorporate any other event type defined by domain-specific service providers. Therefore, any Cloud provider could use this model to add any other event type that could be emitted by his/her platform, or even incorporate new non-functional dependencies and cloud-specific layers. Events can be simple or composite, where composite events actually map to logical or timely-related combination of events. Thus, in the meta-model, a composite event is related to two events and indicates a logical combination or a time ordering (where ordering operators are actually a special type of AND-based logical operators) (e.g., sequence in case that one event occurred before the second one) between these two events. Simple events can be classified as functional and non-functional, where the non-functional events are usually attributed to the violation of SLOs while the functional events indicate an erroneous situation concerning a particular component that could exist in the three possible layers (infrastructure, platform and software). Events also map to the layer that they concern, such as infrastructure (IaaS), platform (PaaS) and software (SaaS).

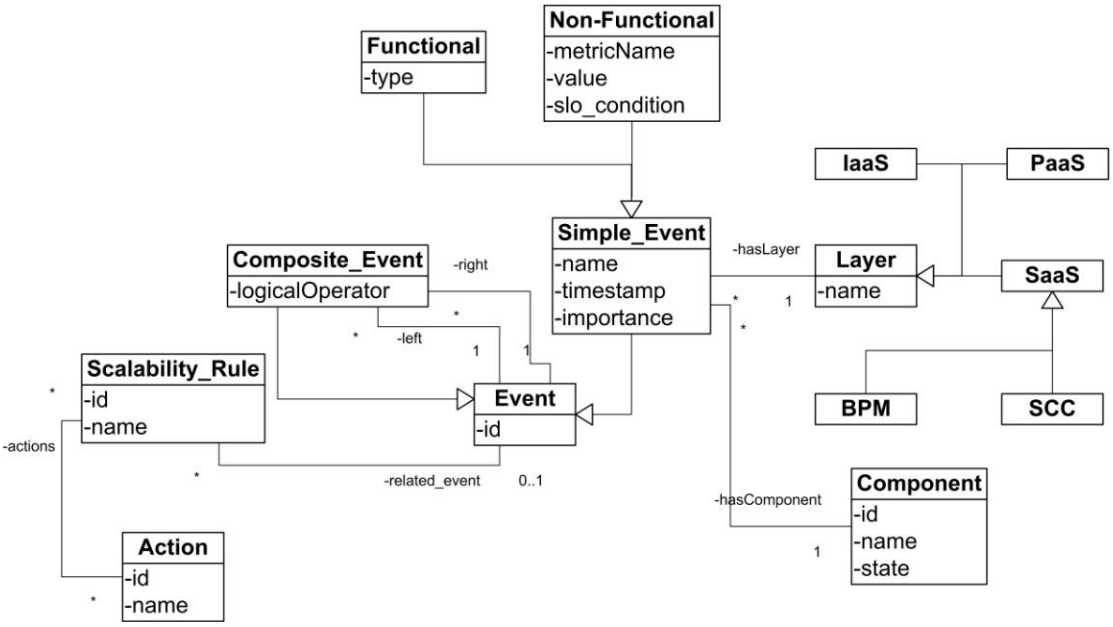


Figure 14 - The meta-model of the scalability rule DSL

Description of Respective MDDB Schema Part

Figure 15 shows the respective MDDB schema part responsible for capturing scalability rules and their provisioning. Scalability rules are represented with the *elasticity_rule table* of the MDDB schema which either maps a single, generic event or an event pattern (i.e., a composite event) to one or more generic (high-level) actions (e.g., *scale-out*). Furthermore, a scalability rule is related to an evaluation frequency, defining how often the specific rule will be evaluated. The event table models generic information about a simple event, thus not concerning any specific event instance, such as the event name and the related *it_slo* so as to map to the metric condition that will be violated. On the other hand, the *event_pattern table* stores the event patterns (composite events) actually occurring during Cloud application execution. Each event pattern maps to either two events or two event patterns and an operator linking them to indicating the timing relationships between these events or event patterns. In this way, we can construct event patterns comprising just two events or even more complex event patterns with many associated events related via specific timing relationships. An elasticity rule is related to the invariant requirements that should hold during its triggering and must be respected, such as scalability policies which restrain the way more resources will be provided or additional VMs could be generated, through the *reqs_to_elasticity_rule* table. In addition, as a scalability rule might trigger more than one action, the *elasticity_rule_action* table associates the scalability rules with these actions. Similarly to events, the actions, as they are at a high specification level, are modelled in a general manner according to the *action* table which stores the id and name of the action.

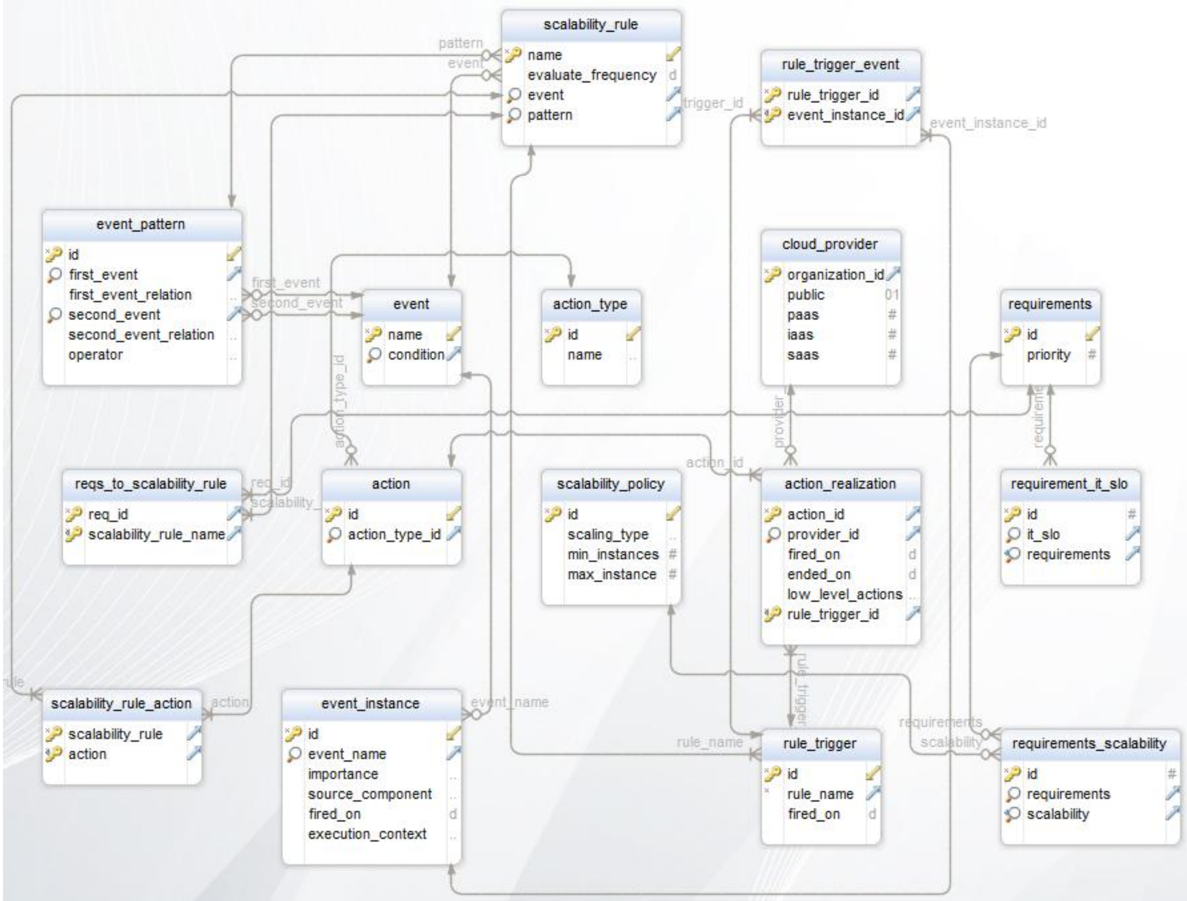


Figure 15 - The respective MDDB part capturing scalability rule information

Apart from being able to capture the specification of scalability rules, the MDDB schema also covers their provisioning by introducing particular tables which map to the triggering of rules by particular event instances and the realization of specific but required (by the rule) high-level actions at the cloud-provider level. The *event instance table* models the information for each unique, concrete event that is raised which includes the generic event of which the concrete event is instance, the source component (e.g., particular VM instance, software component or application) of the event, when the event was fired and under what execution context (see Section 3.1.6). A set of event instances might lead to the triggering of a specific event. To this end, two tables have been created for this purpose: (a) table *rule_trigger* captures generic information about the triggering of the rule, such as which rule and when was triggered, while (b) table *rule_trigger_event* relates a rule trigger with the event instances that triggered it. To also associate the rule trigger with the actual actions that have been executed for it, table *action_realization* models all the necessary information (including the id of the *rule_trigger* as a foreign key), such as which action was executed, when it started running and when it ended its execution, which low level actions were actually used to realize this action and at which cloud provider.

From the above description, it can be understood that the mapping between this DSL and the MDDB schema is straightforward and only some information at the DSL level needs to be splitted across multiple tables in the MDDB schema. A visual representation of part of this mapping (for the DSL meta-model portion shown in Figure 14) can be seen in Figure 16. It must be noted here that as the DSL focuses on covering the specification of scalability rules and not their actual provisioning, the mapping does not include particular, related MDDB tables, such as *rule_trigger* and *action_realization*. It should be decided if the scalability rule provisioning information should also be covered by the DSL proposed.

Running Example

To exemplify the usage of the DSLs for events and scalability rules, we continue with the running example of the SENSAPP, described in deliverable D2.1.1 [D2.1.1] (section 7.3). Assume that the application provider and/or external experts extract the following ECA-like scalability rule:

- *Scale out when the average application response time goes beyond 10sec or its availability falls below 99,9%.*

This rule consists of two conditions on two metrics values, respectively, but the violation of just one of them is adequate to fire the rule and scale up the VM hosting the application. Scaling up means moving to a more powerful VM, with respect to all resources (i.e. CPU, memory, disk), thus addressing both violations. Suppose, now, that two events occur (with ids 2345 and 3456) which lead to the triggering of the rule and to the execution of the respective action (with id 4567) which maps to increasing the CPU, memory and disk capacity (low-level actions) and lasted one and a half minute (between 10/01/2014:20:43:00:234 and 10/01/2014:20:44:36:234).

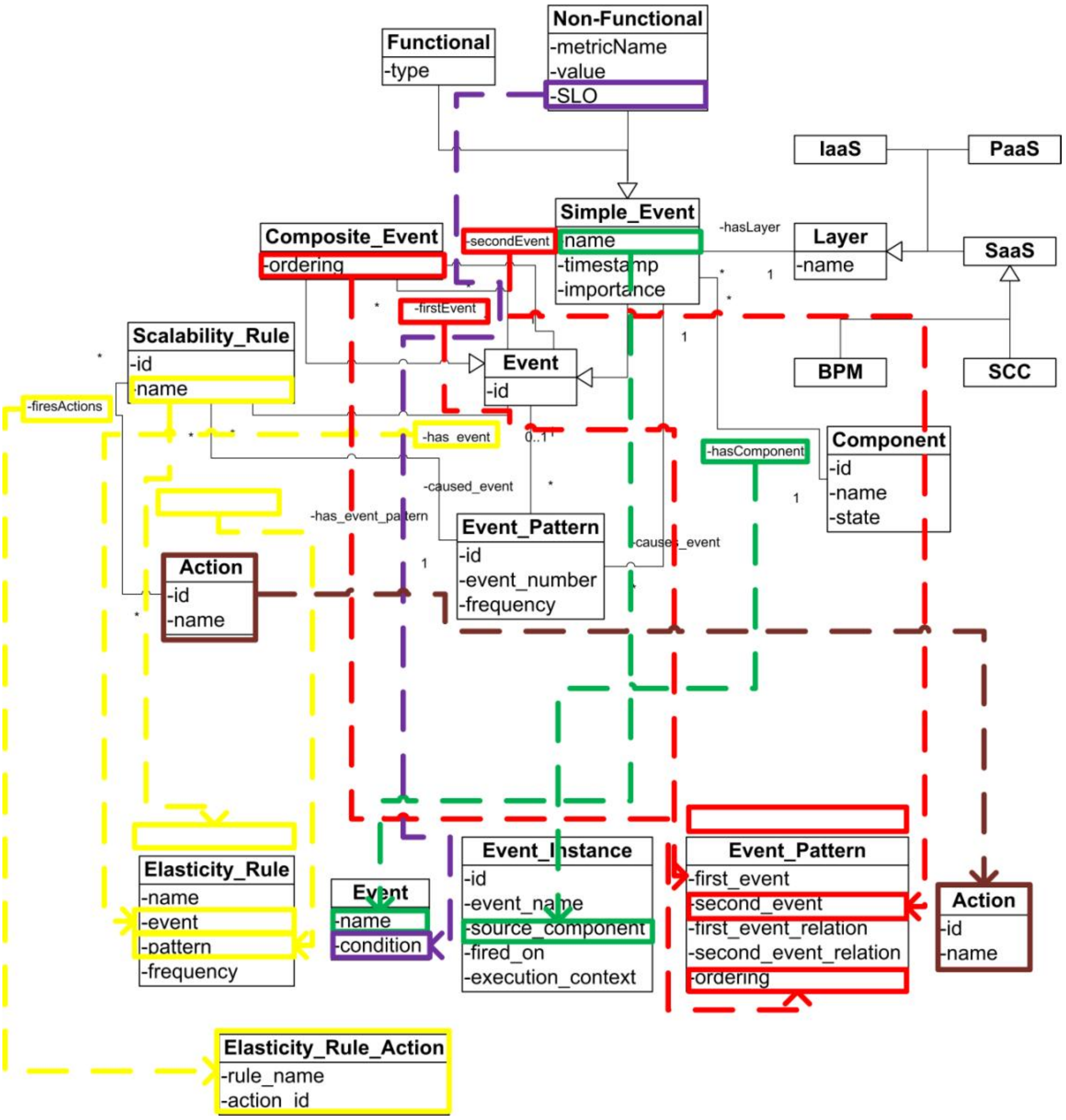


Figure 16 - The mapping from the scalability rule DSL to the respective MDDB schema part

To show how the mapping is realized according to the running example, we first show what would be the content of the scalability rules according to the scalability rule DSL of PaaSage, and then we visualize the rows that will be inserted in all the rule-related tables of the MDDB schema according the mapping indicated. Rows in blue colour indicate content that is generated based on the mapping while rows in grey colour indicate existing content mapping to already defined information supporting the description of the scalability rule and its related event information. It should be noted that as the rule provisioning information is not covered by the DSL, we only show what would be the respective content of the MDDB which could be created by the ExecutionWare module of PaaSage when the respective scalability rule is fired.

The Scalability rule DSL specification expressing the running example rule would be the following:

```
<ScalabilityRule name=" Scale_out1">
        <related_event>
        <event name="Composite Event" logicalOperator="OR">
                <left>
                        <event name="response_time_violation">
                                <slo_condition operator="GREATER_THAN"
value="30">
                                        <metric id="1" name="Response
time/>

                                </slo_condition>
                        </event>
                </left>
                <right>
                        <event name="availability_violation>
                                <slo_condition operator="LESS_THAN"
value="99.9">
                                        <metric id="2"
name="Availability/>

                                </slo_condition>
                        </event>
                </right>
        </event>
        </related_event>
        <actions>
                <action id="1" name="Scale-up" type="SCALE_UP">
                </action>
        </actions>
</ScalabilityRule>
```

The respective mapped content in the MDDB according to the selected colour convention is shown visually in the tables below.

Table *scalability_rule*:

name	evaluate_frequency	event	pattern
Scale_out1	Every 5 minutes	null	12345

Table *event_pattern*:

id	first_event	first_event_relation	second_event
12345	SenseApp_respTime_violation		SenseApp_availability_violation

id	second_event_relation	operator
12345		OR

Table *event*:

name	condition
SenseApp_respTime_violation	123
SenseApp_availability_violation	234

Table *it_slo*:

id	metric	threshold	platform
123	Response time	30	
234	Availability	99,9	

Table *action*:

id	action_type_id
1	1

Table *action_type*:

id	Name
1	Scale_out

Table *scalability_rule_action*:

scalability_rule	Action
Scale_out1	1

Table *reqs_to_scalability_rule*:

req_id	scalability_rule_name
1	Scale_out1

Table *rule_trigger*:

id	rule_name	fired_on
111	Scale_out1	10/01/2014:20:45:36:234

Table *event_instance*:

id	event_name	importance	source_component	fired_on	execution_context
2345	SenseApp_respTime_violation	Critical	SENSAPP	10/01/2014:20:42:36:234	Exec_context1
3456	SenseApp_availability_violation	Critical	SENSAPP	10/01/2014:20:42:37:234	Exec_context1

Table *action_realization*:

action_Id	provider_id	low_level_actions	fired_on
1	1	Increase CPU Increase Memory Increase Disk capacity	10/01/2014:20:43:00:234

ended_on	rule_trigger_id
10/01/2014:20:44:36:234	111

Table *rule_trigger_event*:

rule_trigger_id	event_instance_id
111	2345
111	3456

Table *execution_context*:

id	app_id	start_time	end_time	info	total_cost
Exec_context1	2	10/01/2014:20:40:36:234	10/01/2014:20:50:36:234		

Table *cloud_provider*:

organization_id	public	paas	iaas	saas
1	true			

Table *organization*:

id	name	www	postal_address	email
1	Amazon	www.amazon.com	...	...

Table *application*:

id	name	Version	user	resource
2	SensApp	1.0	4	1

3.1.6 User Requirements and Monitoring Aspect - WS-Agreement

An SLA seems to be one of the most prominent ways of expressing an agreement between two or more parties about a particular service that is offered and its respective service levels that it needs to sustain. Such an agreement is usually a product of negotiation between these parties, although there are cases where an SLA is offered as it is to potential service requesters. SLAs have proven to be one of the main means of steering the service provisioning and adaptation phase as they usually specify particular quality of service guarantees, commonly named as Service Level Objectives (SLOs), which need to be sustained during service execution. Thus, violation to one or more of these SLOs can lead to particular adaptation actions which aim at raising up again the appropriate service level offered to the service requester.

To this end, some SLA languages pave the way for expressing particular actions that need to be taken when specific SLOs are breached.

In the context of the PaaSage project, SLAs are envisioned to play a crucial role in the deployment and provisioning of cloud-based applications as they can determine what cloud services must be offered to a particular application and under what guarantees. It is expected that such an SLA is being produced after a particular deployment plan has been determined as only at this moment it is known which particular cloud services will be required for the particular application at hand. There is no actual requirement of how the SLA is really produced. The common practice in the current cloud world is that cloud providers offer particular SLA templates in a take-it-or-leave-it manner. However, this situation can change in the near future after the cloud providers realize the real benefits of providing more flexible SLA templates leaving place for variability and customization leading to cost savings, additional gains and more satisfied customer. Thus, based on the above analysis, a possibly composite or set of SLAs (each pertaining to a particular cloud service) could be produced through a broker which functions on behalf of the customer in order to negotiate and agree with all the cloud providers involved. Alternatively, the service requester might follow a manual approach where he/she contacts with each of the cloud provider in order to arrange for the production of the appropriate SLA(s).

DSL Description

Various SLA languages have been proposed by researchers or even by standardization organizations. For a detailed analysis over their capabilities concerning the support of the service management life-cycle, the survey in [Kritikos et al., 2013] can be studied. WS-Agreement, proposed by Open Grid Forum (OGF), is one of the most widely-used SLA language which has been successfully applied in the context of Grid services and is thus quite suitable for being used in the context of Cloud services. This language is XML-based and is supported by particular implementation frameworks, such as WSAG4J. It has been also heavily used in the context of other European research projects.

WS-Agreement has the main strength that it is a standard and that it captures well basic SLA information, such as what are the (signatory) parties involved, which are the services offered and what are their main quality of service (QoS) guarantees. It also allows the logical combination of QoS guarantees in such way that different service levels can be guaranteed under different qualifying conditions, although the service level notion is not formally defined. Another strong feature of the language is the mapping of QoS guarantees to rewards and penalties which enable to create a trust level between the parties involved in the SLA formation such that service requesters are compensated when SLOs are violated and service providers are rewarded when a higher service level than the one promised was actually delivered.

On the other hand, we can distinguish two main shortcomings of WS-Agreement. First, this language does not define particular metrics that have to be evaluated in order to assess when SLOs are violated. Instead, the WS-Agreement developers indicate that the definition of these metrics is out of scope of this language and should be performed through domain-specific service description languages. This certainly creates the need of introducing or exploiting an existing ontology of the various metrics that will be exploited in PaaSage for cloud service monitoring and adaptation which also enables the formal definition of all their possible aspects, such as the metrics' assessment formula, value type, and frequency/timing as well as unit of measurement. An example of such an ontology language is OWL-Q [Kritikos and

Plexousakis, 2007], which has various interesting capabilities and merits, as indicated in [Kritikos et al., 2013]. Second, apart from defining penalties for SLO violations, the language does not map these violations to particular adaptation actions that resolve them in order to sustain an acceptable service level (that is why a scalability rule DSL is required). This means that either the language is used in the context of PaaSage at a high-level to express QoS requirements for cloud services or the language must be extended to allow this required connection. The consortium for this second problem has taken the path of introducing a separate (scalability rule) DSL for defining exactly those rules that identify which adaptation actions will be performed when one or more SLO objectives are violated. In this way, the only concern with respect to the modelling is the correct linking of these two DSLs according to similar or equivalent concepts that might use which can be appropriately handled by considering that the scalability rule DSL is currently proposed and designed by the consortium and thus this linkage and integration can be more carefully performed through the appropriate realization of this language.

A detailed class diagram showing the conceptual elements of WS-Agreement is depicted in Figure 17. As it can be seen, the main conceptual element is the Agreement, representing a specific SLA agreement between one or more parties. This element is then composed of a context element and a set of service and guarantee terms. The context is related to parties involved in the agreement, the duration of the agreement as well as optionally to the name of the SLA template on which the agreement relied to be built.

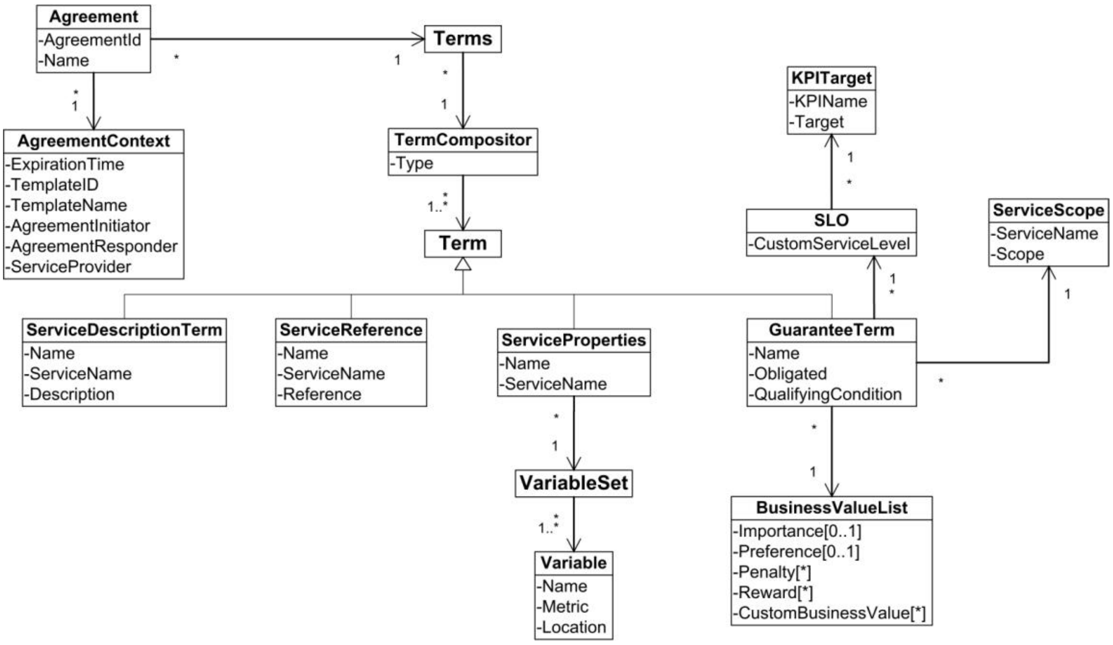


Figure 17 - The class diagram of WS-Agreement

Service terms relate to a particular service description which is either defined inline in the agreement document or referenced. This service description is also related to a set of service properties which can be used in the building of guarantee terms. Each service property has a name, a URI pointing to its definition in an external domain-specific language as well as a structural reference to a location in another (service property definition) document, such as an XPATH expression. It should be mentioned here that many service terms can be defined for the same service. This allows to express two alternative specification cases: (a) service terms refer to the description of the components of a (composite) service and (b) service terms refer to different facets of a service description, such as the service's interface in WSDL.

Guarantee terms, as already indicated, map to SLOs. A guarantee term first references the party that is obliged to satisfy it which can be either the service provider or even the service requester (e.g., when particular guarantees must be offered by this party in order to enable the service provider to sustain a particular service level). It also defines what is the service scope, i.e., the particular service or service part that it concerns. SLOs are defined by indicating the Key Performance Indicator (KPI) name (i.e., referencing a particular metric) and the respective target (metric threshold value) and can be accompanied by qualifying conditions (such as that the invocation rate for the service is below a particular threshold) which must be met in order to assure the satisfaction of the SLO. Finally, guarantee terms are mapped to business-level information, such as rewards and penalties, guarantee importance (indicating the relative importance of guarantee terms) and preferences (which are fine-grained business values for different alternatives that can be offered).

SLA-Based MDDB Schema Description and Mapping

PaaSage has certainly realized that it is important to store and exploit information that originates from SLAs as such information will guide the cloud-based application deployment and execution phases. To this end, it has accommodated for particular tables in the MDDB schema which are used for the storage of SLA information. These tables are *sla* and *it_slo*. The first table is responsible of storing general information about an SLA, such as the targeted cost, location and revenue. This information cannot be mapped into a straightforward manner in WS-Agreement. For instance, the targeted cost could be part of an external service description/offering that is referenced by the WS-Agreement document. In addition, it also references one or more rows of the *it_slo* table which represent information about a specific IT-based SLO, such as the metric involved, its required threshold and the platform as a service involved (i.e., to which it actually applies). However, in contrast to WS-Agreement, we can see here two main differences: (a) the expressivity of WS-Agreement in terms of SLO combinations is missing from the MDDB Schema and (b) the MDDB Schema takes a different approach in metric definition where important aspects of a particular metric are actually modelled in the *metric* table such as the metric's name, textual description, layer, unit and value direction. The latter information could, of course, be drawn from metric specifications defined via languages such as OWL-Q [Kritikos and Plexousakis, 2006]. Apart from the two main differences, it is apparent that some other information can also be mapped, especially the one concerning the description of the service offered which can be drawn from the *platform_as_service* and *cd_vm_type* tables as well as the one concerning the description of the service providers and requesters which can be drawn from tables *cloud_provider*, *organization* and *user* (depending also on the type of party involved in the agreement). It must be mentioned, though, that other important information, such as rewards, penalties and SLA duration (or expiration date) is currently missing from the MDDB Schema and this information will certainly be added in the forthcoming versions. Figure 18 shows the current mapping that can be enforced between the two current versions of the compared schemata.

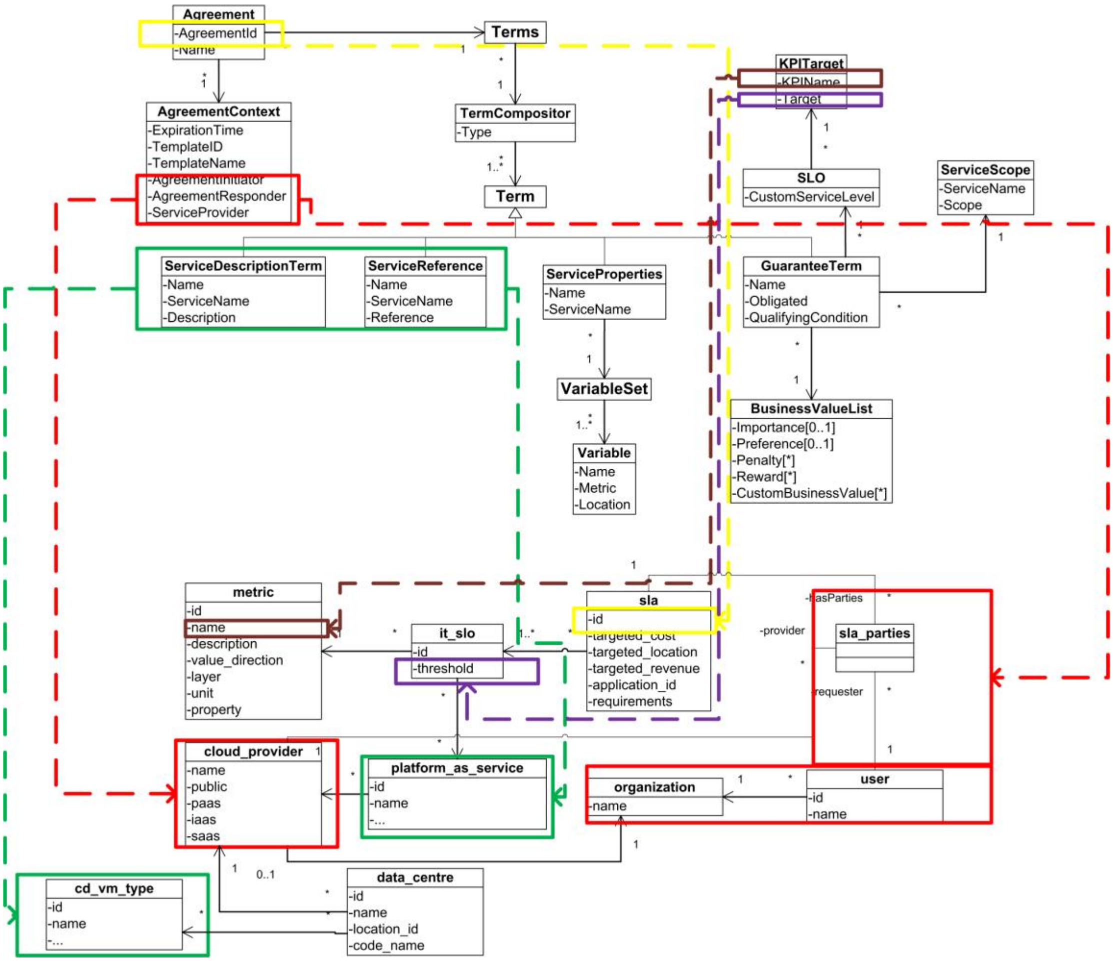


Figure 18 - The mapping between WS-Agreement and MDDB at the schema level

Apart from being able to define requirements, it is also important to capture monitoring information as this is crucial for various purposes: (a) enhance application profiles, (b) associate applications with best deployments such that this knowledge could be exploited in the Social Network by users that desire the run the same or equivalent applications, (c) avoid bad application deployments, and (d) enable the assessment of IT SLOs and the triggering of adaptation actions when these SLOs are violated. By also linking deployment and execution information with user requirements, we also achieve traceability such that we are able to not only understand which requirements drove the respective deployments but also which deployment and execution information lead to the violation of these requirements or their updating.

To this end, MDDB was designed to capture all of the required monitoring and execution information for applications through the following tables:

- *execution_context*: it encapsulates generic information that is related to a particular application execution, such as when application execution started and ended, what was the total cost incurred and for which application. This context is also used as a reference to connect to all other more detailed information about a particular application execution.

- *deployment_execution*: it captures which requirements lead to a particular deployment under a specific execution context and for which user. This is where traceability is actually achieved.
- *appl_monitor*: This table is related to the monitor created for observing the performance of an application according to a specific metric. It covers information such as the considered application, the relevant *it_slo* and *execution_context*, the *metric* used for the monitoring, the *value* actually monitored, when this value was reported and a pointer to raw measurement data (stored in the TSDB, see Deliverable D5.1.1 [D5.1.1]) used to generate the monitored value. Similarly, monitors are created for other entities and the following tables are used to capture the respective monitoring information:
    - *software_component_monitor*: it monitors the performance of a particular software component instance according to a specific SLO.
    - *resource_monitor*: it monitors the usage (according to a specific metric for a particular SLO) of a resource (see *resource_class* column) of a specific VM instance.
    - resource_coupling_monitor: it monitors the performance (e.g., communication bandwidth) with respect to a coupling of resources (actually VM instances)
- *slo_assessment*: this table captures the assessment of a particular SLO under a specific (application) execution context.

An overall visualization of the MDDB schema part dedicated to the capturing of requirements and monitoring/execution information is depicted in Figure 19.

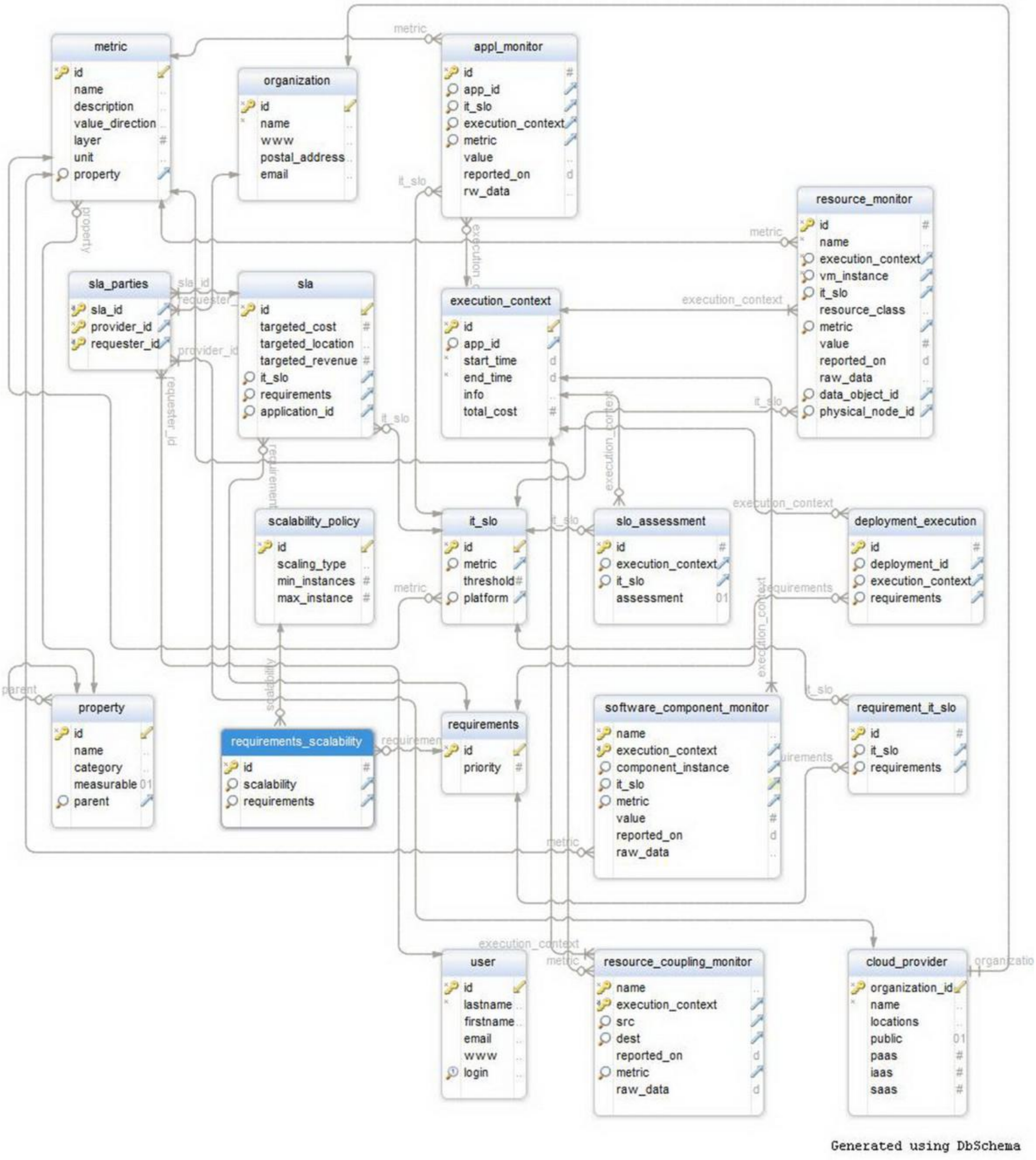


Figure 19 - MDDB schema part capturing requirements and monitoring/assessment information

Running Example

For better comprehension of how the compared schemata are mapped, let us rely on the running example of SENSAPP. Suppose that a WS-Agreement document has been developed which concerns the provisioning of VM "SL" from Cloud Provider "Amazon" for Cloud Customer "CC1" (the user of SENSAPP) under the guarantee that the VM's CPU usage will not go below a particular threshold (60%). The WS-Agreement specification for this example is given below.

```
<wsag:Agreement AgreementId=”1”>
  <wsag:Name>
      VM Agreement for Application SENSAPP
```

```
   </wsag:Name>
   <wsag:AgreementContext>
    <wsag:AgreementInitiator>
      CC1
    </wsag:AgreementInitiator>
    <wsag:AgreementResponder>
      Amazon
    </wsag:AgreementResponder>
    <wsag:ServiceProvider>
      Amazon
    </wsag:ServiceProvider>
   </wsag:AgreementContext>
   <wsag:Terms>
      <wsag:All>
        <wsag:ServiceDescriptionTerm
   wsag:Name=”ServiceTerm1” wsag:ServiceName=”SL”/>
        <wsag:GuaranteeTerm Name=”GuaranteeTerm1”
Obligated=”ServiceProvider”>
         <wsag:ServiceLevelObjective>
             <wsag:KPITarget>
               <wsag:KPIName>cpu_usage</wsag:KPIName>
               <wsag:Target>60</wsag:Target>
             </wsag:KPITarget>
         </wsag:ServiceLevelObjective>
        <wsag:BusinessValueList>
             <wsag:Penalty>10$</wsag:Penalty>
        </wsag:BusinessValueList>
       </wsag:GuaranteeTerm>
      </wsag:All>
   </wsag:Terms>
</wsag:Agreement>
```

The respective content of MDDB that will be produced from the transformation of the above document is the following (expressed in rows of the mapped tables). It must be highlighted that table *metric* might be updated if the agreement terms in the WS-Agreement document refer to metrics which are defined elsewhere (e.g., wrt. the metric ontology used by PaaSage), provided that the metric definition can be mapped to the content of the *metric* table.

Table *sla*:

id	targeted_cost	targeted_location	targeted_revenue	it_slo	application_id
1	…			3	2

Table *it_slo*:

id	metric	threshold	platform
3	4	60	

Table *sla_parties*:

sla_id	provider_id	requester_id
1	1	4

Table *metric*:

id	name	description	value_direction	layer	unit	property
4	cpu_usage	The average usage of CPU for the VM	positive	Infrastructure	percentage	

Table *cloud_provider*:

organization_id	public	paas	iaas	saas
1	true			

Table *organization*:

id	name	www	postal_address	email
1	Amazon	www.amazon.com	...	...

Table *application*:

id	name	version	user	resource
2	SensApp	1.0	4	1

Table *user*:

id	lastname	Firstname	email	www	login
4	User4	CC1	CC1.User4@email.com	...	...

Table *cd_vm_type*:

id	name	datacenter	ci_vm_type	classified_on	evaluated_on
11	SL	1	111	...	...

Table *cd_vm_type (cont.)*:

id	**cpu**	**ram**	**disk**	**benchmark_rate**	**cost_per_hour**	**resource**
11	2	2	100	...	0.5	7

Table *data_center*:

id	**name**	**location_id**	**cloud_provider**	**code_name**
1	DC Ireland	1111	1	DCI

Table *ci_vm_type*:

id	**cpu_class**	**memory_class**	**io_class**	**network_class**
111	medium	small	small	low

3.1.7 Security Requirements, Capabilities and Policies Aspect

Security seems one of the most major obstacles for the adoption of Cloud computing as most of the businesses and organizations are concerned with the ability of cloud providers to secure the access to their data and services, especially if we consider the multi-tenant context under which cloud platforms are operating. To this end, various cloud providers are undergoing a certification process which guarantees that they realize and have in place various security controls which mitigate the risk of reaching security breaches in their cloud platforms. These security controls are specified at a high-level of abstraction and realized by adopting various security mechanisms.

For a cloud customer, it is essential that he/she provides his/her requirements through indicating the necessity of the existence of particular security controls realized in a cloud provider's platform. In this way, he/she will be guaranteed that no matter how a particular security control is realized, particular security problems related to this security control will not occur. Moreover, in order to obtain additional guarantees and further enhance the trust level between the cloud customer and provider, security controls can be linked to particular security properties whose service level can be measured. In this way, the cloud provider and customer can come into an agreement over which actual service levels will be provided for these security properties and such an agreement could be expressed through an existing SLA language, such as WS-Agreement.

Based on the above analysis, it is apparent that there should be a DSL which is able to express security requirements and capabilities in terms of high-level security controls. This DSL should also be able to link these security controls to particular security properties, some of which will need to be measured through specific security metrics. This DSL could be part of a bigger DSL, as already mentioned, which could be used to express security level agreement templates (expressing security needs and requirements supplied by cloud providers and customers) and actual security level agreements, such as a DSL dedicated to the specification of (generic and not only security-based) service level agreements. Alternatively, the DSL could also be standalone which would indicate two main issues: (a) cloud service matching should be accompanied by cloud security matching and (b) appropriate descriptions are produced and connected to those of the more generic cloud service specifications. No matter the way this DSL is actually developed, it is essential that a particular security control ontology is also needed to enable a proper and more accurate matching of cloud security requirements and capabilities. Such an ontology could rely on particular products, such as the Cloud Control Matrix (CCM) (https://cloudsecurityalliance.org/research/ccm/) proposed by Cloud Security Alliance.

Apart from the need for a security control DSL, which would provide added-value to the cloud application deployment process and thus to the PaaSage project, it is also essential that a security level is imposed on the MDDB prototype so that the access to its resources is restricted only to those entities that have the appropriate rights. Such a security level can be realized through adopting particular authentication and authorization mechanisms, such as single sign-on (SSO) and role-based access control (RBAC) (as has been indicated in the description of the MDDB architecture), and exploiting particular DSL languages for the proper description of the authentication and authorization information in terms of resources, roles and policies imposing the rights that roles will have on the MDDB resources. After thorough examination of all possible alternatives, the DSL languages that have been selected are SAML and XACML, where the first will be responsible for the description of authentication information while the second is responsible for the description of security policies. The selection of these languages relied on the fact that these languages are XML-based, they already have particular realizing frameworks and are powerful enough for the description of the appropriate security artifacts. More importantly, they are open standards proposed by standardization organizations, such as OASIS. Finally, another criterion that lead to their selection was that there is close cooperation of these two standards where a specific profile (SAML profile for XACML) has been developed which is used for exploiting SAML 2.0 for the protection, storage and transport of XACML schema instances as well as other information required by a XACML implementation.

Some of the above security DSLs will need to be mapped to the MDDB schema but for obvious security reasons, it was decided that quite sensitive information will be stored outside of MDDB (actually it could be realized as a dedicated MDDB instance that is isolated and only available to the security mechanisms that will be realized for MDDB) in order to not enable MDDB intruders to have access to quite sensitive information. The DSLs that will indeed be mapped to MDDB are the security language and XACML. To this end, the following sub-sections shortly analyze these DSLs and indicate the way they will be mapped to MDDB.

3.1.7.1 Security DSL

DSL Description

As the agreed security levels for the properties related to a specific security control can be expressed by WS-Agreement, which is a language already selected by PaaSage to express agreements between cloud providers and customers, we have decided to develop a DSL which will be able to not only express what security controls are required and offered by a cloud customer and provider, respectively, but also to link these security controls to security properties which can be monitored and measured. The main benefits and usages of the DSL will be the following:

- The DSL will enable the matching of cloud providers based on the cloud customer requirements, thus paving the way for negotiation
- The DSL will set up the context of the negotiation in terms of the security levels that will need to be sustained for each security control required
- It will enable the monitoring of the security capabilities of the cloud providers in order to assess whether the promised security levels have been violated and impose particular penalties.

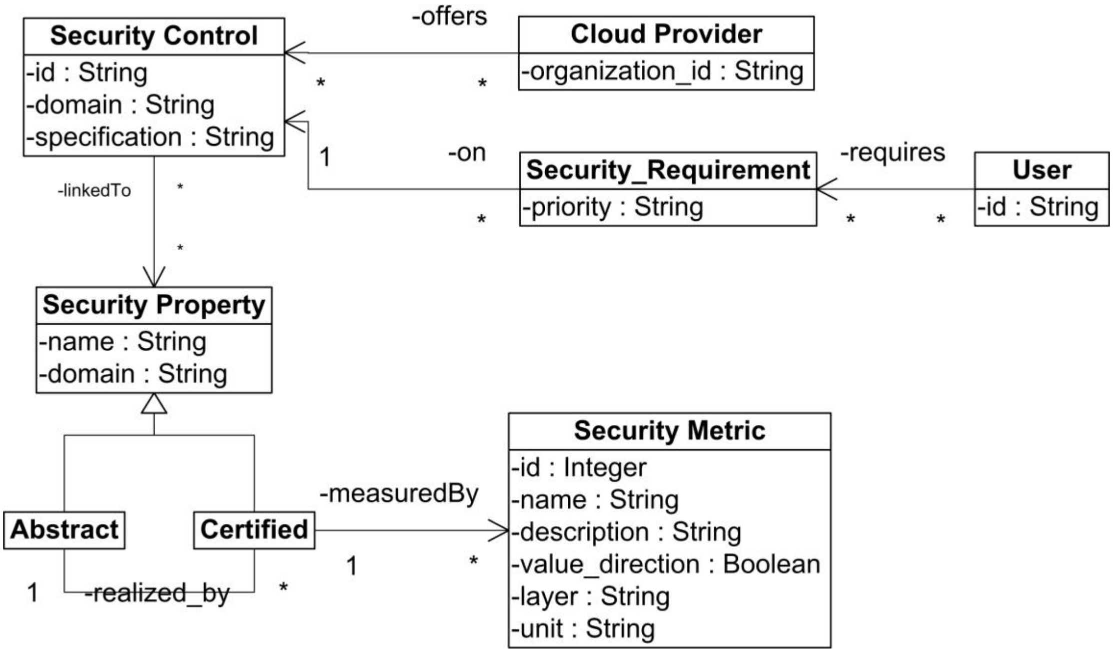


Figure 20 - The class diagram of the security DSL

While the specification of security controls can rely on CCM and it is quite easy to model it, the design focus of the required DSL language shifted to the specification of security properties that map to these security controls. Among the various candidate security property DSLs, it was decided to use the one developed through the research work conducted in the CUMULUS European project [Cumulus] with particular adjustments needed to be more uniform and consistent across all DSLs. This DSL relies on a particular security property domain vocabulary which is able to describe in a precise way those security properties that can be utilized in the specification of security capabilities and requirements.

Based on the main entities that constitute the above security property domain vocabulary and the information required for modelling security controls, we have developed a DSL whose meta-model is depicted in Figure 20. As it can be seen, the main entity is security control which maps to a particular security domain, it has a particular id derived from this security domain, and is textually described by the specification attribute. A security control is required by a particular cloud user with a specific priority. The introduction of priorities enables better matching the security capabilities (e.g., through considering hard and soft requirements) as well as their ranking. Security controls are also implemented and offered by cloud providers. Security controls are finally linked to security properties. The latter are modelled through the *Security Property* class which has two main sub-classes: *Abstract* and *Certifiable* representing high-level security properties which concern desirable security features of the CP and map to a set of concrete security properties realizing them and highlighting a particular aspect over them. The latter represent concrete security properties which are measurable. Abstract security properties are classified under a specific category of the Cloud Control Matrix (CCM) which actually represents a particular security domain, satisfying in this way the goal principle that security properties must not be defined through mechanisms but goals. The certifiable security properties comprise a minimum set of attributes that is enough for distinguishing between different measurements of the same property on the same or different systems, thus satisfying the equivalence principle. This is the point we differentiate based on this DSL. In our opinion, the set of security attributes for a particular property, in the way they are defined and according to the equivalence principle, map to the definition of a security metric. In this way, we become uniform as we are able to have IT SLOs which comprise various types of metrics, such as the ones measuring performance and security properties.

Description of Respective MDDB Schema Part

Initial versions of the MDDB Schema were not able to capture well the definition of security properties so the current version has extended them with this capability. Figure 21 shows the relevant MDDB schema part. The actual extensions realized cover two main aspects: (a) security control information captured by the *security_control* table (b) security requirements expressed through *requirement_sec_control* table which maps to a requirement belonging to an SLA, the security control that is required and the respective preference/priority of the user, (c) the *provides_sec_control* table which maps cloud providers to the security controls that they offer, (d) the association of security IT SLOs to security control requirements through the *sec_control_slo* table, and (e) the definition of security properties. Please note here that security IT SLOs are already captured by the *it_slo* table where the metrics involved measure particular certified/measurable properties (where these properties can be indirectly discovered via accessing the *metric* table).

As the mapping of an SLA language is analyzed in a different deliverable section (3.1.6), the focus of the analysis is now on the definition and mapping of security property models. In MDDB, security properties are specified through the *prop* and *sec_prop* table where the first table captures generic information about any type of property, such as the property's id, name, category, parent (abstract property) and whether it is measurable (obviously mapping to certifiable for security properties), while the second table additionally models the property's CCM domain. As indicated above, the *metric* table is enough for describing the way certified security properties can be measured while it also links the metrics to the attributes that they measure, so no further additions or extensions are needed.

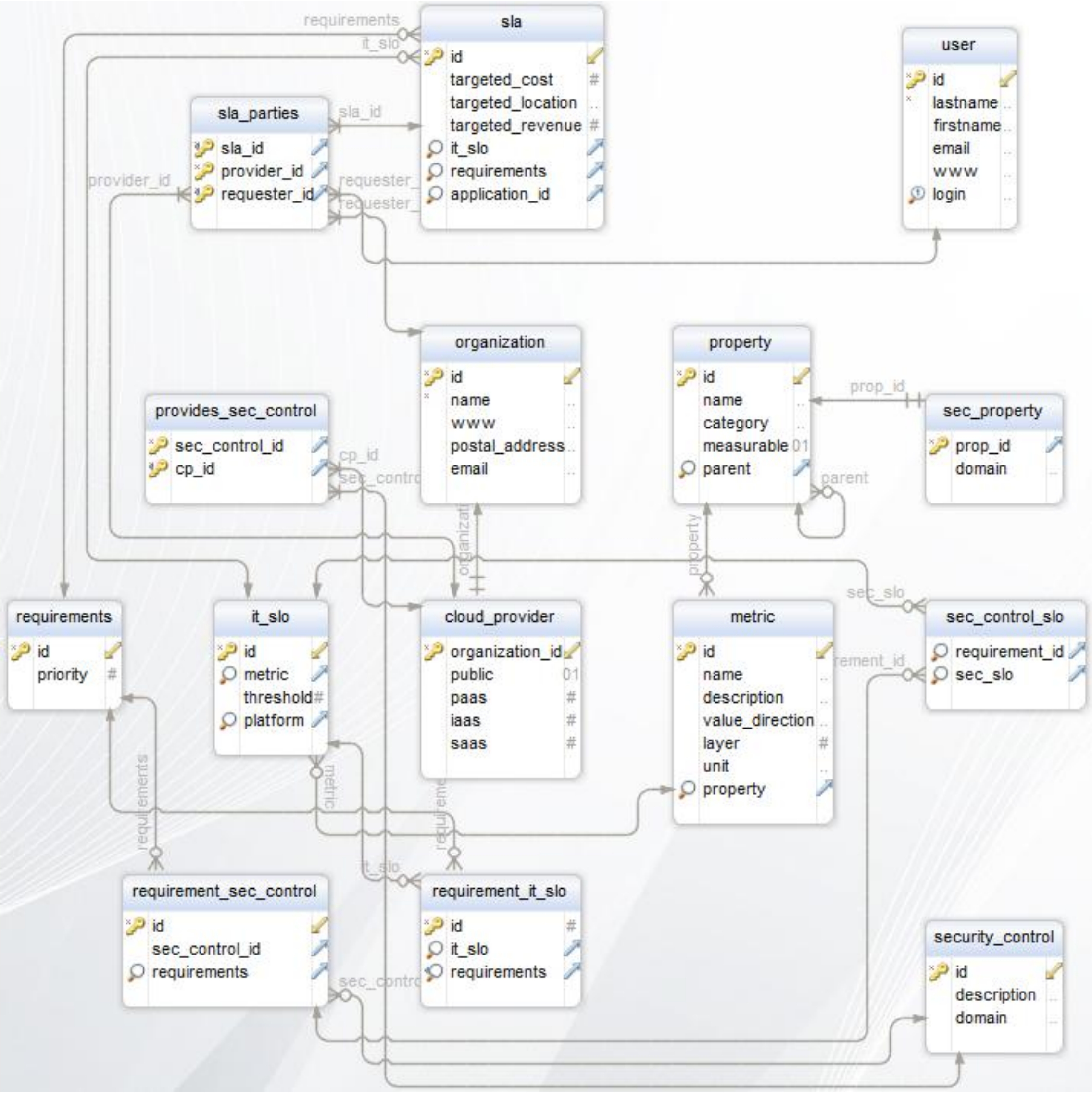


Figure 21 - The MDDB schema part capturing security requirement and capability information

Based on the above analysis, the (more or less straightforward) mapping at the schema level between the proposed security DSL and the (extended) MDDB schema can be seen in Figure 22.

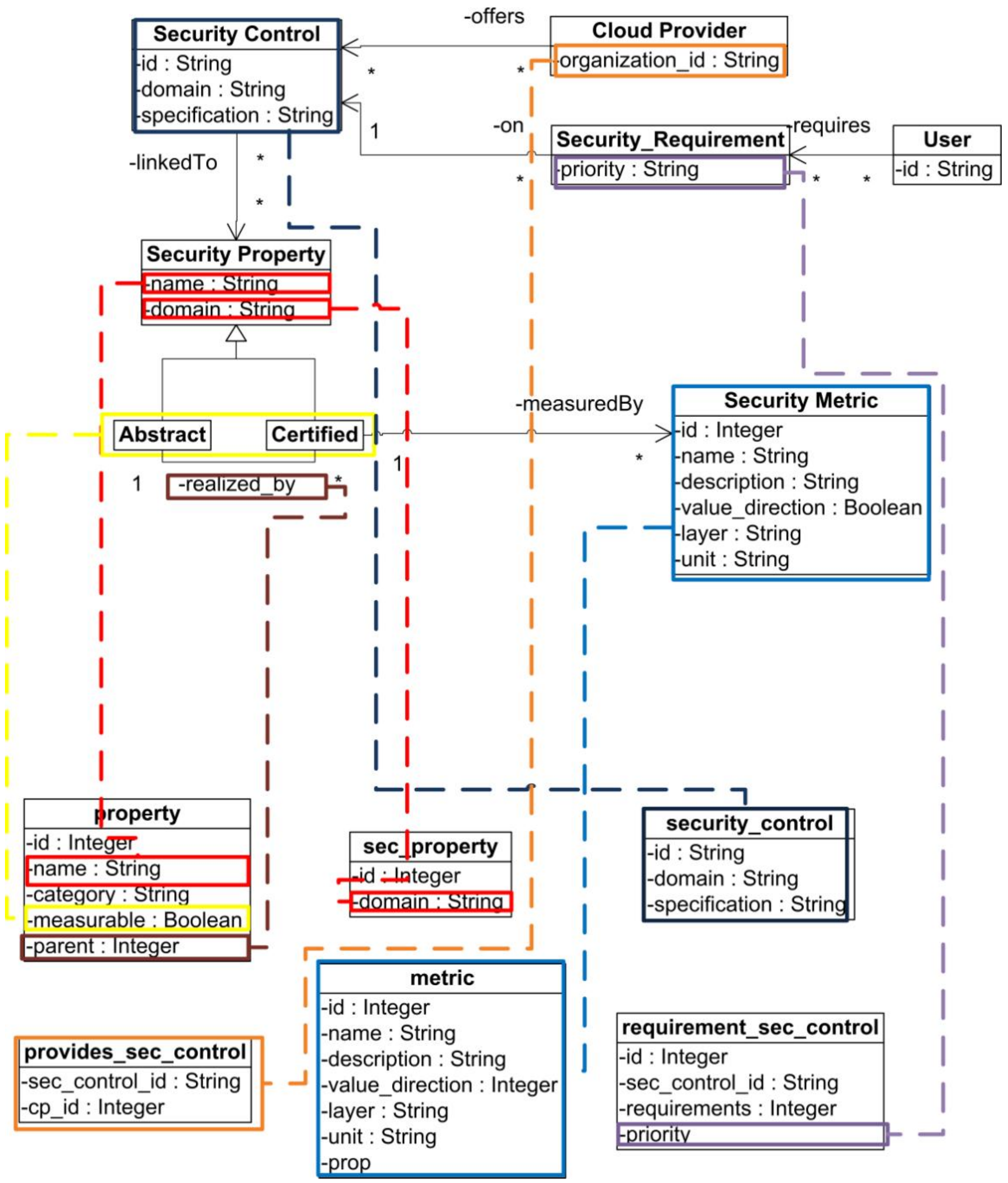


Figure 22 - The mapping of security DSL to MDDB schema

Running Example

Based on the SENSAPP running example, suppose that the cloud application user (named as *CC1* with ID 4) requires the security control *SEF-05* belonging to the control domain: *Security Incident Management, E-Discovery & Cloud Forensics (SEF) - Incident Response Metrics*. Further, suppose that this user has made an agreement with the Amazon cloud provider (with organization_id 1) which dictates a particular SLO for property *incident_management_quality* indicating that the metric *percentage_of_timely_incident_reports* measuring this property has a threshold of 99%. Then, the respective MDDB content that will be produced will be the following (here all tables are in blue mapping to new MDDB content as we assume that the security SLO might refer to a property that has not been inserted in the MDDB but is described by the security language of CUMULUS):

Table *security_control*:

id	description	domain
SEF-05	Mechanisms shall be put in place to monitor and quantify the types, volumes, and costs of information security incidents.	*Security Incident Management, E-Discovery & Cloud Forensics (SEF) - Incident Response Metrics*

Table *requirement_sec_control*:

id	sec_control_id	requirements
1	SEF-05	5

Table *sec_control_slo*:

requirement_id	sec_slo
1	2

Table *requirements*:

id	priority
5	"high"

Table *provides_sec_control*:

sec_control_id	cp_id
SEF-05	1

Table *itslo*:

id	metric	threshold	platform
2	1	99	

Table *metric*:

id	name	description	value_direction	layer	unit	property
1	Percentage of timely incident reports		1		percentage	1

Table *property*:

id	name	category	measurable	parent
1	incident_management_quality		true	

Table *sec_property*:

prop_id	domain
1	SEF

Table *cloud_provider*:

organization_id	public	Paas	iaas	saas
1	true			

Table *organization*:

id	name	www	postal_address	email
1	Amazon	www.amazon.com	...	...

Table *user*:

id	lastname	firstname	email	www	login
4	User4	CC1	CC1.User4@email.com	...	...

3.1.7.2 XACML

DSL Description

XACML (eXtensible Access Control Markup Language) is a declarative access control policy language proposed by OASIS as a standard. This standard also indicates a processing model specifying the way access requests can be evaluated based on the rules defined in the available policies. XACML is used to realize Attribute Based Access Control (ABAC) systems where attributes are provided for resources, actions and users and policies are described through patterns over them so as to reach decisions of whether a particular user can perform a specific action on a certain resource. As XACML supports ABACs, it can be easily used also to support the realization of Role-Based Access Control (RBAC) systems which is the one of the main security requirements for the MDDB prototype.

The UML diagram showing the main entities and their relationships of XACML is depicted in Figure 23. As it can be seen, there are three main elements in XACML: policy sets, policies and rules which are actually the three levels of security policy description that can be specified in XACML over subjects, resources and actions. A PolicySet is actually what its name signifies, a set of policies which can be jointly used for reaching a particular decision through exploiting policy combining algorithms. This set can also contain other policy sets and is obviously considered as the standard medium for combining policies into a single one. A Policy signifies a single access control policy which comprises a set of rules as well as a rule combining algorithm which is used for combining the results of their evaluation. This policy is considered as the basis for an authorization decision. Finally, rules constitute the most fine-grained elements in access control policy description which cannot be used in an isolated manner. In fact, rules can only be exchanged if they are packaged inside policies. A rule comprises the following information:

- the *target* which indicates the set of requests on which the rule can be applied by defining conditions over the requests attributes. The latter attributes can map to the description of subjects, resources and actions. In fact, subjects can have various attributes, while resources represent data, services or system compositions with one attribute. Figure 1Figure 23 shows the mechanism through which the set of requests are actually identified. In particular, a target comprises 0 or more any-of elements which represent a disjunctive list of all-of elements. The latter elements represent a conjunctive list of Match elements which are those elements which indicate a set of entities by matching attribute values in requests with the attribute value in them.
- the *effect* indicating the outcome of the rule evaluation which should be a deny or accept decision over a particular request
- the *condition* which is an optional boolean expression further refining the applicability of a rule beyond the predicates specified on its target. Such expressions can use a broader set of function and be able to compare two or more attributes. Through conditions, the standard is actually able to achieve segregation of data checks as well as realize RBAC.
- the *obligation expressions* which define obligations/directives dictated by a Policy Decision Point (PDP) to the Policy Enforcement Point (PEP) on what must be performed before or after an access to a resource is granted. If a PEP cannot comply to an obligation, then the approved access might not be realized. Thus, it seems that obligations are used to fill the gap between formal requirements and policy enforcement.
- the *advices* which are similar to obligations but are not necessarily enforced by a PEP.

It must be indicated that a target can be identified also for policies and policy sets. Moreover, the standard defines different ways (i.e., realizations of the rule/policy combining algorithms) where the results of rule or policy evaluation are combined into a final decision/evaluation for policies and policy sets, respectively. Some of these ways are the following: (a) deny-overrides (any single deny evaluation results to a combined deny evaluation value), (b) permit overrides (same for previous one but not a single permit evaluation influences the decision result), (c) first applicable (from the ordered sets of rules, promote the decision of the first applicable), and (d) only one applicable (only one rule/policy should be applicable and its decision is promoted).

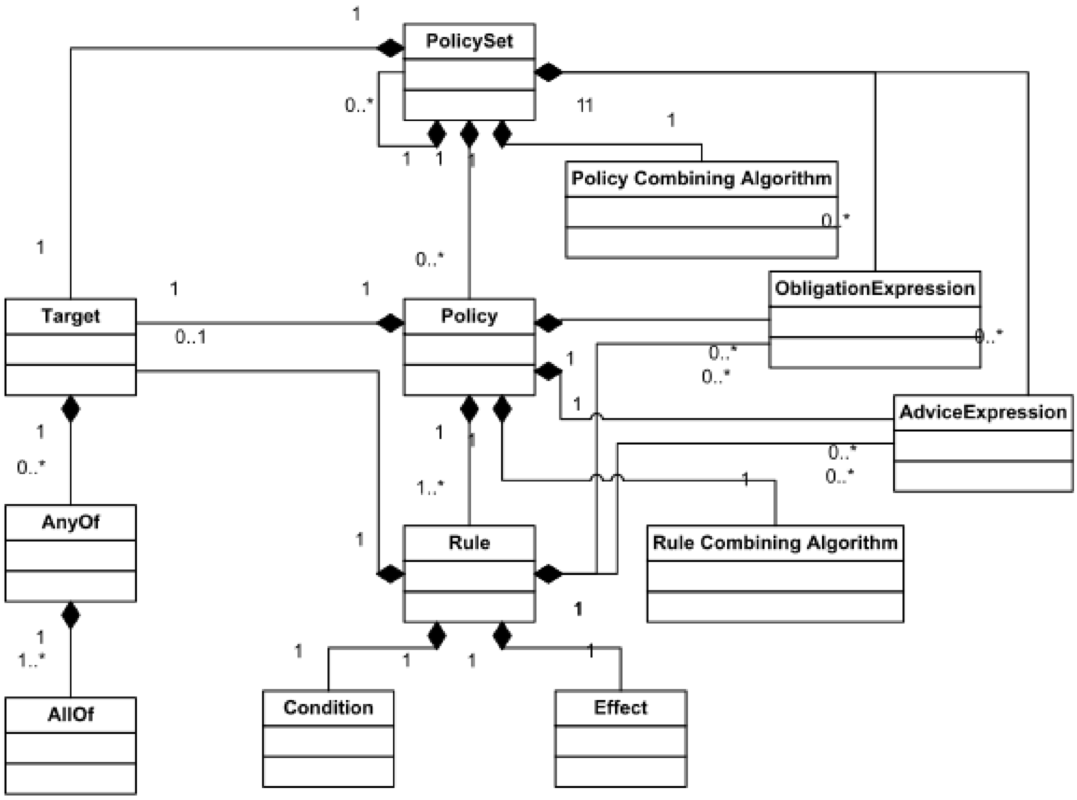


Figure 23 - The UML class diagram of XACML

Description of Respective MDDB Schema Part

Based on the CERIF extension, which is described in detail in Section 3.1.2, the MDDB schema was updated with information which can be expressed by CERIF concerning the modelling of organizations, user and roles and includes the description of policy information. In particular, in the current version of the MDDB schema, there is the notion of permission which can be mapped to access control policies and is related to the (user) role that has the permission and to the action(s) that can be executed by this role. The resource affected by the permission is described through the resource table and maps to all possible resources that will be available by the MDDB prototype and its components. For instance, concerning the MDDB itself, it is possible to describe resources at the table level and express which PaaSage components/modules will have read and write access to which MDDB tables. Moreover, the mapping of permissions to access control policies seems to be at the rule level but it is essential to distinguish which parts of these two elements description are going to be mapped as there is a gap between the declarative description in XACML and the direct description in MDDB schema.

Running Example

Based on the running example, suppose that SINTEF has issued a local policy indicating that any user with the role "Modeller" can modify and run the SENSAPP application. An example of a XACML specification describing such a policy could be the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy  1030
```

```
    xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
    xmlns:xacml ="urn:oasis:names:tc:xacml:3.0:core:schema:wd-
17"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:md="http://www.med.example.com/schemas/record.xsd"
PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:1"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-
combining-algorithm:deny-overrides" Version="1.0">
       <PolicyDefaults>
       <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-
19991116</XPathVersion>
       </PolicyDefaults>
       <Target/>
       <Rule
RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:1"
Effect="Permit">
       <Description> A user with the "Modeller" role can
modify and run the SENSAPP application
     </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"
>Modeller</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:ro
le"
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"
>10</AttributeValue>
       <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"
>1</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>
       <AllOf>
       <Match
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string"
>2</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>
```

This XACML specification would then be transformed into the following MDDB content:

Table *permission*:

id	role_id	resource	allowed	start	end	issued_by
1	1	10	yes	...	...	3

Table *role*:

id	name
1	Modeller

Table *permission_action*:

id	permission_id	action_id
1	1	1
1	1	2

Table *action*:

id	action_type_id
1	1
2	2

Table *action_type*:

id	name
1	Modify
2	Run

Table *resource*:

id
10

Table *application*:

id	name	version	user	resource
2	SensApp	1.0	4	10

Table *organization*:

id	name	www	postal_address	email
3	SINTEF	www.sintef.no	...	...

3.1.8 Mapping Considerations

To realize the identified mappings at the schema level, we could rely on a set of technologies that are already available as well on particular assumptions on the actual transformation requirements. By starting from the latter and by considering that a object-to-relational interface will be developed, we can assume that there are two transformation modes:

1. The domain objects are directly mapped to MDDB content in a straightforward manner;
2. The domain objects are processed (possibly after being transformed to model code) through procedural code in order to realize the more involved mappings to the MDDB.

The first mode seems more appropriate when the mapping is easy to realize and usually one entity/object maps to the content of one or two MDDB tables. For this mode, we can consider annotation techniques which enable defining the mapping between the domain classes and the MDDB tables. On the other hand, the second mode is better when the mapping is not so easy to realize in a one-to-one or one-to-many manner but also some procedural logic is required. It is obvious that the techniques activating in the second mode are more powerful than the first and can accommodate any type of required mapping.

In our opinion, both types of modes can be exploited depending on the mapping case by especially considering that sometimes you do not need great expressive power in order to do something simple as this can spend more resources and lead to more realization effort. For instance, the mapping between a Saloon model and MDDB is better to be handled by the second type of mapping techniques as one Saloon element/entity is not always mapped to the same MDDB table but the mapping actually depends on the actual element content (at the model level). On the other hand, the mapping between CloudML and MDDB is more straightforward and an annotation-based technique could be more appropriate in this case.

Based on the above rationale, we shortly analyse in the sequel some particular mapping techniques that could be used for realizing the identified schema mappings. We also assess which of these techniques could be more suited for handling the mapping cases designated.

3.1.8.1 Mapping Technology Analysis

3.1.8.1.1 Code Generators / Persistence Solutions via Annotations

3.1.8.1.1.1 Acceleo

Acceleo (current version 3.0) is a code generator which conforms to OMG's Model to Text specification (MOFM2T 1.0). It allows you to handle the whole life-cycle of the code generator, including offering refactoring tools as well as a debugger, profiler and traceability API. Acceleo does not restrict the input model used in the generation as it can span across UML models, ecore models, and (custom-made) meta-models (XMI). It also allows you to use any EMF tool to create the input model, such as EMF tree-based editors, graphical editors, text-based editors and CDO applications. It follows a template-based approach in developing any type of code, such as Java, Scala, Javascript, and HTML. Thus, it can also be easily used to produce SQL code. A module under open-source licence (by Obeo) for transforming UML models to Java/JEE code is available which exploits Struts, Spring and Hibernate technologies. The latter indicates that Hibernate can be exploited through this code generation framework to create the appropriate data-access layer (between objects and relational data) enabling capabilities such as cache handling, 3NF SQL, integrity constraints and concurrent access.

3.1.8.1.1.2 Teneo

Teneo constitutes an EMF-based solution enabling the mapping between models and relational data as well as a respective runtime persistence layer. It exploits the Hibernate and EclipseLink technologies to persist Eobjects, Elists and other types of objects and relations as well as to support JPA-compliance. Both Hibernate and basic JPA annotations can be exploited in order to map model objects to relational data. While the former type of annotations is more suitable for one-to-one types of mappings, the latter type enables the mapping of model objects to more than one RDB table as well as the support of other interesting mapping capabilities (enabling the use of join tables, inheritance mappings, etc.). The annotations can be entered in the model (XSD or ecore) or specified in a separate XML file to separate the domain from the annotation logic. The persisted objects can be retrieved either via an EMF-based resource approach or through HQL queries. Most RDBs are supported. It must be noted that Teneo is actually used as a component of the CDO Hibernate Store (see Deliverable D2.1.2 [D2.1.2].

3.1.8.1.1.3 Texo

Similarly to Teneo, Texo provides a JPA-based solution where the JPA annotations can be generated and enhanced at the source code or an orm.xml mapping (ORM 2.0 standard is supported) can be generated and extended from ecore/xcore models. Texo is able to generate true PoJos from Ecore/Xcore/XSD models with no direct compile link from the entities produced to EMF. It actually supports overridable and extendable code generation with important support for EMF tasks like merge and formatting. It also provides a web services layer for performing querying and CRUD operations. It should be highlighted that it also provides appropriate runtime support through models@runtime thus enabling the realization of important functionality such as security, archiving and code generation. Finally, it should be mentioned that support for multiple annotation models (e.g. JPA and Hibernate) is planned.

3.1.8.1.1.4 XDoclet

XDoclet is an open-source code generation framework which supports attribute-oriented programming through Java. It parses the input data/files provided and through the appropriate templates it generates the respective information, such as XML descriptors or code. The generation process is defined through Jakarta Ant. Although this framework originally aimed at producing EJBs, it has now been transformed into a general-purpose code generation framework. This framework provides support for all major server and tools, such as IBM WebSphere, JBoss, Oracle IAS. Similarly to Acceleo (although Acceleo is rather more weak at this point) it also exploits various technologies, such as Hibernate, Castor, many JDO vendors, Struts and WebWork. Thus, as it can be seen, Hibernate annotations can also be exploited in the code generation. In fact, different annotation models are supported depending on the underlying technology (e.g., Spring bean annotations, Jboss annotations and so on).

3.1.8.1.1.5 AXGen

AXGen is a another code generator engine which takes as input XMI files and transform them to any type of code. It uses Velocity templates to perform the transformation and thus relies on Velocity's sophisticated template engine. It offers an UML abstraction layer (through Nsuml or NetBeans MDR) which provides an object-oriented view to the model. AXGen exploits an OR Mapping tool in order to persist the model object into a relational database, where OJB (Apache's ObjectRelationalBridge) technology is actually exploited. While OJB is quite powerful and is comparable to other persistence frameworks and technologies, the corresponding Apache project has been retired.

3.1.8.1.1.6 XMI-to-SQL

XMI-to-SQL is a free library that can be used to transform XMI input files to SQL code. This library was implemented in the context of a bachelor project. It requires that the input models conform to XMI version 2.1 and are produced by an Enterprise Architect v7.0 or above project. Moreover, to enable a proper code generation, the project must contain a Class diagram and its version must be transformed to DDL. Due to the context of this project, we need to actually check whether its functionality is stable and can be exploited for performing the respective transformations. We need also to check whether there is actually a one-to-one mapping between XMI model entities and the respective RDB tables.

3.1.8.1.2 Model-to-Model Transformation / Procedural Approaches

3.1.8.1.2.1 XSLT/YSLT

XSLT is a XML-based specification for model-to-model transformation which follows a procedural way for defining mappings. It is widely used especially for transforming XML-based models to models in XML or other formats. It supports the XPATH standard which can be used for identifying any component of a particular (input) XML specification. Despite its popularity, XSLT exhibits particular issues which can be a little bit annoying to the (mapping) modellers: (a) no separation of indention of the XSLT program and that of the output text, (b) complicated syntax and (c) lack of good escaping mechanisms. To this end, some extensions have been proposed, such as YSLT which solves the above problems. Due to its simplicity and expressive power, X/YSLT could be used for defining any type of transformation, such as the XML-to-SQL one that we desire to achieve. The only issue that we foresee with its use is that it is up to the modeller to produce correct resulting models. Thus, it is usually good practice to check in the end whether the resulting model complies to its meta-model and is correct.

3.1.8.1.2.2 MOLA

The MOLA project has been developed with the aim to offer a simple and user-intuitive graphical model transformation language which is able to cover all the typical transformation scenarios in the context of the Model-Driven Software Development. The project offers the following artifacts: (a) the MOLA (Model transformation LAnguage) language and (b) the MOLA tool enabling users to define MOLA transformations and execute them on models. The MOLA language is quite powerful and exploits the well-known concepts of pattern matching and rules for identifying the appropriate input model parts to be transformed. The order of rule application is specified through exploiting traditional programming constructs, such as sequence, loop and branching. Another interesting characteristic of the language is that it allows the use of variables and calls. A complete transformation specification in MOLA comprises a MOF-compliant meta-model and a set of MOLA transformation diagrams (procedures) which constitute a MOLA program. The result of the transformation is a model which conforms to a MOF-compliant meta-model. MOLA diagrams are actually sequences of graph statements linked by arrows which start by a UML start symbol and finish with a UML end symbol. The MOLA tool is freeware and comprises the following components: (i) a graphical editor for metamodel and MOLA procedures, (ii) a MOLA compiler set able to produce C++ or Java code using intermediate transformation languages and (iii) the MOLA runtime environment which has been constructed from meta-model in-memory repositories (mii_rep, JGralab and EMF). In the context of this project, the transformation example of UML class model to RDB content has been realized according to the QVT-P proposal and the respective MOLA program is available.

3.1.8.1.2.3 ATL

ATL is a model transformation language which is defined both as a meta-model and a concrete textual syntax. ATL can be used to transform a set of input source models to one target model. The main rationale is that all these models conform to a specific meta-model and that the model transformation specification conforms to the ATL meta-model. Model transformations in ATL can be specified either in a declarative or a imperative way, where the declarative is the preferred one which is bypassed only when the transformations cannot be easily expressed in a declarative way. Such model transformations contain rules which indicate how to navigate and select the appropriate elements of the input source models for the generation of the respective elements of the target model. ATL also offers a model query facility which can be used to issue requests on models. Code can also be factorized in ATL via the definition of ATL libraries. ATL is accompanied with an Integrated Development Environment (IDE), developed for the Eclipse platform, which offers various development tools through which the model transformations can be specified. This environment also offers additional facilities which can be used for model and meta-model management, including a simple textual notation for meta-model specification and a number of bridges between common textual syntaxes and their respective model specifications. Through ATL, a MySQL to KM3 transformation has already been defined which could be used in the context of this project for transforming relational data to respective model representations. It must be highlighted that the model transformations in both directions have been defined such that also a model representation can be mapped to the respective relational content.

3.1.8.2 Assessment

From the above analysis, it is apparent that there are various approaches that can be used to realize the two designated mapping cases. Thus, the goal of the consortium is to decide which candidate language to use for each case. By considering the annotation techniques, we can distinguish Teneo which is an Eclipse-based solution which supports JPA and Hibernate annotations and is a component of a CDO Hibernate Store. This seems a stable solution which has been adopted in order to build a more added-value persistence technology for CDO repositories. As far as the procedural techniques are concerned, we can have two different choices: (a) if we go for simplicity and not for exploiting a quite involved technique which might have more power than actually needed, then XSLT could be our choice and (b) if we desire a more sophisticated approach which also ensures that the produced model conforms to a particular meta-model, then we should go for ATL which is a language that is widely used for model-to-model transformations and already has available various (sample) mappings as well as a graphical development environment for specifying model transformations.

3.2 Security, Privacy & Trust for MDDB Prototype

A key feature of the MDDB is the integration of data from third parties. We envisage that this integration can be done via authentication and authorisation of a federated organisation with the specific PaaSage instance. Sources data could include social networks, federated implementations of PaaSage, trusted business partners plus others. In these cases users from trusted organisations can share data with the PaaSage implementation using credentials from their home organisation.

The Authentication and Authorisation provides the basis on which the trust fabric used in PaaSage is built. Trust is derived at user and organisational level. Federated users authenticate with PaaSage and are associated with unique identity tokens which contain their role and organisation. These tokens contain access privileges for the user in the PaaSage instance which is derived from the role and organisation of the user.

Authentication and Authorisation linked trust is present within the PaaSage identity management framework which is linked to the MDDB. This framework supports single sign on from users via the use of OpenID, SAML and OAuth. The identity assertions from each method of authentication translated to a trust level linked to the user's previous use and reputation of their organisation. For example an OpenID user could have lower trust than a previous good quality SAML federated user from a trusted partner organisation.

In terms of secure access to data the trust associated with the user is expressed as permissions in the user's identity token. This token is issued by the identity management component and checked by the MDDB in order to grant access for users. Security policies are present in the MDDB in order to control the level of access to the retrieval or recording of data. Policy is expressed in XACML and covers who has access rights to data and what the access rights are. Policies are enforced by Policy Enforcement Points (PEP) and Policy Decision Points (PDP) before access is granted. The flow can be seen in Figure 24 below.

As the figure illustrates the access to data is based on both identity granted from the MDDB and policy stored in the MDDB. The PEP and PDP components sit outside of the MDDB and can be provided by a third party in order to ensure neutrality from the PaaSage platform when policies are applied.

The combination of authentication, authorisation and the use of secure identity tokens and policy provide the backbone to the trust, security and privacy architecture of the MDDB. As PaaSage evolves the trust model is expected to become more complex and the use of the PDP and PEP will support different deployment architectures and the potential use of trusted third parties for Policy Enforcement.

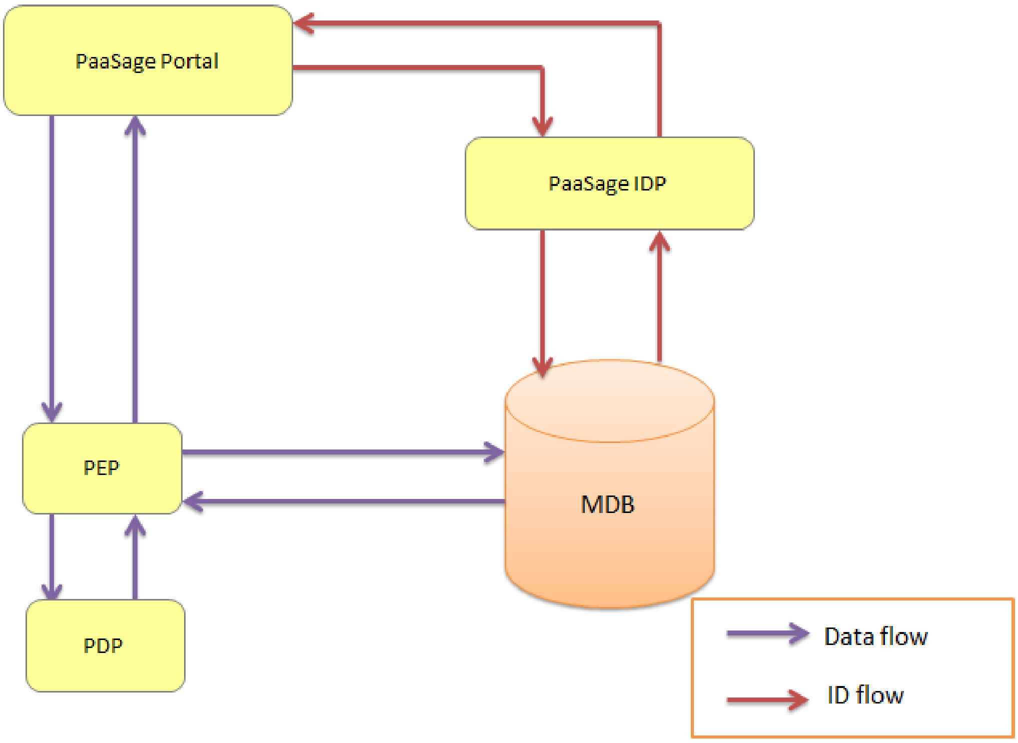


Figure 24 Trust and identity management

3.3 *Scalability, Availability*

The current MDDB prototype is implemented over standard relational (SQL) database technologies. Centralized relational databases are known to exhibit scalability and availability issues. In PaaSage we aim to address this challenge in a number of ways. First, standard SQL database management technologies offer scalability and availability features that can be used in the context of PaaSage. For example, the MySQL *cluster*[7] option can seamlessly offer horizontal partitioning of the dataset into several nodes achieving parallelism (scalability) as well as availability via replication. While effective in sustaining larger levels of load than single-node installations, options such as MySQL cluster are known to exhibit limited levels of scalability. There are two alternative options to scale beyond the limits of conventional technologies. First, leverage recent advances in ultra-scalable transactional processing systems[8]; or second, explore the NoSQL class of database management systems (examples include Apache HBase, Cassandra, MongoDB, and others), which however come at the expense of losing full transactional semantics. Besides scalability and availability, the two alternative options offer good elasticity features as well. In our evaluation of the MDDB we aim to determine the performance and availability requirements of applications from the MDDB and consider the appropriate database technology to use based on those requirements and the results of our evaluation.

[7] http://www.mysql.com/products/cluster/

[8] http://www.cumulonimbo.eu

4 Knowledge Base Design & Implementation

Design Rationale

While the MDDB schema is capable of storing a vast amount of different types of information which is appropriate for the proper management of cloud-based applications, there is a clear need of additional information which can be derived from the lower-level stored one which could be of an added-value to many tasks/processes involved in cloud-based application management. Obviously, such information could be produced by processing the data stored in MDDB but this would have the following main drawbacks: (a) the MDDB would be loaded with additional queries possibly affecting many parts of the data stored, (b) the knowledge produced through MDDB content processing could be required by many PaaSage components and this will certainly lead to a redundancy of computation/processing for the production of the same information, and (c) even if a dedicated component is realized in order to implement the above processing functionality, again there will be the issue that the information produced must be cached or stored in order to avoid performing redundant computations.

To this end, it was decided that a new component is required which is responsible for deriving additional, high-level knowledge from the MDDB content which can be queried at any time by the PaaSage components in order to draw particular parts and exploit them. Such a component would need to consider the following requirements which mainly concern the way the processing functionality must be realized, installed and updated, the way the derivation knowledge can be managed and the way the MDDB content can be exploited in order to produce the required, high-level knowledge:

- Knowledge should be produced once and updated only at particular time points or cases (e.g., when particular MDDB data are updated) in order not to frequently burden the MDDB layer with additional load.
- There should be a simple and user-intuitive way to query the new knowledge derived.
- The new knowledge should be persisted.
- New knowledge derivation functionality (e.g., the one produced/proposed by expert users of the PaaSage SN) should be realized and installed in an easy manner into the component.
- The knowledge should be built in a bottom-up manner and be re-used as much as possible, especially when new knowledge derivation functionality is needed.
- The communication between this new component and the MDDB layer should be realized in an efficient and appropriate way such that no time is spent in transforming the MDDB content into the format required by the new component.
- There should be facilities for the proper management of the knowledge and of the functionality used to derive it.
- Particular programming constructs should be provided in order to realize the processing/derivation functionality in a uniform manner.

Candidate Technology Selection

Based on the above requirements, two main candidates were identified: (a) knowledge bases and (b) ontology-based reasoners which could fulfil many of the above requirements, if not all. However, the decision finally went to knowledge bases as: (i) there are already particular efficient facilities in order to draw and exploit database data and (ii) ontology-based reasoners require that the MDDB data must be already available or transformed into RDF (or other ontology language) data and (iii) some project partners are not familiar with the semantic (ontology-based) technology.

Nevertheless, the main logic between these two alternatives is similar as rules are mainly used as the basic constructs for the derivation of new knowledge. In addition, new rules can be inserted in such way that they exploit the knowledge produced from old rules in order to produce novel knowledge and this perfectly maps to the bottom-up approach designated by the above requirements. Moreover, the modern rule languages utilized by the current state-of-the-art knowledge base frameworks exhibit a particular user-friendly syntax and provide various important and sufficient constructs which can be exploited in order to produce an efficient set of rules, most of the times in a programming-language independent way (although some frameworks can also enable you to write rules in two modes in order to utilize your favour programming language, such as Java). Finally, most of the current state-of-the-art frameworks enable you to manage and easily query the knowledge derived (with some also exhibiting caching capabilities) as well as manage the rules developed in various meaningful ways. To conclude, knowledge bases perfectly match the above requirements and are the best candidates for the realization of the required MDDB component.

Open-source KB frameworks can be separated into two types: (a) internal KB engines operating on the DB of the DBMS and (b) external KB engines operating on data fetched from a DB through a particular technology, such as Hibernate. From these KB frameworks, the internal ones are rejected as the most sophisticated frameworks are provided only by proprietary solutions that are part of a specific DBMS. To this end, we resorted and evaluated only external, prominent open-source KB frameworks, namely Drools Expert, Jess and Prova, based on the following criteria:

- powerful rule building constructs
- simple and user-intuitive rule and query language
- simple and efficient integration with DBMS technologies
- extensive documentation
- extensive and highly active user base

Drools Expert fulfilled all of the requirements, while the other frameworks were not satisfying one or more of them. For instance, Drools has native support for Hibernate while Jess does not and this means that efficiency issues might arise when DB content is queried and fetched. Moreover, the user base for Jess and Prova is small or getting smaller while there seems to be a move or trend to resorting to the solution of Drools Expert. Apart from this, another factor that influenced our decision was the fact that Drools Expert is accompanied by other, complementary open-source solutions that could be exploited, such as the Drools Guvnor which is a centralized repository for KBs.

Knowledge Base Design

Drools Expert as well as many other current state-of-the-art frameworks provide you with various conceptual elements which can be used for the management of rules and the derived knowledge. In particular, the main conceptual element is a *Session* which can come in two flavours:

1. Stateless sessions which can be used to produce and return the whole knowledge derived at once. After that, they are usually destroyed.
2. Stateful sessions which can persist and which can be continuously exploited by firing rules, issuing queries and obtaining back the results, and adding new objects (if needed). Such sessions can be easily informed when new rules are added in order to fire them and produce additional knowledge on top of the one already derived.

Sessions are connected to *KnowledgeBases* which constitute the management point of rules. In this way, once a new rule is added, the session is automatically updated. Moreover, when a Knowledge Base is removed, then all the corresponding sessions are usually removed (or no more valid). KnowledgeBases can also persist and this is crucial in order not to lose the knowledge production logic.

Based on the aforementioned description, it was decided to realize a component through web service technology which will be responsible for managing both different knowledge bases as well as their respective stateful sessions. Stateful sessions were chosen as there was a clear need to not only persist the derived knowledge but also to have a continuous management medium through which this knowledge could be updated and queried. On the other hand, the design choice of enabling the creation of many knowledge bases relied on the following rationale:

- a basic knowledge base is needed for holding the formal core rules that are exploited by the PaaSage components which is usually close and only updated after it is decided that a new rule proposed by an (expert) user should be accepted.
- other knowledge bases can be created by PaaSage components or even PaaSage users in order to fulfill their own individual goals. Such knowledge bases can be pre-loaded with the basic PaaSage rules as well as obviously extended with new component or user-specific rules.

In addition to this, the component also allows one component/user to create many stateful sessions for a particular knowledge base as needed in order to cater user requirements such as the one that different sessions should be kept for different parts of the knowledge/rule base (i.e., for a particular partition of the user-defined rule set).

As can be easily understood, the realized KB service is able to manage the whole lifecycle of KBs and the sessions created out of them. The KB management methods provided include the following:

- create a Knowledge Base
- add a set of rules to a Knowledge Base
- remove a Knowledge Base

while the session management methods offered are the following:

- create a session for a particular knowledge base
- fire rules for a particular session
- get all objects of a particular session
- add objects to a session
- issue a query to a session
- delete a session of a knowledge base

Apart from providing these methods, the service also ensures that all KBs and sessions created as well as the rules provided are persisted in physical storage (plus other configuration information) such that even if it is interrupted or restarted, it can load all the appropriate KB, session and rule information such that no user data are damaged or lost.

Important details about how to use the service and what is the signature of the service methods can be found in:

https://projects.gwdg.de/projects/paasageproject/repository/raw/Work%20Packages/WP4/KB_CLIENT.zip

where the client API code is also included and can be exploited.

The current content of the KB component relies on the exploitation of rules which derive three types of knowledge: (a) similarity degree between application and between artifacts, (b) successful deployments of applications and artifacts and (c) best deployments of applications and artifacts according to one or more quality requirements. The similarity degree of artifacts is currently derived through name matching but it is planned that additional information is exploited once it is available in the MDDB. The similarity degree of applications is derived based on the percentage of common or equivalent artifacts between these applications. Similarly, it is planned that additional information is also exploited when it is also available in the MDDB (or can be derived from other files or models of the application). Apart from the rules that derive the above knowledge, particular queries have been designed which allow to enquire successful/best deployments for new artifacts or applications based on the derivation knowledge of old artifacts or applications that match them, respectively. The answer of such queries can be exploited during CloudML model design as modelling hints to designers for new applications which are depicted and loaded by the SN or during reasoning in order to obtain interesting deployments for new applications that could be potentially exploited. New rules and queries are planned to be designed concentrating on the same or even different aspects of the cloud-based application management information stored in the MDDB.

The knowledge derived is structured into objects that are instances of particular Java bean classes. These classes build upon the Java bean classes corresponding to the tables of the MDDB schema. This is a basic requirement for exploiting the DB fetching mechanisms of the exploited KB framework as the DB content is drawn in the form of objects that are instances of the latter classes. It should be noted here that the DB content is fetched by particular queries which are placed in the "if" part of rules and which are issued once when the rules are pre-checked if they can fire. This means that the DB queries map to static knowledge which is produced once and exploited by one or more rules. The re-evaluation of queries, if needed, is only enforced through performing particular tricks. Thus, DB queries in rules must be handled with care. We should mention here that this is a common restriction that appears in most of the KB frameworks reviewed.

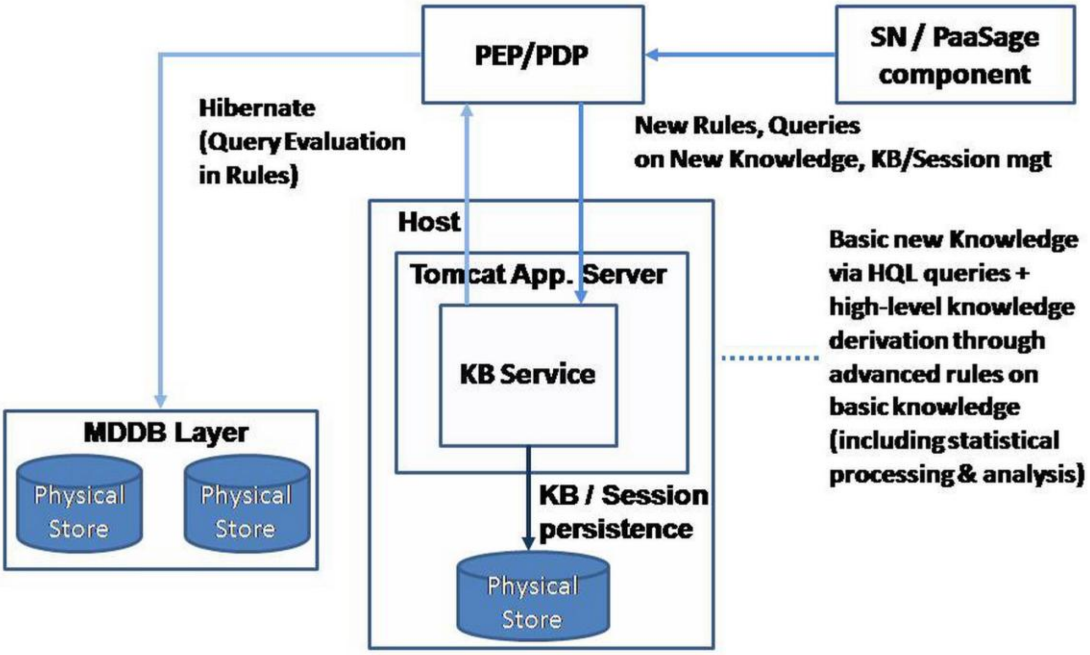

PEP/PDP
SN / PaaSage component
Hibernate (Query Evaluation in Rules)
New Rules, Queries on New Knowledge, KB/Session mgt
Host
Tomcat App. Server
KB Service
Basic new Knowledge via HQL queries + high-level knowledge derivation through advanced rules on basic knowledge (including statistical processing & analysis)
MDDB Layer
Physical Store
Physical Store
KB / Session persistence
Physical Store

Figure 25 - The architecture of the KB Component

The architecture of the realized component is depicted in Figure 25. As it can be seen, the user / PaaSage / MDDB component issues any type of request to the KB management service which first passes by the PEP/PDP points for authorization (it is assumed that the user/component is already authenticated and has incorporated in his/her request the authentication token). Once the authorization is granted, then the respective method of the KB service is called. Depending on the method called (i.e., rule firing method), communication with the MDDB layer takes place through the widely-used technology of Hibernate in order to fetch (domain) objects (actually rows for a particular table which can be seen as a Java bean) from the MDDB as the raw information exploited by the "if"/"left hand side" of rules. The execution of many methods also results in persisting crucial KB/session management information in the current node that hosts the service, while in the future the information will be also persisted in other places for ensuring that no data are lost when critical host failures occur (such as physical disk faults). The service is currently deployed on a Tomcat application server running on a particular node of the Flexiant cloud and is available at the following URL (where also documentation information can be viewed): http://109.231.122.77:8080/ksession-manager-1.0-SNAPSHOT.

Implementation details

Implementation fully relied on the use of the Java programming language. The KB framework exploited is Drools Expert. This framework offers a Java API which enables the full management of KBs and their respective sessions. The main management functionality was wrapped into a restful web service which was realized through the jersey Java library, while the enunciate library was used for producing the web-based, user-friendly documentation of the service. The Java Hibernate library was used for fetching the appropriate MDDB content through HQL queries in rules.

5 Social Network Design & Implementation

5.1 Core Design

The PaaSage social network is implemented over the extensible Elgg social network framework. Elgg comprises a Core system and a number of plugins, all of which share a common structure. All objects in Elgg inherit an ElggEntity Object, which provides the necessary general attributes of an object. Elgg Core comes with four basic objects: ElggObject, ElggUser, ElggGroup, ElggSite, ElggSession, and ElggCache, and a lot of others classes necessary for the proper engine operation.

The extensibility of Elgg can be established not by modifying the core system but by introducing new plugins. The plugins follow the MVC (Move View Controller) model. A new plugin can create new objects (e.g., ApplicationObject for the rest of the document we mention as draw_app), which extend ElggEntity. Thus, each Entity is characterized by a numeric Globally Unique IDentifier (GUID), owner GUID, and Access ID. The Access ID determines the permissions that other users have. Thus, when a page requests data, it never touches those data that the current user doesn't have permission to see. Other application-specific attributes are defined within the application plugin such as the view of a specific object. All Entities may have relationships between each other or may store data (metadata) to the Elgg Database.

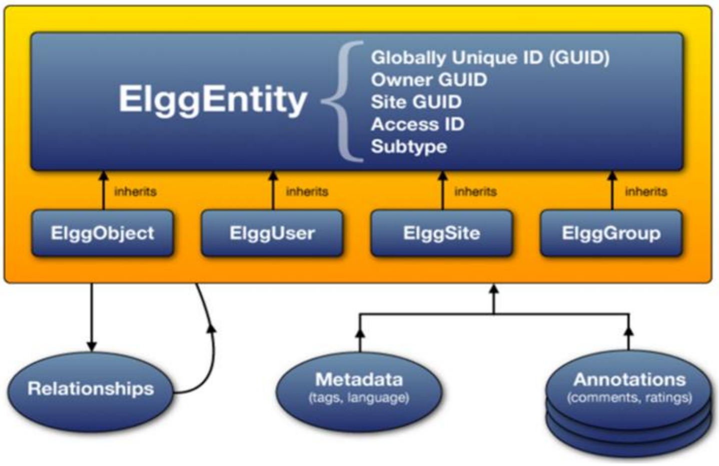


Figure 26 - Elgg Structure (a)

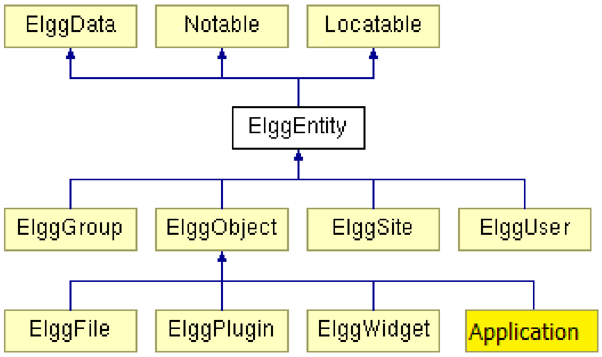
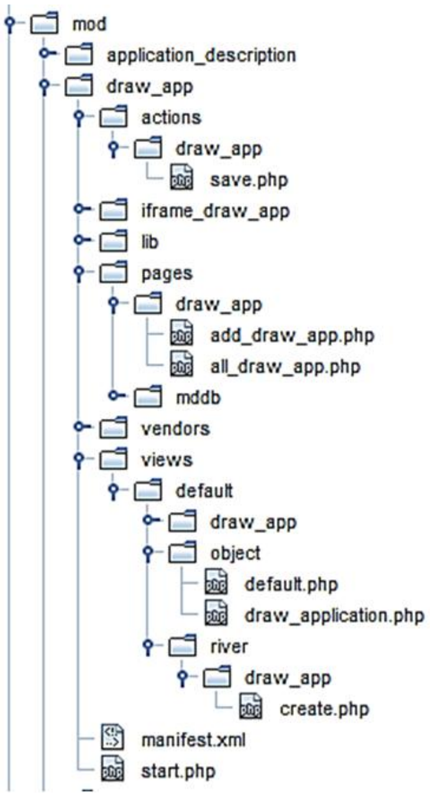


Figure 27 - Elgg Structure (b)

The implementation of a new plugin follows a well defined directory structure such as the one shown in Figure 28 for the Application plugin. The structure resembles a tree whose leaves are executable php, css or javascript scripts.



```
mod
  application_description
  draw_app
    actions
      draw_app
        save.php
    iframe_draw_app
    lib
    pages
      draw_app
        add_draw_app.php
        all_draw_app.php
      mddb
    vendors
    views
      default
        draw_app
        object
          default.php
          draw_application.php
        river
          draw_app
            create.php
    manifest.xml
    start.php
```

Figure 28 - Plugin structure

Any plugin (including Application) has:

- Default views. Views are responsible for creating the output. Generally, this will be HTML sent to a web browser, but it can be also JSON or other data formats.
- Actions, which are Elgg's way of providing interactivity: every active participation by the user is performed via an action. Logging in, creating, updating or deleting content are all generic categories of actions. For instance, the save button of an application composition triggers a "save" action.
- Events. There are two types of events: Elgg Events and plugin hooks. Elgg Events are triggered when something is created, updated, deleted or when the Elgg framework is loading. Examples would be a blog being created or a user logging in. Each event is determined by an event name and an object type (draw_app, system, user, object, or group). By registering handlers for Elgg events, our plugins can have code executed when that event occur, such as river creation events. The river creation is the event that is responsible for the news activity posts of the network. Plugin hooks tend to be triggered when an action has occurred and parts of that action can be overridden by a plugin though there are additional cases where a plugin hook can be used to overwrite the action.
- Pages, which can be from chunks of presentation output (like sidebars or objects) down to individual html code.
- Libraries, such as jsPlumb which is in the vendors folder
- A page-hander function described in start.php is a facility to manage our plugin pages, enabling custom url redirect to a specific page php script in a specific folder. The plugin initialization is also defined in the start.php and registers actions, events and determines the views. For example, a "Save" user action in the draw_app (when the "Save" button is pressed) produces a "Save" action in our plugin, and this action adds a "river" stream functionality which is handled within "Activity" (core system) and creates the view for our own object in the home page.

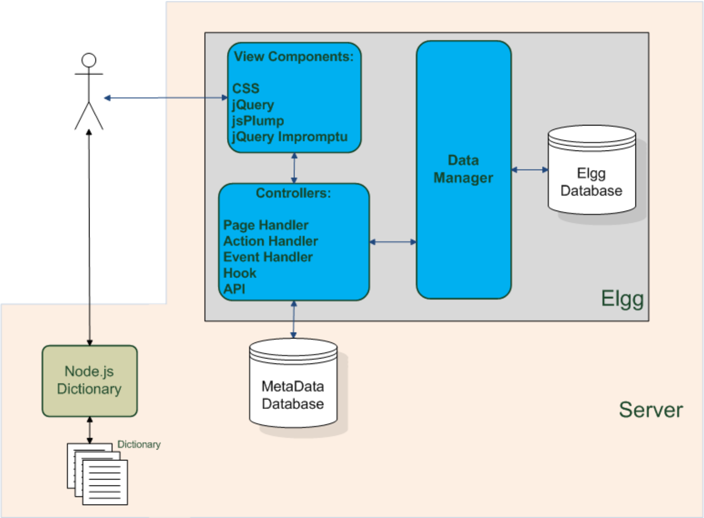


Figure 29 - Social Network Server Architecture

The Elgg Social Network is implemented as a classic MVC framework which also exhibits the standard functionality of a social network engine. In order to extend its operability to compose the social network according to the project requirements, the following extra plugins were created:

- An Application Description (draw_app) plugin that allows users to compose / view their applications in a graphical approach similar to an IDE.
- A MDDB plugin, which is responsible for the Metadata Database interaction and provides a standard HTTP API to execute queries. The more sophisticated users can post a pending query directly to MDDB through the social network. For security reasons, the moderators of the web page have to accept the query as benign in order to be executed.
- A Custom View query plugin, which modifies the default CSS view of Elgg.
- A Group plugin, which is responsible for the groups and discussion html pages.
- A Message plugin which allows users to send each other private messages.

Javascript Libraries

We have imported the jQuery Impromptu library in our system as it is a extension which provides a more pleasant way to spontaneously prompt a user for input. More or less this is a great replacement for an alert, prompt, and confirm javascript pop-ups as it does not only replace them but it also allows for creating forms within these controls.

The jsPlumb is the core system of the Application Description plugin. jsPlumb provides a means for a user to visually connect objects. Furthermore, with the help of jQuery Impromptu the user can configure objects and execute queries to MDDB.

Node.js Language Processing

In group discussion forms we need to interact with the user in order to provide information from the MDDB when the user is typing a topic. Apart from the jQuery interaction, we use the node.js to determine if a word is a natural language word or is a keyword known to MDDB. When a word is neither of them, then with jQuery Impromptu the user can define this word, providing a definition and a classification of the keyword. Apart from javascript libraries that perform the same work, the node.js is faster and does not overwhelm the client with a big dictionary file and the process complexity to look up for words.

As a future work, we will include natural/node.js which can classify sentences and offer a way to understand what the user is trying to ask through the topic and check if the MDDB knows something about it.

Interaction with MDDB and KnowledgeBase

The MDDB query plugin provides an API to logged in users to execute queries about the deployment of applications. Those queries are being executed with jQuery Ajax post requests to the MDDB plugin. The queries currently exploited are the following:

- Similar Applications based on the name of the application.
- Similar Applications based on the artifacts they use.
- Deployment hints based on cost.
- The cloud provider on which a specific application was deployed in the past.

Integration with CloudML model editor

In the current prototype we have integrated a CloudML model editor provided by SINTEF that is shown in Figure 30. As the CloudML model editor is extended to provide additional features, we intend to maintain and expand its integration with the implementation of the social network.

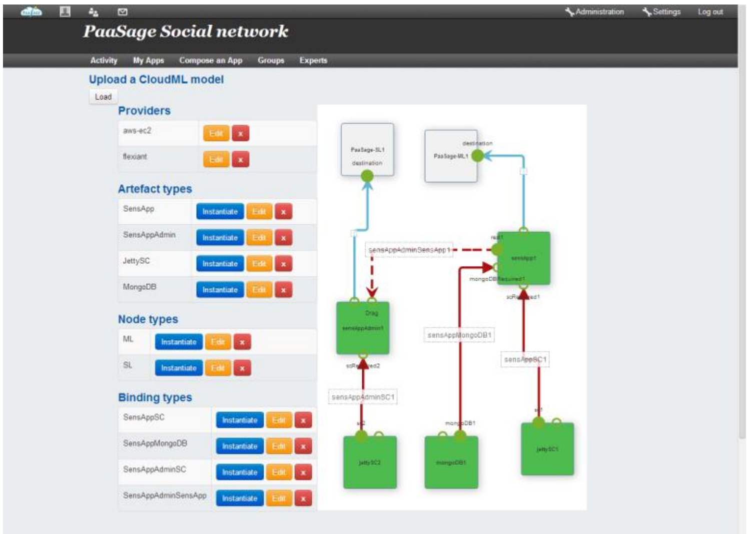


Figure 30: Integration of Model Editor with Social Network UI

5.2 *UI design*

In this section we describe ongoing work on UI design for the social network, including mock-up screenshots representative of the typical user experience.

5.2.1 Header

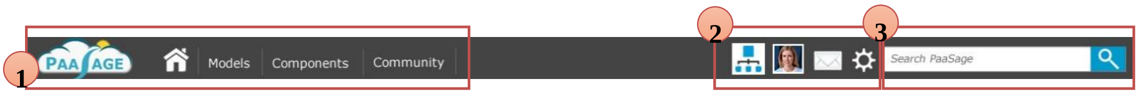


Figure 31 - The header shown in PaaSage's SN

(1) Main menu that facilitates the exploration of the PaaSage Social Network
   a. Activity feed (home icon): displays the most interesting recent activity taking place on PaaSage (depending on the user’s connections and preferences)
   b. Models: facilitates browsing among publicly available PaaSage’s application models
   c. Components: facilitates browsing among PaaSage’s software components
   d. Community: facilitates the exploration of PaaSage's social community (e.g., individual users, groups, discussions, etc.)

(2) Personal area: Includes options related to the profile of the logged-in user
   a. Models created - owned by the user
   b. User profile
   c. Personal Messages
   d. Account Settings

(3) System-wide search area: permits searching for models, components, groups, connections and messages

5.2.2 Models: Home

This screen is displayed when the “Model” option is selected. Its purpose is to help the user locate interesting models by displaying: (i) recommendations personalized to the user’s areas of interest, (ii) featured models promoted by the network and (iii) popular models based on community dynamics.

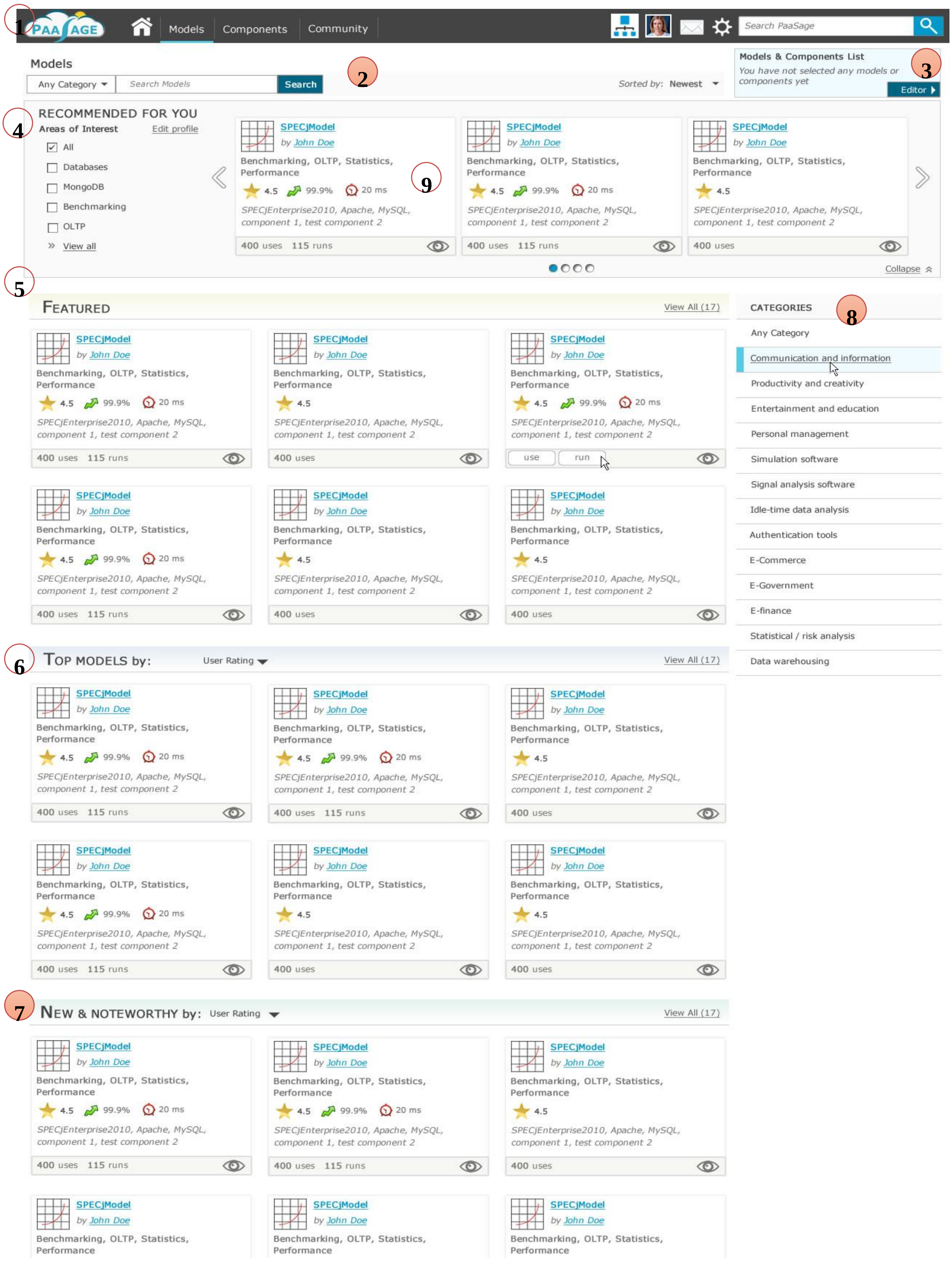


Figure 32 - Main page for models exploration in PaaSage SN

(1) Page title

(2) Page-wide search area: facilitates searching for models of the selected category

(3) Models & Components List: enables the collection of various models and software components that can be later imported for manipulation in the “Model Editor”

(4) Recommended models based on the user’s areas of interest

(5) Featured models promoted by the network

(6) Top Models; through a drop down menu the user can select the attribute based on which the “Top Models” will be selected (e.g., user rating, performance, number of uses, number of executions, etc.)

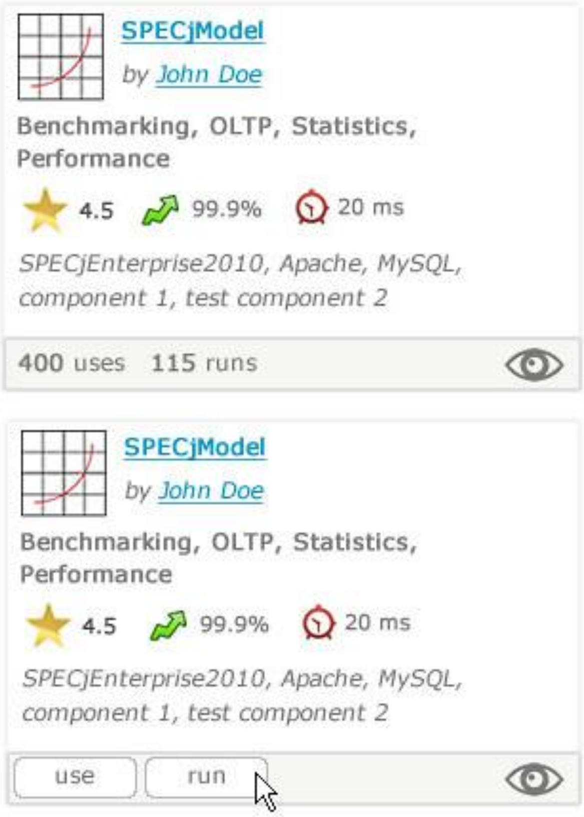
(7) New and Noteworthy Models; through a drop down menu, the user can select the attributed based on which the “New and Noteworthy Models” will be selected (e.g., user rating, performance, number of uses, number of executions, etc.)

(8) Sub-menu containing the main categories of the PaaSage models. By selecting a category, the page will refresh to display all the models of that category. The option “Any category” allows the user view all models regardless of their categories.

(9) Model Short View

5.2.2.1 Model Short View

Model short views contain the most important information of a specific model, so as to enable users to identify those that better suit their needs:



SPECjModel

by John Doe

Benchmarking, OLTP, Statistics, Performance

4.5 99.9% 20 ms

SPECjEnterprise2010, Apache, MySQL, component 1, test component 2

400 uses 115 runs

SPECjModel

by John Doe

Benchmarking, OLTP, Statistics, Performance

4.5 99.9% 20 ms

SPECjEnterprise2010, Apache, MySQL, component 1, test component 2

Figure 33 - The short view of models in PaaSage SN

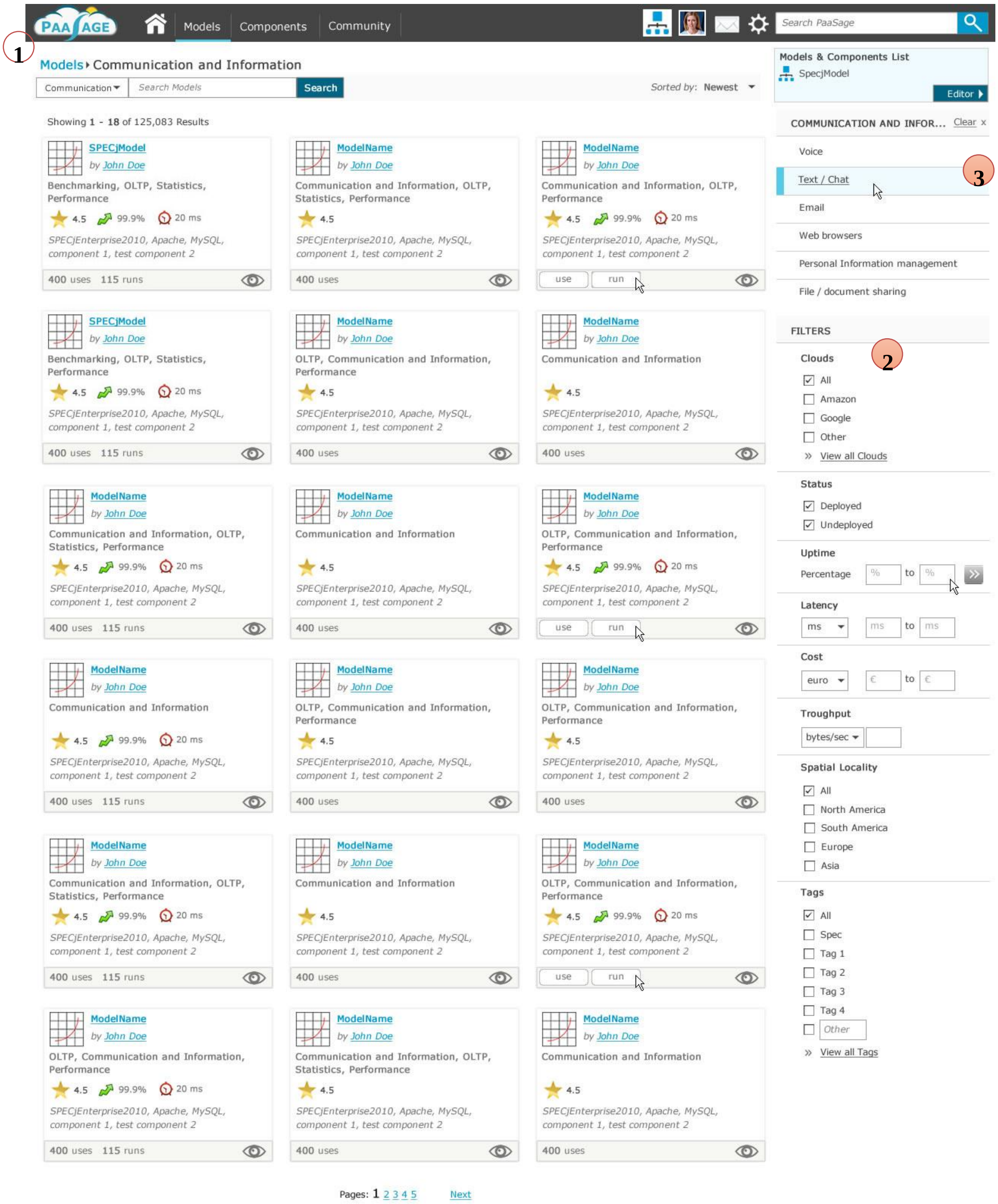
- Model Icon, Title and Creator
- Associated Category (-ies)
- User rating and Badges (e.g., 99.9% uptime, 20ms execution time, etc.)
- Associated tag(s)
- Number of uses and executions

A preview 👁 icon is also available for the users that wish to view further details about the model.

On mouse-over, the actions “use” and “run” are displayed. “Use” will add the selected model to the “Models & Components List” for later use through the model editor, while “run” will immediately launch the model editor for configuring deployment parameters and resources.

5.2.2.2 Models: Selected Category

This screen is displayed when a model category is selected.



1
PAASAGE
Models
Components
Community
Search PaaSage
Models ▸ Communication and Information
Communication ▾
Search Models
Search
Sorted by: Newest ▾
Showing 1 - 18 of 125,083 Results
SPECjModel
by John Doe
Benchmarking, OLTP, Statistics, Performance
4.5
99.9%
20 ms
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
400 uses 115 runs
ModelName
by John Doe
Communication and Information, OLTP, Statistics, Performance
4.5
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
400 uses
ModelName
by John Doe
Communication and Information, OLTP, Performance
4.5
99.9%
20 ms
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
use
run
SPECjModel
by John Doe
Benchmarking, OLTP, Statistics, Performance
4.5
99.9%
20 ms
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
400 uses 115 runs
ModelName
by John Doe
OLTP, Communication and Information, Performance
4.5
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
400 uses
ModelName
by John Doe
Communication and Information
4.5
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
400 uses
ModelName
by John Doe
Communication and Information, OLTP, Statistics, Performance
4.5
99.9%
20 ms
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
400 uses 115 runs
ModelName
by John Doe
Communication and Information
4.5
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
400 uses
ModelName
by John Doe
OLTP, Communication and Information, Performance
4.5
99.9%
20 ms
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
use
run
ModelName
by John Doe
Communication and Information
4.5
99.9%
20 ms
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
400 uses 115 runs
ModelName
by John Doe
OLTP, Communication and Information, Performance
4.5
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
400 uses
ModelName
by John Doe
OLTP, Communication and Information, Performance
4.5
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
400 uses
ModelName
by John Doe
Communication and Information, OLTP, Statistics, Performance
4.5
99.9%
20 ms
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
400 uses 115 runs
ModelName
by John Doe
Communication and Information
4.5
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
400 uses
ModelName
by John Doe
OLTP, Communication and Information, Performance
4.5
99.9%
20 ms
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
use
run
ModelName
by John Doe
OLTP, Communication and Information, Performance
4.5
99.9%
20 ms
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
400 uses 115 runs
ModelName
by John Doe
Communication and Information, OLTP, Statistics, Performance
4.5
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
400 uses
ModelName
by John Doe
Communication and Information
4.5
SPECjEnterprise2010, Apache, MySQL, component 1, test component 2
400 uses
Pages: 1 2 3 4 5
Next
Models & Components List
SpecjModel
Editor ▸
COMMUNICATION AND INFOR...
Clear ×
Voice
Text / Chat
3
Email
Web browsers
Personal Information management
File / document sharing
FILTERS
Clouds
2
☑ All
☐ Amazon
☐ Google
☐ Other
» View all Clouds
Status
☑ Deployed
☑ Undeployed
Uptime
Percentage
%
to
%
Latency
ms ▾
ms
to
ms
Cost
euro ▾
€
to
€
Troughput
bytes/sec ▾
Spatial Locality
☑ All
☐ North America
☐ South America
☐ Europe
☐ Asia
Tags
☑ All
☐ Spec
☐ Tag 1
☐ Tag 2
☐ Tag 3
☐ Tag 4
☐ Other
» View all Tags

Figure 34 - Page shown when a model category is selected

(1) The page title updates to include the selected model category

(2) Filters that can be used to narrow down the displayed models

(3) Sub-menu containing the subcategories of the previously selected category

5.2.2.3 Model Subcategories

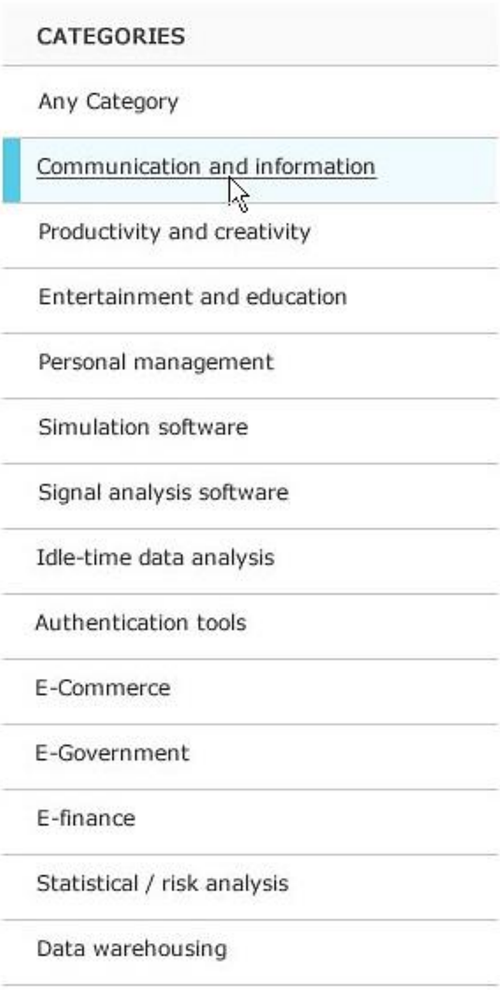


Figure 35 - Model Category Selection

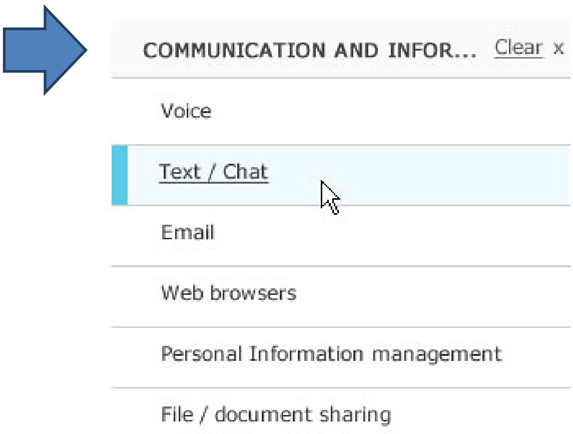
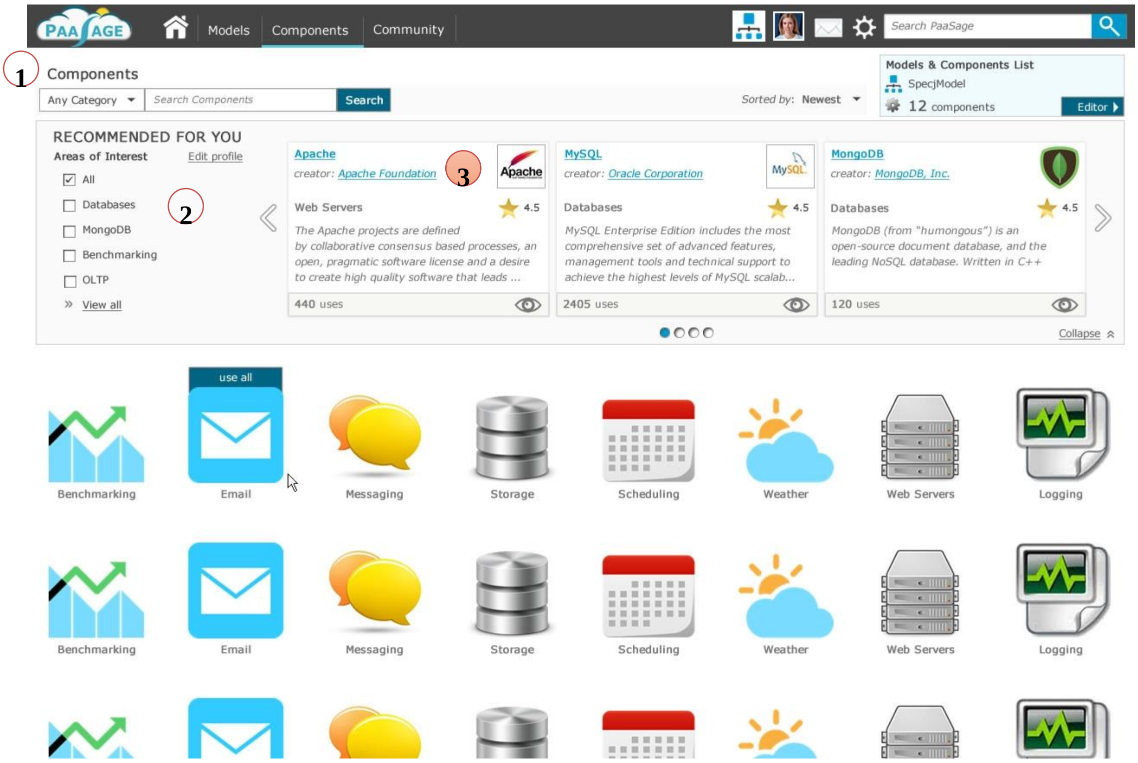


Figure 36 - Refreshed page as a result of category selection

When a model category is selected, the page refreshes to display all the models of that category, while the categories' sub-menu adapts to display the subcategories of that category. The user can undo by pressing the "clear x" control displayed next to the category title.

5.2.3 Components: Home

This screen is displayed when the “Components” option is selected.



The Apache projects are defined by collaborative consensus based processes, an open, pragmatic software license and a desire to create high quality software that leads ...

MySQL Enterprise Edition includes the most comprehensive set of advanced features, management tools and technical support to achieve the highest levels of MySQL scalab...

MongoDB (from "humongous") is an open-source document database, and the leading NoSQL database. Written in C++

Figure 37 - Home page for component exploration in PaaSage's SN

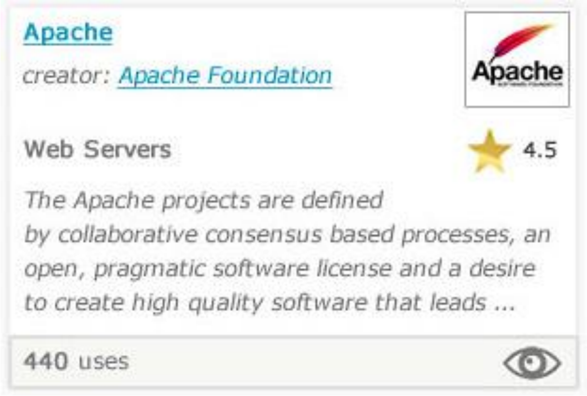
(1) Recommended components based on the user’s areas of interest

(2) Components categories; the user can either select a category to view all the contained components or click on the “use all” button that appears on mouse-over to add an abstract component to the “Models and Components List” that can be specified later in the “Model Editor” (benefiting from the Knowledge Base recommendation facilities)

(3) Component Short View

5.2.3.1 Component Short View

Component short views contain the most important information of a specific component to enable users to identify those that better suit their needs:



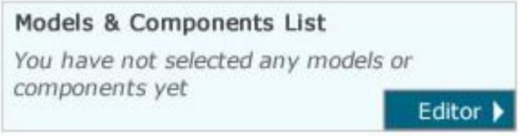
The Apache projects are defined by collaborative consensus based processes, an open, pragmatic software license and a desire to create high quality software that leads ...

- Component Icon, Title and Creator
- Associated Category
- Small Description
- User Rating
- Number of uses

Figure 38 - Short view of components in the SN

On mouse-over, the “number of uses” label transforms to a “use” button for immediate addition of the component to the “Models and Components List”

5.2.3.2 Models and Components List



You have not selected any models or components yet

Figure 39 - Empty models & component list

As soon as the user adds a model to the list, it updates to display the contained model. If the contained models are more than one, a summary is provided instead (e.g., 3 models).

Similarly, if the list contains components, the user is informed appropriately.

When empty, the “Models and Components List” informs the user that it contains no items.

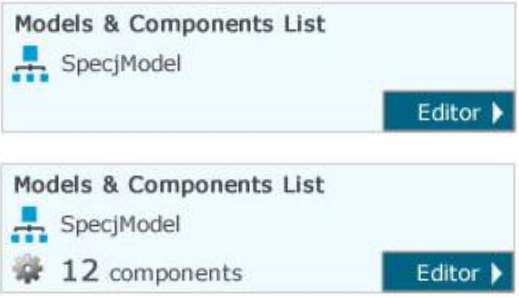


Figure 40 - Updated list with the incorporation of a model and particular components

5.3 *Identity Management for Social Network Prototype*

The implementation of the identity management infrastructure to support trust, security and privacy in PaaSage is designed to scale in order to support deployment in various domains. The Identity Management Framework supports SAML, OAuth and OpenID for authentication requests. SAML is typically used for enterprise level authentication while sources such as OpenID and OAuth are used on popular Internet Service Providers such as Google or Facebook.

The initial implementation of security privacy and trust in the MDDB is focused around the integration of the Elgg social network with the MDDB. Support for OpenID, OAuth and SAML is added to the network. This functionality has been added using Open Source plugins (SAML http://www.yaco.es/uniquid/, OAuth http://community.elgg.org/plugins/385119/0.3.1/oauth and OpenID http://community.elgg.org/plugins/433999/1.3/openid-client).

In order to process the authentication requests from the social network the OAuth and OpenID assertions are integrated within Elgg and details of the users authentication is stored in the Elgg database linked to the MDDB. For SAML, SimpleSAMLphp (http://simplesamlphp.org) is used to handle the SAML requests and act as a WAIF server. Once the SAML assertions are de-serialized, the identity information like in the Elgg plugins is stored into the MDDB.

Data protection in PaaSage is implemented using ZXID (www.zxid.org) which is an Open Source Identity Management Platform. The use of ZXID enables the wrapping of data objects in order to check identity tokens when incoming requests for access to data is received. ZXID is the mechanism by which the calls to the PEP and PDP are conducted. The PDP is implemented using OpenAZ in order to read stored XACML policies.

Once authenticated the identity assertions stored in the MDDB are checked against a lookup table in the MDDB which lists levels of trust of organizations and users. In cases where organizations have users with the same role but different privileges we provide mappings. The mapping of identity has been added with the integration of CERIF. This DSL in PaaSage can be used to express roles and hierarchy's in organisations. CERIF is used to express the relationship between roles in the specific XACML policies and the authenticated user's token.

For example an organization using PaaSage could be a US university where professor has the same functionality as lecturer in a UK university which is presenting the data protected by a XACML policy. In this case the use of CERIF mapping between the organization will ensure the users from the US organisation using PaaSage are assigned the right privileges.

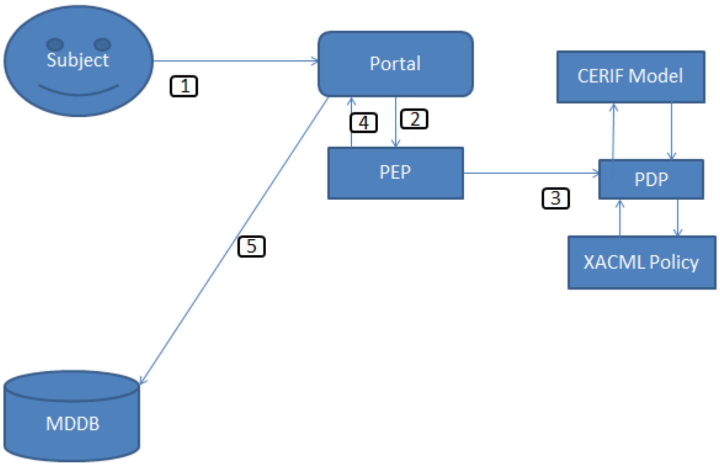


Figure 41 - XACML and CERIF integration.

Figure 41 illustrates the identity management architecture in PaaSage. By considering this diagram, the following main events can actually occur:

1 User authenticates with portal providing request

2 Portal checks userID against policy enforcement point for request

3 Policy decision made by policy decision point

4 Decision fed back to portal

5 If access granted request sent to MDDB

In its current state the main implementation using Elgg demonstrates the support of multi-domain identity management schema. The link between identity and trust is done using the MDDB schema. In the future the schema of the MDDB could be developed to better match standards for expressing trust such as WS-Trust that could be supported in future PaaSage as a DSL if WS-Security is supported.

6 Prototype evaluation

6.1 MDDB performance

To assess MDDB's performance, we have mapped its schema in a MySQL v5.5 database. The following subsections focus on two main areas: the space evolution of the database when more application data are stored over time and its response time while increasing the load (the number of concurrent clients querying the database).

Space evolution

The MDDB schema was designed to avoid redundancy when recording time-evolving state. They key to achieving this is to explicitly represent time-dependent associations, for example that of a software component with the resources used to deploy it in each execution of the application. If we recorded the time of deployment of every software component within the *software_component_instance* table, we would need to create a new software component instance for every execution of the application even though no other aspect changed across executions. This design ensures that our metadata database grows at the reasonable pace.

To get a feeling of the rate of growth for the physical size of the MDDB we assume that we have applications consisting of 3 software components each. We deploy and execute each application using random VMs and random monitoring data. Figure 42 shows the growth of the database as we store more applications in the MDDB.

The size of the database when data for only one application are stored is about 2.9 MB, while for 100 applications the size raises up only until 3.03 MB. It must be highlighted that due to the correct MDDB design, even when data for 1 million applications are stored, the MDDB size is about 3.5 GB.

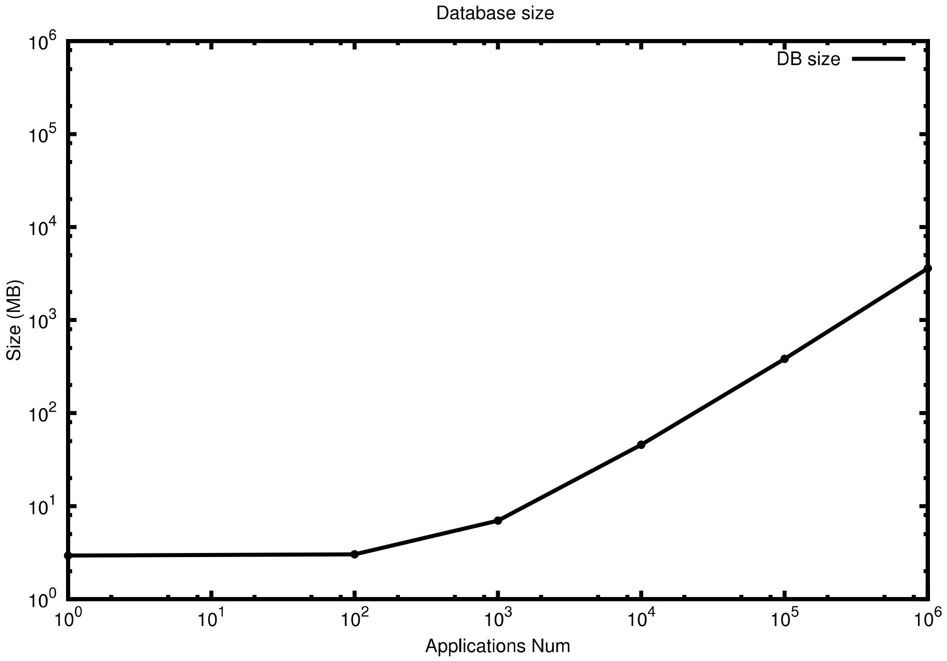


Figure 42 - Database size evaluation results

Load performance

On the second part of the performance evaluation of the MDDB, we focus on the response time of query evaluation by executing two different queries. The first query is as follows:

```
SELECT SQL_NO_CACHE d.component_instance, d.on_vm_instance
FROM execution_context as exec, slo_assessment as slo,
deployment_execution as de, deployment as d
WHERE slo.assessment=true AND
slo.execution_context=exec.id AND
de.execution_context=exec.id AND d.id=de.execution_context
LIMIT 10000
```

This query returns the mapping between the VM instances and respective software component instances. The MDDB contains sample data of 100 000 applications. As the query returns results for all the applications in the database and involves joining 4 tables which are highly populated, it can be considered as mapping to an extreme case. To avoid the overhead of sending too many results to the clients, we used the LIMIT keyword to retrieve the first 10 000 records in order. Figure 46 depicts the response time of the query as the load of the database increases (more concurrent clients query the MDDB). Even in the extreme case of about 600 concurrent clients, the response time is about 4 seconds. Please note that we don't make use of the query cache offered by the MySQL database by using the keyword SQL_NO_CACHE in the query.

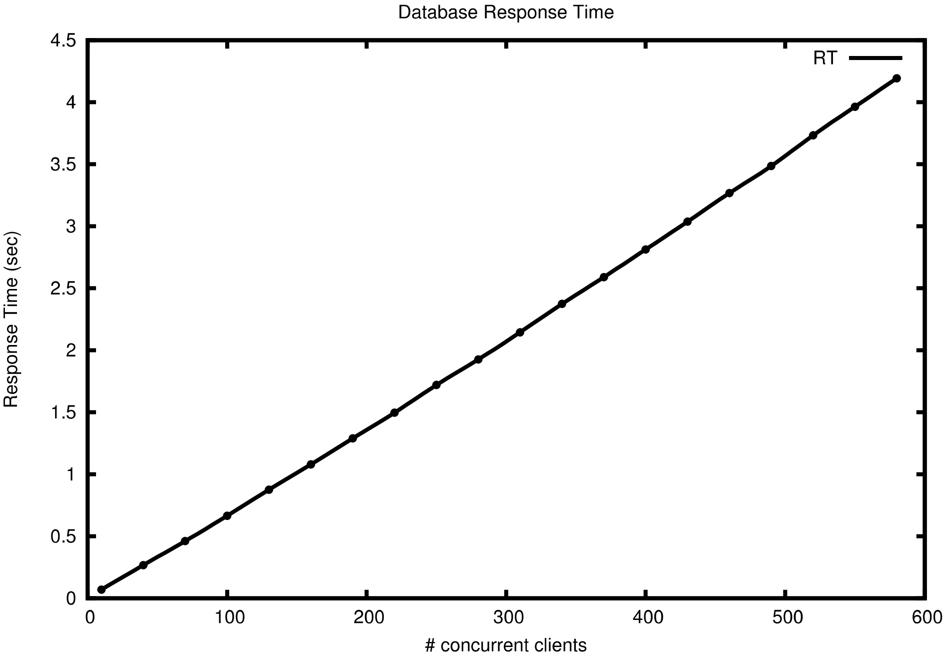


Figure 43 - Response time of MDDB while increasing the load (num of concurrent clients)

The second query (see below) exploited database performance evaluation involved the retrieval of the VM type most widely used (i.e., mostly instantiated in application deployments). Figure 44 depicts the response time of the database when its load is increased. We can again see that the performance of the database is affected by the load. However, even in the extreme case of about 600 concurrent clients, the response time is reasonable (about 6 seconds).

```
SELECT SQL_NO_CACHE vmi.cd_vm_id, count(vmi.id)
FROM vm_instance as vmi
GROUP BY vmi.cd_vm_id
```

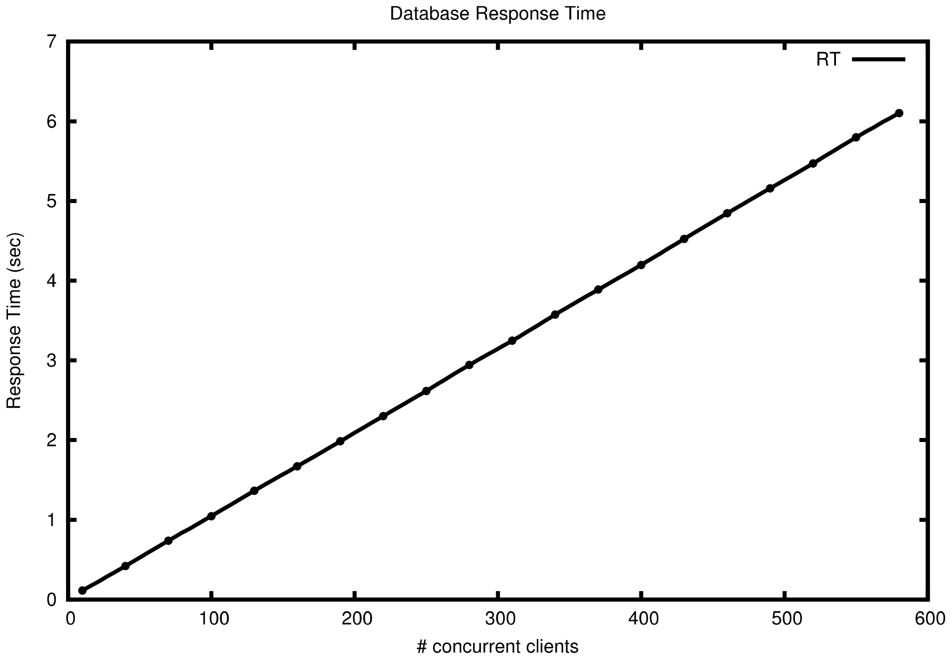


Figure 44 - Response time of MDDB while increasing the load (num of concurrent clients)

6.2 MDDB – Hibernate-Based Object-Relational interface

The main purpose of this section was to evaluate how well the MDDB fit the needs of its primary clients, i.e., the PaaSage components and modules and especially the Profiler and Reasoner. By assuming that an object interface will be offered to these clients through which they will be able to interact with the MDDB, the evaluation focused on the query performance of the MDDB through the hibernate-based object interface already realized. The main rationale for this was to have a indication of the query performance which could be slightly worsen when a different mapping from the realized one-to-one would be implemented.

Three main experiments were conducted to assess the hibernate object interface query performance which involved three main queries, respectively:

1. obtain all successful deployments for all applications
2. same as previous one but with a limit on the results (1000 rows)
3. retrieve the most widely used VM type for a cloud provider

These three queries were similar to those used for the evaluation of the pure, SQL-based performance of the MDDB. In this way, we are able to show what is the overhead introduced by hibernate and whether its scalability is appropriate. Moreover, we should also highlight the rationale for each query. The first query is quite involved as it includes joining many tables which are populated by a huge amount of rows and obtaining back the results. This query maps to a worst case scenario for queries that are not very selective and require a huge amount of data to be returned. This scenario has of course a great impact over the hibernate realization as the huge amount of data returned needs to be transformed into a graph of highly-connected domain objects. Thus, in contrast to the case of pure SQL queries, it is expected that the overhead introduced through performing this transformation will be quite high. On the other hand, the second query exactly shows that when the amount of information to be returned is not very big, then the performance is better and more appropriate.

The goal of the third query was to introduce a scenario of a selective query issuance which involves comparing aggregations of MDDB data. Thus, now it is not the amount of data to be returned that play a significant role in query time but actually the query provisioning. In this way, we are able to show that for this type of scenarios, the hibernate-based performance is quite close to the one exhibited by pure SQL queries.

The experiments, as in the previous MDDB evaluation, were performed over a MySQL DB running on a Flexiant FCO instance and involved the issuing of HQL queries by an increasing number of concurrent users. For each experiment, the average query time across each number of specific concurrent users was assessed.

First query experiment - Simple querying with huge amount of results

In the first experiment, we increased the number of concurrent users from 10 to 40 with a step of 10 and evaluated the average query performance of the hibernate-based solution. The experiment results are shown in Figure 45. As it can be seen, while there is a linear increase in the query time with the increase in the number of concurrent users, which signifies the scalability of the examined solution, the query time is not very satisfactory. This dictates that for the particular type of queries the hibernate solution can only be used by (PaaSage) components/modules which do not perform a real-time task which requires a fast query performance. Moreover, this query type should be avoided if there is the possibility of a great number of concurrent issuers.

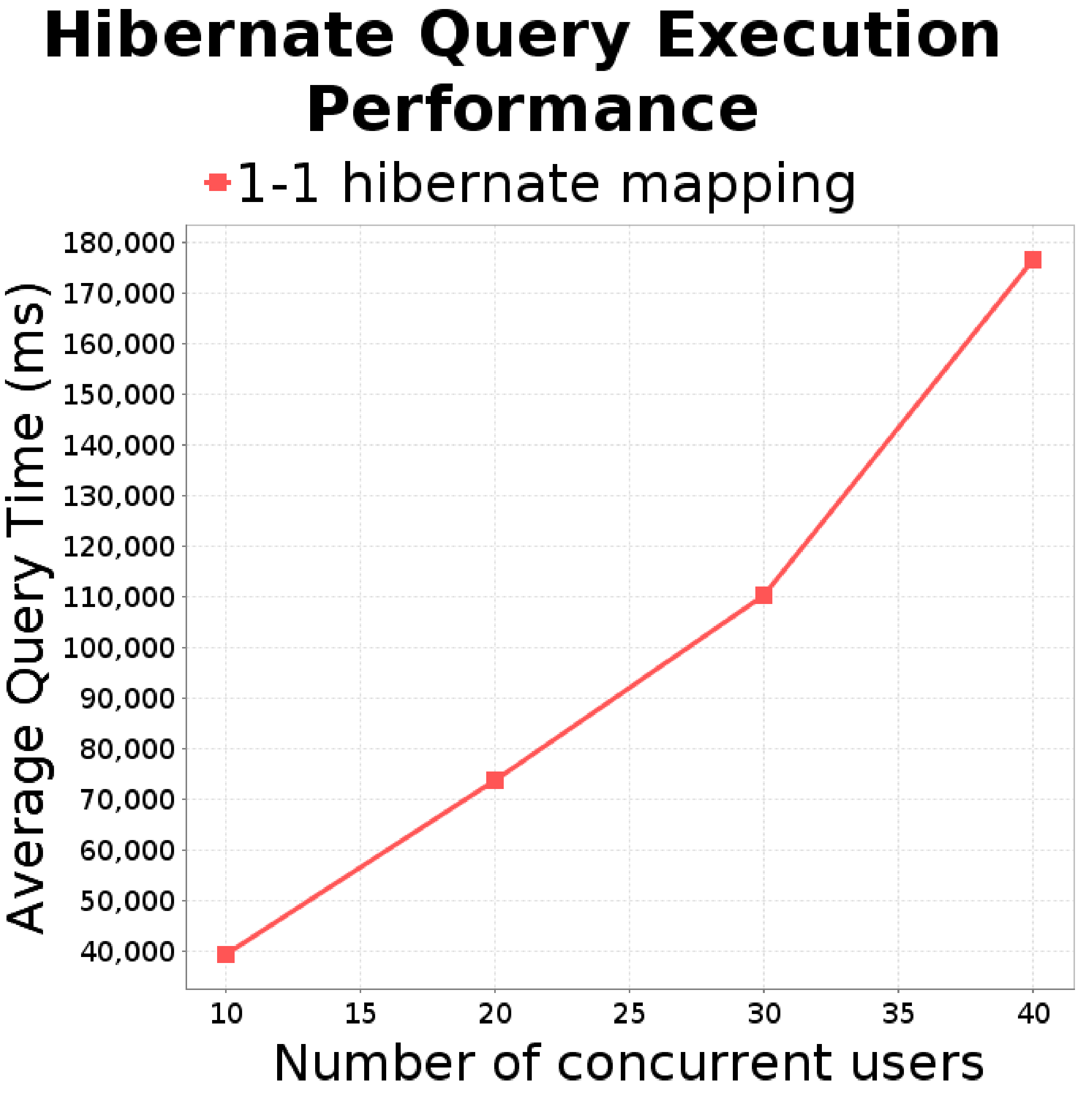


Figure 45 - The first query performance of the 1-1 hibernate mapping solution

Second query experiment - Simple querying with small amount of results

The second experiment involved the issuing of the second query over an increasing number of concurrent users from 100 to 600 with a increase step of 100. The experiment results are shown in Figure 46. By comparing with the previous assessment results, we can clearly deduce that the performance in this case is much better and a greater number of concurrent users can be supported. Thus, the hibernate solution is quite scalable in this case and can be used by component tasks that need a fast query response time. These results also prove our assumption that the overhead of a great amount of query results significantly impacts the performance and scalability of the hibernate-based solution. Please bare in mind that again the performance trend is the same with a linear increase in query time with the increase of the concurrent users.

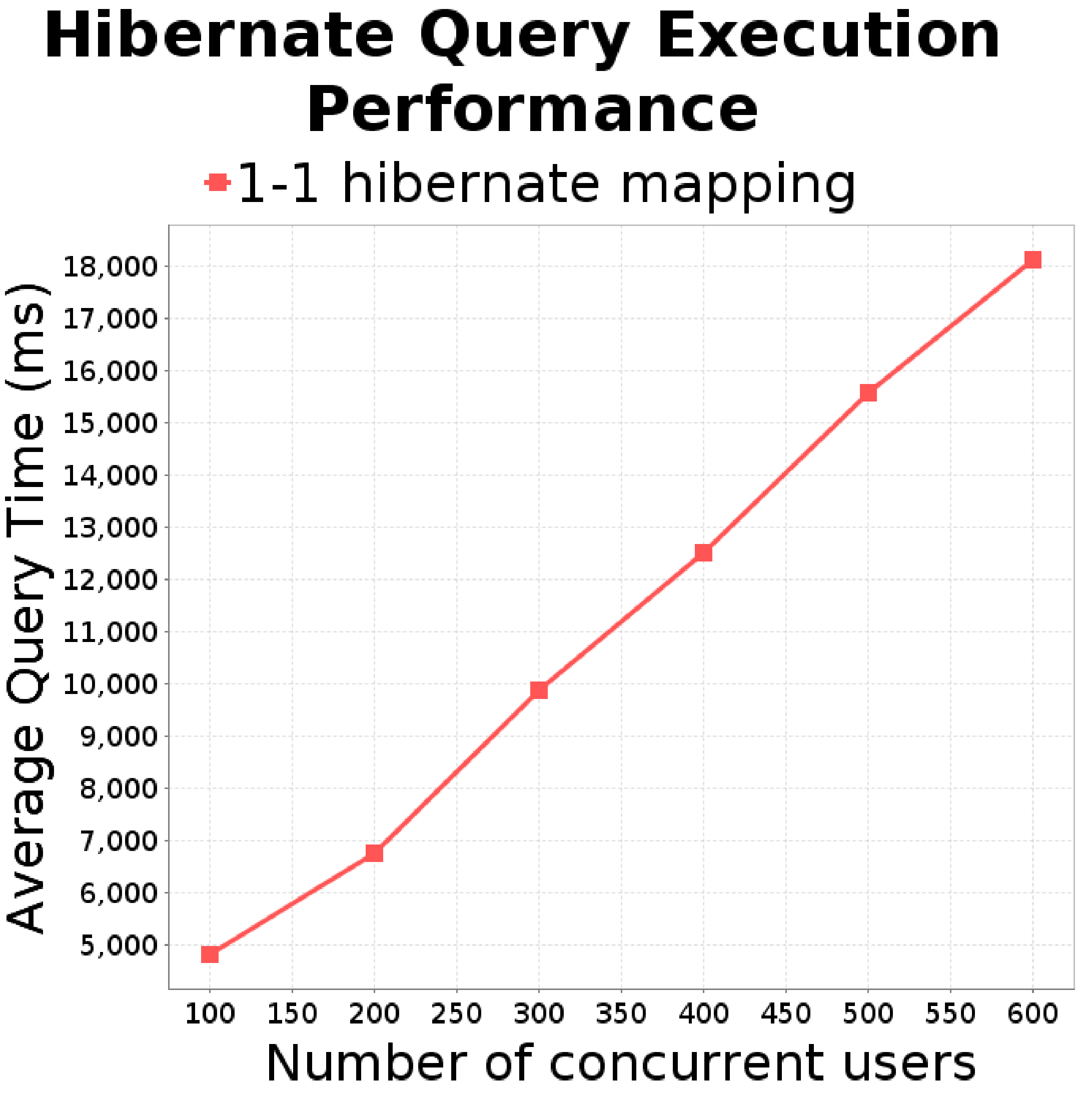


Figure 46 - The second query performance for the 1-1 hibernate mapping solution

Third query experiment - Involved querying with small amount of results

In the third experiment, the number of concurrent users issuing the third query increased from 50 to 300 with a step of 50. The respective experiment results are depicted in Figure 47. Again, there is a linear increase in the performance behavior. In addition, by comparing these results with the respective ones according to the pure-based SQL evaluation, we can clearly see that for this type of queries the object transformation overhead is small and the major query time percentage is spent in query provisioning (by the underlying DBMS). To this end, this type of queries seem to be ideal for being used under this specific solution. In fact, as we expect that this type of queries will be the ones most widely used in the context of the PaaSage project, then it can be definitely deduced that the hibernate-based solution will be the ideal alternative for the realization of the envisioned object interface. What remains to be shown, after a particular non straightforward DSL-to-MDDB mapping is realized, is whether the non one-to-one mapping can increase the overhead introduced by hibernate and in what degree. If the overhead is insignificant, then the hibernate-based solution will be the final realization choice of the object-to-relational interface.

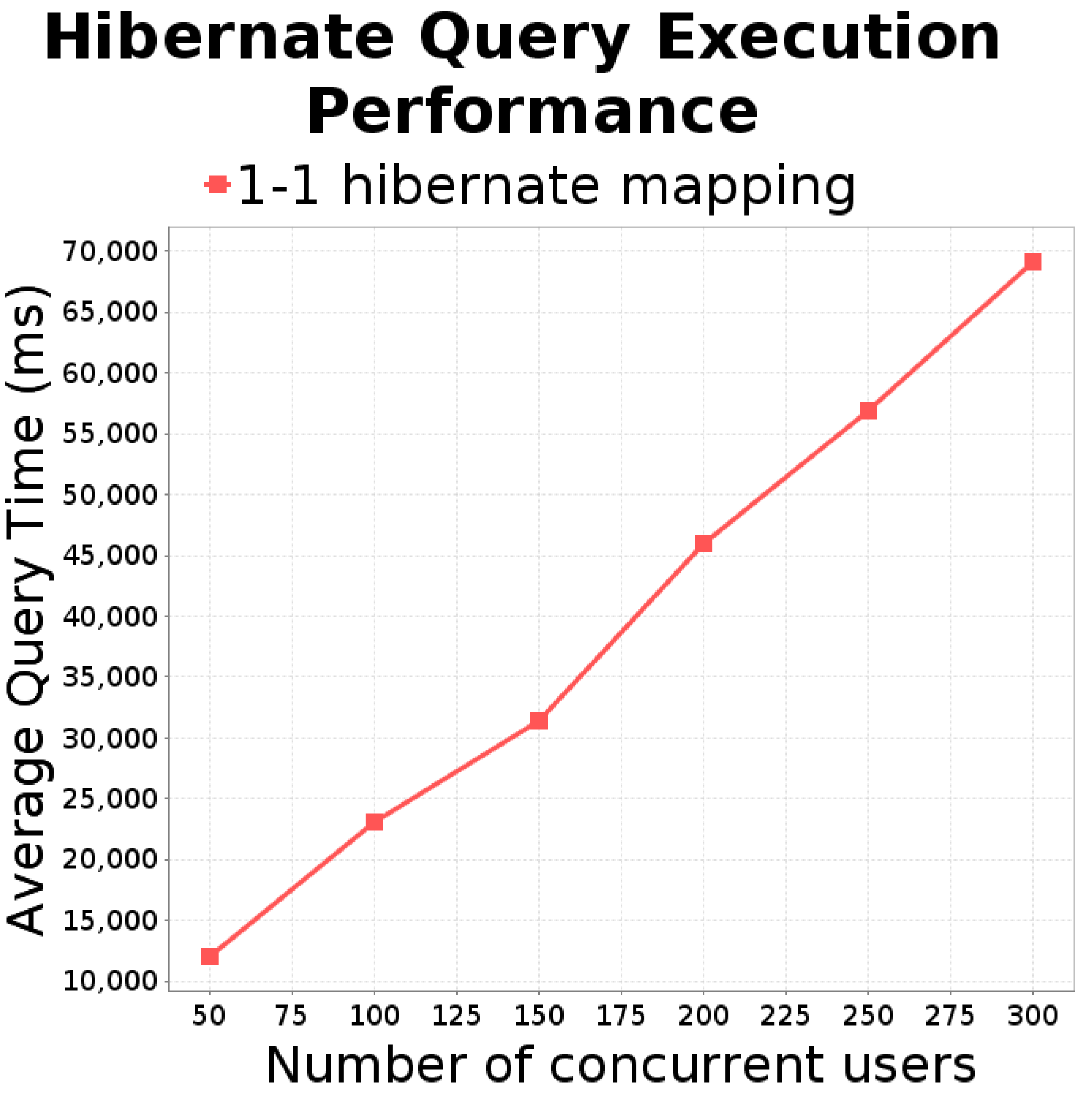


Figure 47 - The third query performance of the 1-1 hibernate mapping solution

6.3 KB Performance

The purpose of the evaluation was to assess whether the Rest-Based implementation of the KB Management Service is scalable enough to accommodate for an increasing number of concurrent users no matter what is the size of the user session repository storing the respective objects derived and has a quite good runtime performance. The main focus of the evaluation was a particular method which is the one most valuable to the PaaSage users, components and modules and which is expected to be executed most of the times by them. This method is obviously the one dedicated to the issuing of queries on user sessions for obtaining particular facts derived which is called *runQuery*. Other methods could also be tested but we restrained our focus just on this method as: (a) some other methods are quite lightweight with respect to their realization, so it is not quite meaningful to assess them, (b) just one method, called *fireRules*, needs to be run sparsely across user sessions as: (i) it generally takes some time to finish especially for quite populated underlying (MDDB) database(s) and (ii) a slight update on the database(s) is highly improbable to require the updating of derivations for old or the derivation of new facts; thus, if this method is called once per day per PaaSage component/module, then it is quite easy to handle it in an appropriate and scalable way.

Based on the above analysis, we made two stress tests on the rest-based implementation to check the scalability and performance of the *runQuery* method for varying sizes of the underlying MDDB: (a) the first focused on issuing different types of queries with different content over an underlying sparsely populated MDDB and (b) the second focused on issuing the same types of queries over a highly populated MDDB. The varying content and type of queries was used to simulate the expected situation where each user may issue different query types (based also on his/her interests) with varying content (mapping to different checks such as the case of the SN which could issue many queries requiring to find which application components are similar to those exploited in the current modelling of a particular application) as well as examine whether performance is highly affected by the size of the query results to be returned (as time is needed to serialize the results in XML and deliver them across the network) such that the caching advantages of the KB session are diminished (where same query with same content will be responded quicker in all times than the first). The four query types map to the four main KB rules: (a) find similar applications for a specific application, (b) find similar artifacts for a specific artifact, (c) retrieve best deployment for a particular application and specific set of SLOs and (d) obtain best deployment for a particular artifact and a specific set of SLOs.

In the first test, MDDB was holding execution data for 5 applications concerning 4 execution contexts, while for the second test the same amount of applications remained but the execution context data for each become 400.

The tool used to run the stress tests was JMeter and all the tests were performed from a local laptop machine (i.e., the stress code - all thread clients issuing HTTP requests to the REST-service) to the Flexiant Cloud (FCO) where the service was deployed and running. In each experiment test, 100 concurrent users were created issuing 100 queries each in one second interval between requests. For issuing his/her requests, each user first created a session (not counted for computing the overall time) and then called the *runQuery* method 100 times before destroying his/session in the end (again not counted for computing the overall time). The query type and parameters are chosen randomly for each call of the *runQuery* method per user.

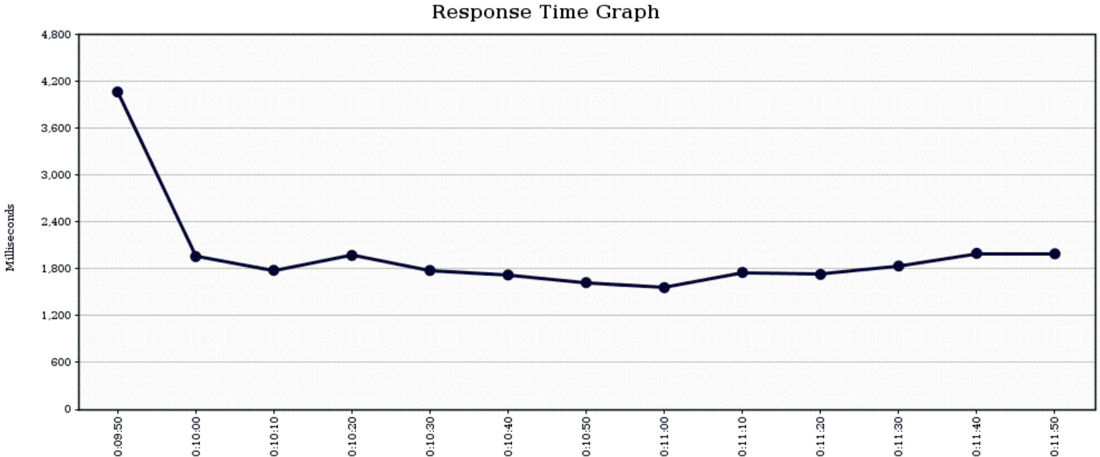


Figure 48 - The response time behavior of KB for the first assessment

6.3.1 Results for First Test - Sparsely Populated MDDB

The assessment results from the first stress test are depicted in Figure 48. As it can be seen, the query time in the beginning is increased but then it drops sharply and gets stabilized until the very end. This behaviour can be justified by the fact that at the beginning most of the queries as well as their content tends to be new so they have to be normally processed. Then, as the query results start to get cached, the performance starts to increase and becomes stable. Thus, for this case, the API method is scalable enough to handle a vast amount of users with the appropriate performance where the response time goes from 420 milliseconds to 1.4 seconds.

6.3.2 Results for Second Test - Highly Populated MDDB

Figure 49 shows the assessment results for the second experiment. The performance seems to follow the same trend with respect to the previous assessment results. However, we can deduce the following differences:

- the query performance is not so stable as in the first experiment.
- The query performance is worse than the first experiment.

These differences can be justified as follows. In order to serialize the query results and pass them over the network, we followed a serialization strategy where as much as possible information from the domain objects is retained. In this way and as the domain objects tend to be highly connected with each other, the same queries with almost the same type of results have now a quite larger size than in the first experiment. For instance, by considering the first column for the best deployment query which maps to an application object, in the first case this object was connected to 4 execution context objects which were inserted internally in its serialization (when this serialization was produced), while in the second case the same object was associated with 400 execution context objects. To this end, the following effects were produced: (a) the different types of queries now map to different result sizes; as such, when more queries from the small size cluster are finished, the average query time is reduced, while, on the other hand, when more queries from the big size cluster end, the average query time is increased; (b) as the result size is very big (MBs of data) for half of all the issued queries and such big size maps to a great XML serialization and network broadcasting time, the performance is now much worse.

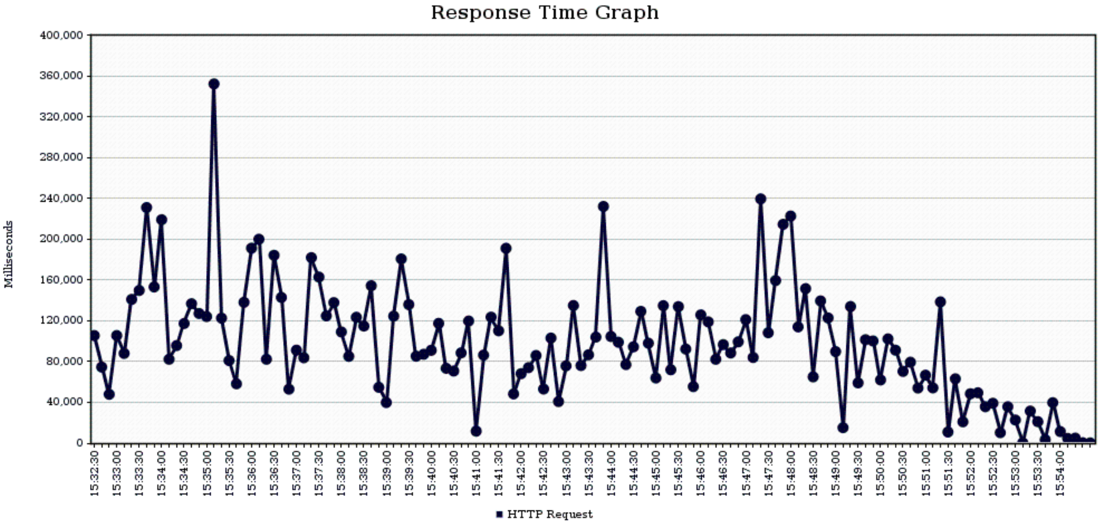


Figure 49 - The KB query performance for the second experiment

The second experiment's assessment results clearly show that while the response behaviour trend is quite good and more or less stable, the selected serialization strategy leads to a big performance penalty induced by serializing and sending results with big size. In this way, a particular solution is chosen to remedy the above issue and improve KB performance which leads to redesigning the main KB rules and its domain model (mapping to the additional (with respect to MDDB) facts that are derived): the serialization strategy should now be minimal with respect to the information considered from the main MDDB objects. Obviously, this solution will certainly lead to performing additional queries for obtaining the information that has been cut out from the serialization but first here we are dealing with a major design trade-off and second queries can be focused and be performed only on those objects that interest the PaaSage user.

As a proof of concept for the new, envisioned solution, we have modified the serialization for some of the domain objects that are returned and re-executed the second experiment. The results can now be seen in Figure 50. As it can be seen, the query performance has been substantially improved such that 100 concurrent users obtain the respective query results in 4 seconds on average. This performance is obviously quite satisfactory and enables the use of the KB by PaaSage modules and components which might have real-time or near real-time tasks with increasing demands on query latency.

Based on the above rationale and analysis, the KB will be redesigned according to the solution considered so as to improve its query performance substantially and be able to handle a great number of concurrent users with the best possible query response time.

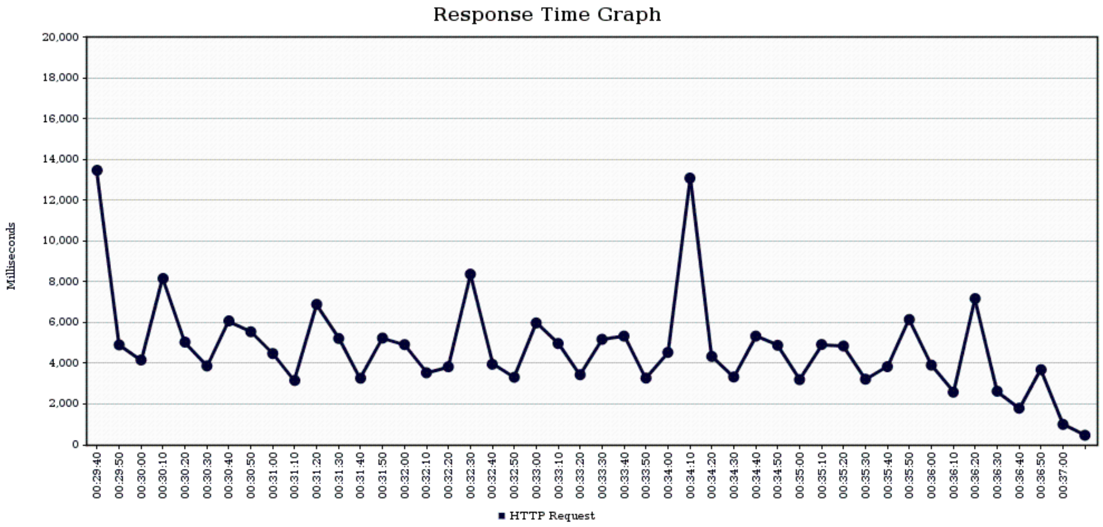


Figure 50 - The query performance of the modified KB for the second experiment

7 Path to Final Prototype (M36)

The current prototype implements a significant amount of the MDDB functionality. Considerable effort however is still needed towards the production of the final prototype in M36, as Table X also shows. Besides progress with individual components, the planned work includes focus on the integration between different components ensuring that they can correctly communicate and interoperate. To coordinate this work, FORTH will continue organizing regular bi-weekly teleconferences. Progress so far has finalized the responsibilities for different tasks; we thus foresee a smooth path to the production of a fully-functional prototype and will now focus on the remaining work.

Work Item Number	Work Item Description	Responsible Partner Organization
#1	DSL-to-MDDB mapping realization	FORTH
#2	Modelling of concepts and relationships not covered by DSLs for extensive ecore meta-model	FORTH
#3	Linking concepts in extensive ecore model	STFC
#4	CAMEL (ontology-based) modelling & mapping	STFC
#5	Further DSL modelling + possible modelling of new DSLs	FORTH, CETIC, AGH (+UULM)
#6	Improve CERIF integration with MDDB	AGH
#7	Additional information sources exploitation	AGH
#8	Scalability and high-availability of MDDB through CDO	FORTH

#9	Small KB redesign and possible KB service API extension	FORTH
#10	Analytics module development	FORTH
#11	SN's UI realization	FORTH
#12	SN's realization of missing functionality	FORTH
#13	SN's (UI) evaluation	Flexiant (+ possibly other partners)
#14	Multi-domain security	STFC & AGH
#15	Further MDDB/KB evaluation	FORTH (+ possibly other partners)

Table 4 - Work Item Summary towards M36

Concerning the MDDB itself, the work can be split into three main directions; two of those directions are joined based on the underlying technology. The first direction concerns the mapping of the DSLs to the MDDB schema. Based on the decisions made in a recent meeting (3/2014) the consortium is considering exploiting Eclipse CDO as the technology for realizing the object-to-relational interface required by the PaaSage components (especially those involved in WP3). This technology (thoroughly analyzed in Deliverable D2.1.2 [D2.1.2]) exploits Teneo and Hibernate (see Section 3.1.8) in order to provide a persistence layer for the storage, updating and retrieving models complying to DSLs. Moreover, through this technology, JPA and Hibernate annotations can be provided which can define in most of the cases (with just the exception of Saloon DSL) all of the identified mappings between the DSLs and the MDDB. In addition, based on the previous decisions, it has been decided to create an extensive *ecore* model which not only covers all the DSL but also the information already captured in the MDDB schema. This means that there will be the need of enriching the *ecore* model with all the appropriate concepts and relationships in such way that not only the additional, required information is covered but also that the mapping is more or less one-to-one in order not create an additional load to the persistence layer in the enforcement of the mappings. This will be a task that will be followed by FORTH in cooperation with the partners that have contributed to this additional information (AGH for CERIF-based modelling, CETIC for the security DSL, and STFC for the WS-Agreement modelling). It should be noted that through this approach, the current MDDB schema will not be modified to a significant degree as the mappings will ensure that the respective tables and columns with their corresponding names and types are still there in most of the cases. However, some of the table information might be enriched by columns additionally inserted by Hibernate.

Apart from modelling the additional missing information from the union of the DSLs, there is also a need for linking different concepts with each other, especially when they are equivalent or there is some interconnection between them. This task will be performed by STFC which will explore first and then enforce some of the appropriate linking in the extensive *ecore* model and then evaluate whether the *ecore*-based technology along with CDO is enough for having the appropriate expressivity and whether an ontology at a higher level will be more appropriate to ensure the integration of all information. This task will be performed in such way so as not to interfere with the CDO solution and the way it is mapped to the MDDB but on the other hand ensure that the ontology-based (CAMEL) models will also be mapped accordingly to MDDB without any loss of (critical) information.

The further modelling of some novel DSLs will also be pursued, such as those of the security and scalability rule DSLs. For the first DSL, FORTH will cooperate with CETIC in order to evaluate whether the current modelling is enough or needs to be extended appropriately. For the second DSL, FORTH will cooperate with UULM in order to further extend the DSL according to particular aspects that have already been discussed during the development of the current, first version of the DSL. Apart from this, it might also be interesting to see whether other additional DSLs could be modelled not just to add a new DSL but to cover a particular need that might originate from the use cases of the PaaSage project. For instance, we could examine whether a new DSL could be created in order to replace WS-Agreement, which although is a standard, seems not to cover some essential aspects, such as the definition of the SLOs and of the respective metrics involved.

AGH plans to extend the current MDDB schema in order to include additional concepts from CERIF and cover cases requiring additional modelling of roles and organizations, the trust and reputation aspects, the accounting and billing for the users, and the tracking of provenance and usage statistics. By considering that CERIF is another DSL, then have certainly the case of extending a DSL in order to cover some additional aspects which will provide an added-value to the project and enable to more fully realize the role of a multi-cloud service management platform. The future work will also focus on further improving CERIF integration with MDDB, including the possibility of uploading CERIF descriptions of users or organizations in CERIF XML exchange format directly through the social network website. Additional information sources will be also considered for improving MDDB functionality such as automatic creation of CloudML configuration models from legacy application descriptions (e.g. JBoss application descriptors) and support for advanced methods of rule discovery in Knowledge Base.

Based on the CDO technology adopted, the way scalability and high-availability for the MDDB can be achieved might be modified as CDO already provides mechanisms which can enforce (DB) distribution with interesting features, such as fault-tolerance and data consistency (see Deliverable D2.1.2 [D2.1.2]). FORTH will be responsible of checking the CDO distribution capabilities.

As far as the KB is concerned, the evaluation in Section 6 has shown that it should be re-designed in order to have an appropriate domain model which does not lead to delays in answering queries which involve a huge amount of results. Apart from this, some of the KB rules will be enhanced in order to better fulfil their purpose. In particular, the rules used for inferring application and component equivalence/similarity will be extended to consider other (richer) information aspects apart just from the name of the components and applications. In addition, additional rules might be created which can be used to derive some other interesting facts, such as which adaptation actions are the best to consider for adapting a particular application when specific performance problems arise. Based on the feedback from the PaaSage developers, the API that has been implemented might be also enriched in order to provide additional functionality or slightly change the way the current functionality is delivered. All of the tasks that involve the KB will be conducted by FORTH.

As it can already been figured out, the Analytics Module has not been developed yet. Its development will start after M18 and will finish before M36. However, it is acknowledged that this component will provide added value to the MDDB prototype and in general to the PaaSage platform, through analyzing the previous execution history and providing interesting insights on the application performance and deployment, and thus the consortium will surely focus on its proper realization on time.

The Social Network is being re-designed in order to reach further usability levels as well as become the single point of interaction between the users and the platform. To this end, while the back-end technology remains the same, the UI is being improved as outlined in the mock-ups presented in this deliverable. Once the UI design is finalized, the UI will be realized based on particular state-of-the-art methods as well as the back-end possibly enriched based on the requirements imposed by the UI. Part of the work concerning the UI also includes the generation of an editor which will be used by the PaaSage users in order to create CAMEL models which will then be used to guide the provisioning of the respective applications. While a significant part of this work is driven by FORTH, guidance and feedback on the usability of the UI will be performed by other partners to provide insights on aspects of the UI that need further improvement.

While implementation of the security layer for the authentication and authorization in the MDDB has been finalized, it can currently be enforced in a single domain. It will be the task of STFC and AGH to extend this layer accordingly in order to work correctly also in multiple domains (an important feature consider distribution of the MDDB) with the appropriate realization of the required functionality and the respective modelling extensions.

Finally, although some initial evaluation of MDDB has been performed, the consortium will attempt to perform further evaluation in order not only to show some additional aspects (scalability due to MDDB distribution) but also to reassess its performance and scalability due to the incorporation of new technologies (such as CDO/Teneo/Hibernate on top of RDB layer).

8 Conclusion

This deliverable reports on the current status of the MDDB and social network infrastructure. It also provides a roadmap of the remaining work to be performed in order to finalize the final prototype offering the full functionality supporting the PaaSage architecture and the use cases. Progress to date has been according to plan through the smooth cooperation of the consortium members. Given the clear implementation plan outlined in this deliverable, we are confident that development of the final prototype will also proceed smoothly through M36.

Bibliography

[CERIF] Keith G Jeffery and Anne Asserson. "CRIS Central Relating Information System". In: *Proceedings of the 8th International Conference on Current Research Information Systems (CRIS2006)*. Leuven University Press, 2006, pp. 109-120. ISBN: 978-90-5867-536-1.

[CloudML] Nicolas Ferry, Franck Chauvel, Alessandro Rossini, Brice Morin and Arnor Solberg. “Managing multi-cloud systems with CloudMF”. In: *NordiCloud 2013: 2nd Nordic Symposium on Cloud Computing and Internet Technologies*. Ed. by

Arnor Solberg, Muhammad Ali Babar, Marlon Dumas and Carlos E. Cuesta. ACM, 2013, pp. 38–45. ISBN: 978-1-4503-2307-9. DOI: 10.1145/2513534.2513542.

[Cumulus] A. Pannetrat, Security-aware SLA Specification Language and Cloud Security Dependency Model. CUMULUS Deliverable D2.1, 2013.

[D1.6.1] The PaaSage Consortium. D1.6.1—Initial Architecture Design. PaaSage project deliverable. Oct. 2013.

[D2.1.1] Alessandro Rossini, Arnor Solberg, Daniel Romero, Jörg Domaschka, Kostas Magoutis, Nicolas Ferry, Tom Kirkham, Maciej Malawski, Bartosz Balis, Dariusz Krol, Achilleas Achilleos, CloudML Guide and Assessment Report (Extended). PaaSage Deliverable D2.1.1e, November 2013.

[D2.1.2] Alessandro Rossini, Nikolay Nikolov, Daniel Romero, Jörg Domaschka, Kyriakos Kritikos, Tom Kirkham, Arnor Solberg, CloudML Implementation Documentation. PaaSage Deliverable D2.1.2, March 2014.

[D5.1.1] Anthony Sulistio, Panagiotis Garefalakis, Damianos Metalidis, Chrysostomos Zeginis, Craig Sheridan, Kuan Lu, Jörg Domaschka, Bartosz Balis, Dariusz Król, Edwin Yaqub, Prototype Executionware and Prototype new Execution Engines. PaaSage Deliverable D5.1.1, March 2014.

[Kritikos and Plexousakis 2006] K. Kritikos and D. Plexousakis. Semantic QoS Metric Matching. In ECOWS (2006).

[Kritikos et al. 2013] Kyriakos Kritikos, Barbara Pernici, Pierluigi Plebani, Cinzia Cappiello, Marco Comuzzi, Salima Benrernou, Ivona Brandic, Attila Kertész, Michael Parkin, and Manuel Carro. 2013. A survey on service quality description. *ACM Comput. Surv.* 46, 1, Article 1 (July 2013), 58 pages. DOI=10.1145/2522968.2522969 http://doi.acm.org/10.1145/2522968.2522969.

[Saloon] Clément Quinton, Nicolas Haderer, Romain Rouvoy and Laurence Duchien. “Towards multi-cloud configurations using feature models and ontologies”. In: *MultiCloud 2013: International Workshop on Multi-cloud Applications and Federated Clouds*. ACM, 2013, pp. 21–26. ISBN: 978-1-4503-2050-4. DOI: 10.1145/2462326.2462332.

[WS-Agreement] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke and Ming Xu. *Web Services Agreement Specification (WS-Agreement)*. Tech. rep., Open Grid Forum, Mar. 2007.

[XACML] Erik Rissanen. eXtensible Access Control Markup Language (XACML) Version 3.0. Technical Committee Specification 01, OASIS, August 2010.

[Zeginis et al. 2013] C. Zeginis, P. Garefalakis, K. Kritikos, K. Konsolaki, K. Magoutis and D. Plexousakis. Towards Cross-layer Monitoring of Multi-Cloud Service-based Applications, European Conference on Service-Oriented and Cloud Computing, Malaga (2013)