PaaSage

Model Based Cloud Platform Upperware

Deliverable D2.1.2

CloudML Implementation Documentation (First version)

Version: 1.0

D2.1.2

Name, title and organisation of the scientific representative of the project's coordinator:

Mr Tom Williamson Tel: +33 4 9238 5072 Fax: +33 4 92385011 E-mail: tom.williamson@ercim.eu

Project website address: http://www.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:	30th September 2013
Document	
Period covered:	
Deliverable number:	D2.1.2
Deliverable title	CloudML Implementation Documentation (First version)
Contractual Date of Delivery:	30th March 2014 (M18)
Actual Date of Delivery:	30th April 2014
Editor (s):	Alessandro Rossini
Author (s):	Alessandro Rossini, Nikolay Nikolov, Daniel Romero, Jörg Domaschka, Kiriakos Kritikos, Tom Kirkham, Arnor Solberg
Reviewer (s):	Maciej Malawski, Stefan Spahr
Participant(s):	
Work package no.:	2
Work package title:	Languages
Work package leader:	Arnor Solberg
Distribution:	
Version/Revision:	1.0
Draft/Final:	Final
Total number of pages (including cover):	63

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 28 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)

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

Contents

Executive Summary

Cloud computing provides a ubiquitous networked access to a shared and virtualised pool of computing capabilities that can be provisioned with minimal management effort. Cloud-based applications are applications that are deployed on cloud infrastructures and platforms, and delivered as services. PaaSage aims to facilitate the specification and execution of cloud-based applications by leveraging upon model-driven engineering (MDE) techniques and methods, and by exploiting multiple cloud infrastructures and platforms.

MDE is a branch of software engineering that aims at improving the productivity, quality, and cost-effectiveness of software development by shifting the paradigm from code-centric to model-centric. Models enable the abstraction from the implementation details of heterogeneous cloud services, while model transformations facilitate the automatic generation of the source code that exploits these services. This approach, which is commonly summarised as "model once, generate anywhere", is particularly relevant when it comes to the specification and execution of multi-cloud applications (*i.e.*, applications deployed across multiple cloud infrastructures and platforms), which allow exploiting the peculiarities of each cloud service and hence optimising performance, availability, and cost of the applications.

Models are frequently specified using domain-specific languages (DSLs), which are tailored to a specific domain of concern. In order to cover the necessary aspects of the specification and execution of multi-cloud applications, PaaSage encompasses a family of DSLs called Cloud Application Modelling and Execution Language (CAMEL). These DSLs, namely Cloud Modelling Language (CLOUDML), Saloon, WS-Agreement, and Scalability Rules Language (SRL), provide support for modelling multiple concerns of multi-cloud applications, such as provisioning, deployment, requirements, goals, SLAs, and execution.

In this deliverable, we provide the initial version of the technical documentation of the DSLs adopted in PaaSage. In particular, we describe the modelling concepts, their attributes and their relationships, as well as the rules for combining these concepts to specify valid models. Moreover, we exemplify how to specify models through a tree-based editor as well as how to programmatically manipulate and persist them through Java APIs.

Please note that these DSLs are under development and will evolve throughout the course of the PaaSage project. Hence, the capabilities offered by these DSLs and presented in this deliverable reflect our understanding of the requirements of PaaSage at month 18. The final version of the technical documentation of the DSLs adopted in PaaSage will be provided in D2.1.3 [23] at month 36.

Intended Audience

This deliverable is a public document intended for readers with some experience in cloud computing and software engineering, and some familiarity with the initial architecture design (*cf.* D1.6.1 [11]) as well as the DSLs adopted in PaaSage (*cf.* D2.1.1 [25]).

For the external reader, this deliverable provides the technical documentation that will facilitate adopting the DSLs outside the PaaSage platform.

For the research and industrial partners in PaaSage, this deliverable provides the technical documentation that will facilitate integrating the DSLs with the components of the PaaSage platform.

1 Introduction

MDE is a branch of software engineering that aims at improving the productivity, quality, and cost-effectiveness of software development by shifting the paradigm from code-centric to model-centric. MDE promotes the use of models and model transformations as the primary assets in software development, where they are used to specify, simulate, generate, and manage software systems. This approach is particularly relevant when it comes to the specification and execution of multi-cloud applications (*i.e.*, applications deployed across multiple private, public, or hybrid cloud infrastructures and platforms), which allow for exploiting the peculiarities of each cloud service and hence optimising performance, availability, and cost of the applications.

Models can be specified using general-purpose languages like the Unified Modeling Language (UML). However, to fully unfold the potential of MDE, models are frequently specified using domain-specific languages (DSLs), which are tailored to a specific domain of concern. In order to cover the necessary aspects of the specification and execution of multi-cloud applications, PaaSage encompasses a family of DSLs called Cloud Application Modelling and Execution Language (CAMEL). This family includes the Cloud Modelling Language (CLOUDML) [8, 9], for modelling and enacting the provisioning and deployment of multi-cloud applications; Saloon [21, 20, 19], for specifying requirements and goals of multi-cloud applications; WS-Agreement [1], for modelling SLAs of web services; and Scalability Rules Language (SRL), for specifying scalability rules.

The abstract syntax of a language describes the set of concepts, their attributes, and their relationships, as well as the rules for combining these concepts to specify valid statements that conform to this abstract syntax. The concrete syntax of a language describes the textual or graphical notation that renders these concepts, their attributes, and their relationships.

In MDE, the abstract syntax of a DSL is typically defined by its metamodel. That is, a metamodel describes the set of modelling concepts, their attributes, and their relationships, as well as the rules for combining these concepts to specify valid models that conform to the metamodel [16]. Moreover, in MDE, the concrete syntax may vary depending on the domain, *e.g.*, a DSL could provide a textual notation based on JavaScript Object Notation (JSON) as well as a graphical notation based on trees or graphs along with the corresponding serialisation in XML Metadata Interchange (XMI).

In this deliverable, we focus on the abstract syntax of the DSLs adopted in PaaSage and describe their metamodels. Moreover, we exemplify how to specify models using a tree-based editor as well as how to programmatically manipulate and persist them through Java APIs.

Structure of the document

The remainder of the document is organised as follows. Section 2 presents some technologies facilitating the integration of the DSLs adopted in PaaSage. Sections 3, 4, 5, and 6 present the metamodels of CLOUDML, Saloon, WS-Agreement, and SRL, respectively, along with corresponding sample models. Section 7 exemplifies the usage of Java APIs to programmatically manipulate and persist models. Section 8 summarises the approach used to map the metamodels of the DSLs to the Metadata Database schema. Finally, Section 9 compares the proposed approach with related work, while Section 10 draws some conclusions and outlines some plans for future work.

2 CAMEL

CAMEL is a family of DSLs to specify models in all life-cycle phases of configuration, deployment, and execution of multi-cloud applications in PaaSage (*cf.* D2.1.1 [25]). In order to achieve the integration of these DSLs, we adopt the Eclipse Modelling Framework (EMF)[1] on top of the Connected Data Objects (CDO)[2] persistence solution. In this section, we outline EMF and CDO and discuss how they facilitate the integration between the DSLs.

2.1 EMF

EMF is a modelling framework that facilitates defining DSLs. EMF provides the Ecore metamodel, which is the core metamodel of EMF allowing for the specification of Ecore models. The metamodels of the DSLs adopted in PaaSage are Ecore models that conform to the Ecore metamodel (see Figure 1). The Ecore metamodel, in turn, is an Ecore model that conforms to itself (*i.e.*, it is reflexive).

EMF allows generating Java class hierarchy representations of the metamodels based on those definitions. The Java representations provide a set of APIs that enables the programmatic manipulation of models.

[1] `https://www.eclipse.org/modeling/emf/`
[2] `https://www.eclipse.org/cdo/`

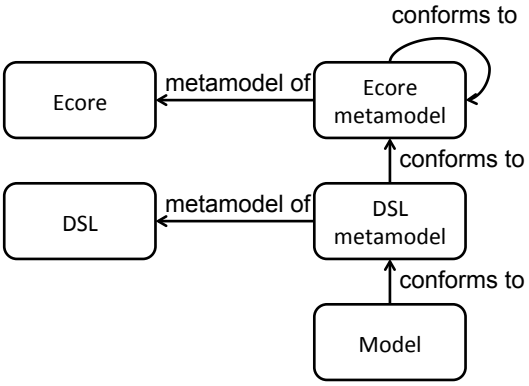

Figure 1: The Ecore-based modelling stack in PaaSage

EMF also provides code generation facilities that can be used to automatically generate a tree-based editor, as well as frameworks such as Graphical Modeling Framework (GMF)[3] or Graphical Editing Framework (GEF)[4] to manually create a custom graphical editor. Please note that, at month 18, the editor functionality of the PaaSage modelling environment is provided by the automatically generated tree-based editor. In this deliverable, this editor is used to illustrate the specification of several sample models.

2.2 CDO

CDO provides a semi-automated persistence mechanism which is adapted to work natively with Ecore models and their instances. It can be used as a model repository where clients persist and distribute their models. Its features can be used to satisfy the design-time and run-time requirements of the PaaSage platform. Such features include providing automated and optimised persistence with pluggable back-ends, query languages, Java interfaces, automatic notifications, and failover mechanisms.

Figure 2 shows the general server architecture of CDO. This architecture is made up of a *physical integration layer* and a *logical layer*. This separation of concerns allows for a high degree of flexibility when adopting a persistence strategy for the DSLs.

[3] `https://www.eclipse.org/modeling/gmp/`
[4] `https://www.eclipse.org/gef/`

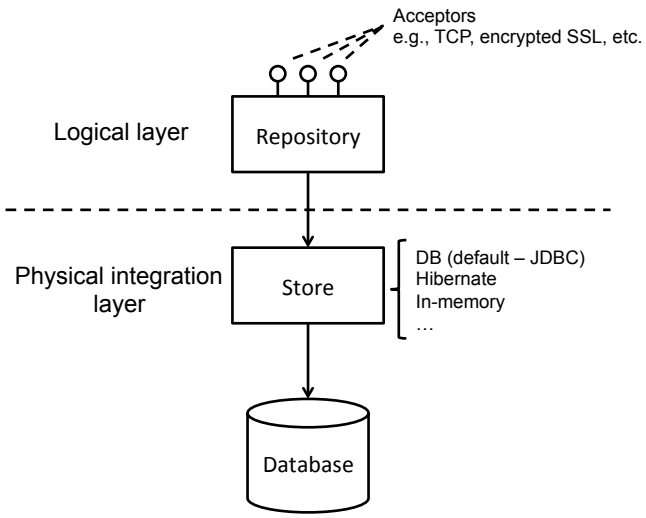

Figure 2: Overview of the CDO architecture

Physical integration layer

The physical integration layer of CDO consists of so-called *stores*, which are associated with different database back-ends. The DB store can be associated with any database management system that implements the JDBC API. This store provides an internal object-relational (O-R) mapping that is used for serialising/deserialising models to/from the database. The internal mapping can be customised by either selecting one of the provided mapping strategies (horizontal mapping strategy: one table per concrete class, no joins, etc.) or by defining annotations on model elements in EMF.

The Hibernate[5] store uses object-relational mapping metadata for serialising model instances to database tuples and vice versa. This metadata can be defined by use of either the Java Persistence API[6] specification (writing annotations directly within the Java objects or defining them in separate mapping files) or the proprietary Hibernate format (hbm.xml mapping files). Hibernate also supports a variety of database management system back-ends including the ones mentioned for the DB store. This store also provides an internal automatic mapping generation using the Teneo[7] model-relational mapping framework.

[5] `http://hibernate.org/`
[6] `https://www.eclipse.org/eclipselink/jpa.php`
[7] `http://wiki.eclipse.org/Teneo`

Logical layer

The logical layer of CDO represents an abstraction over the underlying stores and provides several types of Java APIs designed to access them. The functional APIs can be used with all store types. They provide a means to programmatically access models persisted in the repository through a set of function calls. Additionally, CDO provides query language interfaces that can also be used for accessing the underlying stores. They are used by passing query strings within a certain function call. The particular query languages currently supported by CDO are Structured Query Language (SQL), Object Constraint Language (OCL), and Hibernate Query Language (HQL). Please note that the latter is only usable with Hibernate stores.

Features beneficial for the PaaSage platform

CDO provides several features that will facilitate the implementation of the PaaSage platform:

- Transactional Support: CDO supports transactions on the models persisted in the repository. The transactional APIs are designed after the JDBC APIs, whereas each client of the repository first opens a *session* to start a *transaction* that can be used to manipulate the model.
- Model Validation: CDO checks that models committed to the repository conform to their metamodel. This ensures that the models persisted in the repository are consistent and valid at any time.
- Model Versioning: CDO supports *optimistic* versioning [24], where each client of the repository has a local (or working) copy of a model. These local copies are modified independently and in parallel and, as needed, local modifications can be committed to the repository. In this respect, CDO facilitates conflict detection: if the modifications are non-overlapping, they are automatically merged; otherwise, they are rejected, and the model is put in a *conflict* state that requires manual intervention.
- Automatic Propagation of Events: CDO allows for clients to receive automatic notifications about the state of the models persisted in the repository. This includes the possibility of consistent propagation of modifications across all clients of the repository.
- Metamodel Branching: By default, all clients of the repository work on the same version of the metamodel. However, each client of the repository can be configured to create and work on different branches of the metamodel.

- Auditing: CDO keeps a record of versions of each model since its creation. This allows for clients of the repository to have access to the history of each model for auditing purposes.
- Role-based Security: CDO provides a built-in access control designed for Ecore models. This allows for a fine-grained role-based access rights definition.

3 CLOUDML

PaaSage uses provisioning and deployment models in all life-cycle phases of configuration, deployment, and execution (*cf.* D2.1.1 [25]), meaning that they are progressively refined throughout the PaaSage work-flow. For this purpose, we have developed CLOUDML[8] [8, 9], which consists of a tool-supported DSL for modelling and enacting the provisioning and deployment of multi-cloud applications, as well as for facilitating their dynamic adaptation, by leveraging upon MDE techniques and methods.

CLOUDML has been designed based on the following requirements:

Cloud provider-independence (R_1): CLOUDML should support a cloud provider-agnostic specification of the provisioning and deployment. This will simplify the design of multi-cloud applications and prevent vendor lock-in.

Separation of concerns (R_2): CLOUDML should support a modular, loosely-coupled specification of the provisioning and deployment so that the modules can be seamlessly substituted. This will facilitate the maintenance as well as the dynamic adaptation of the deployment topology.

Reusability (R_3): CLOUDML should support the specification of types or patterns that can be seamlessly reused to design the system. This will ease the evolution as well as the rapid development of different variants of a system in time and in space.

Abstraction (R_4): CLOUDML should provide an up-to-date, abstract representation of the running system. This will facilitate the reasoning, simulation and validation of the adaptation actions before their actual enactments.

White- and black-box infrastructure (R_5): CLOUDML should support both IaaS and PaaS solutions. This will enable various degrees of control over underlying infrastructures and platforms of a multi-cloud application.

[8] `http://cloumdml.org`

CLOUDML is inspired by the OMG Model-Driven Architecture [17] and allows developers to model the provisioning and deployment of a multi-cloud application at two levels of abstraction: *(i)* the Cloud Provider-Independent Model (CPIM), which specifies the provisioning and deployment of a multi-cloud application in a cloud provider-agnostic way (addressing the requirement R_1); *(ii)* the Cloud Provider-Specific Model (CPSM), which refines the CPIM and specifies the provisioning and deployment of a multi-cloud application in a cloud provider-specific way. This two-level approach is agnostic to any development paradigm and technology, meaning that the application developers can design and implement their applications based on their preferred paradigms and technologies.

CLOUDML is also inspired by component-based approaches [31], which facilitate separation of concerns (R_2) and reusability (R_3). In this respect, deployment models can be regarded as assemblies of components exposing ports, and bindings between these ports.

To this end, CLOUDML implements the *type-instance* pattern [2], which also facilitates reusability (R_3) and abstraction (R_4). This pattern exploits two flavours of typing, namely *ontological* and *linguistic* [15]. Figure 3 illustrates these two flavours of typing. SL (short for Small GNU/Linux) represents a reusable type of virtual machine. It is linguistically typed by the class VM (short for Virtual Machine). SL1 represents an instance of the virtual machine SL. It is ontologically typed by SL and linguistically typed by VMInstance.

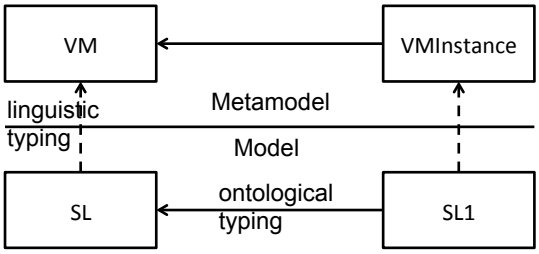

Figure 3: Linguistic and ontological typing

In the following, we provide a description of the most important classes and corresponding properties in the CLOUDML metamodel as well as sample models conforming to this metamodel. These sample models are analogous to the sample models in the SENSAPP running example from D2.1.1 [25].

Figure 4 shows the type portion of the CLOUDML metamodel, and Figure 5 shows the hierarchy of classes of the complete CLOUDML metamodel.

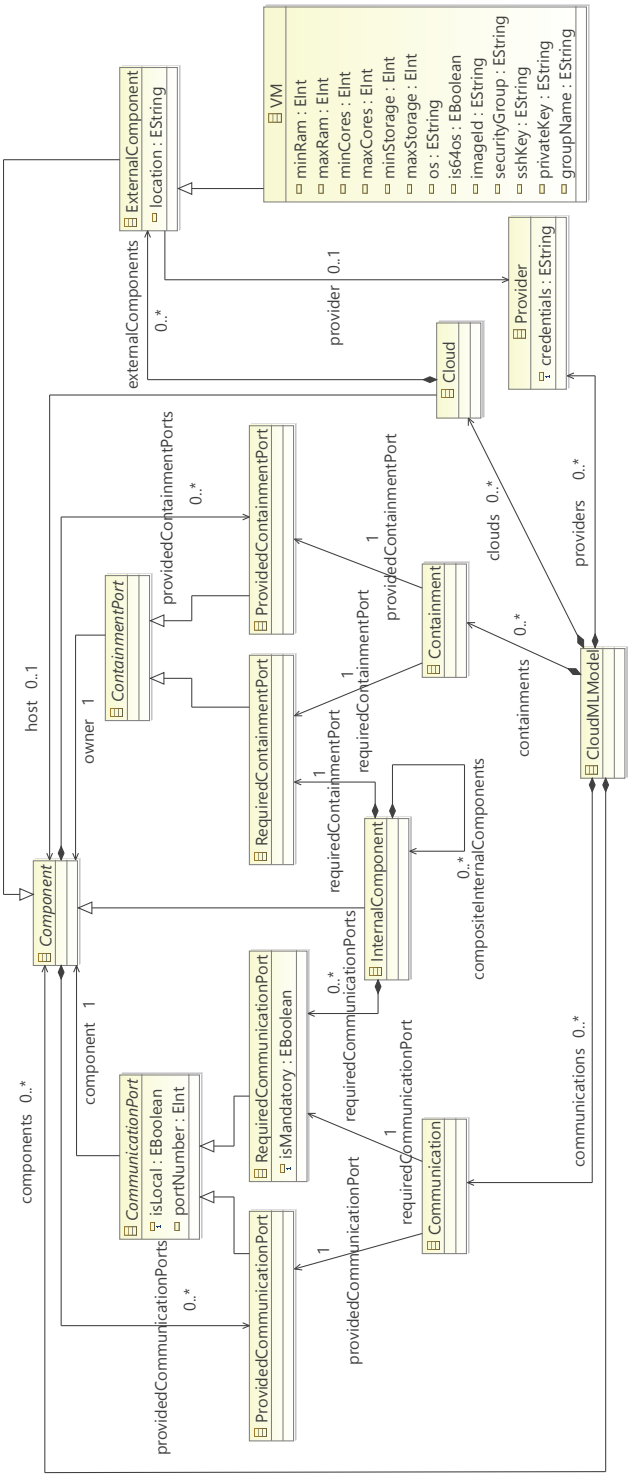

Figure 4: Type part of the CLOUDML metamodel

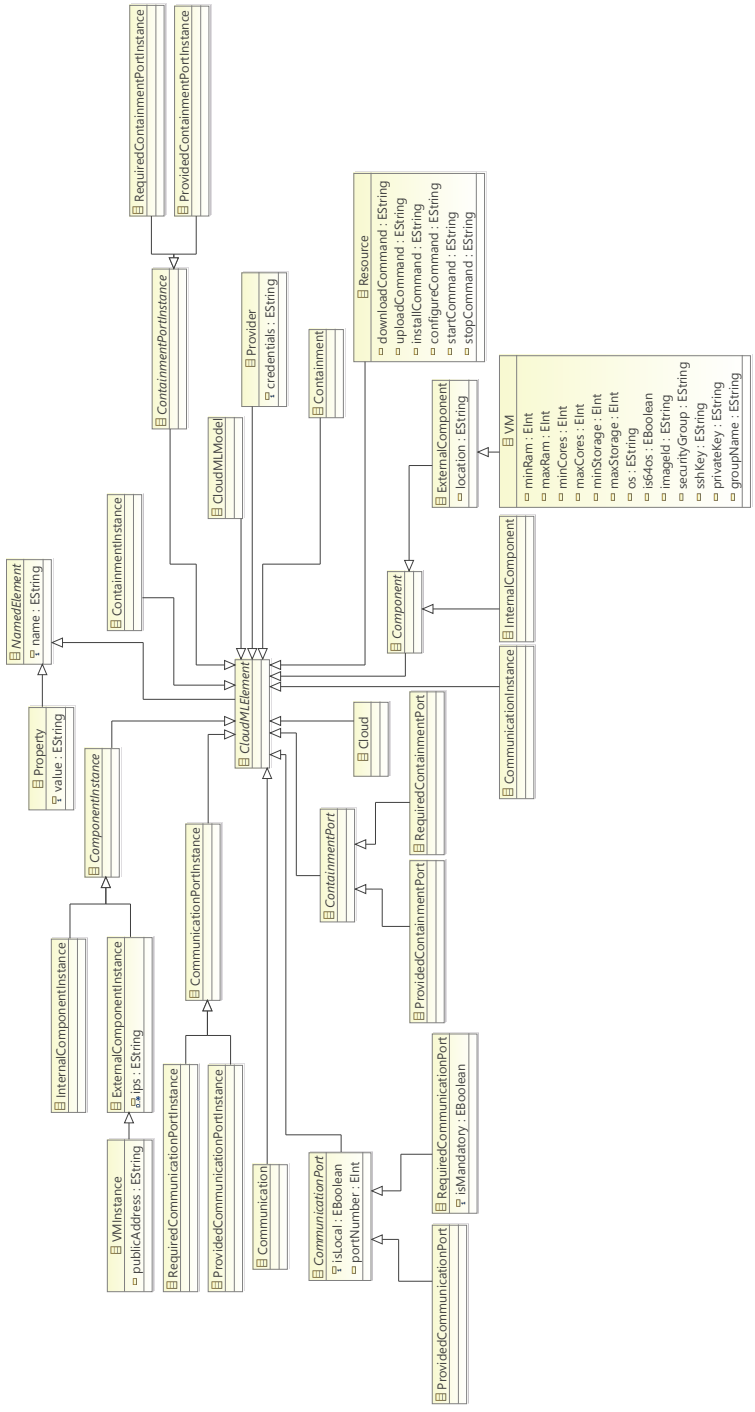

Figure 5: Class inheritance hierarchy of the CLOUDML metamodel

A CloudMLModel consists of CloudMLElements which can be associated with Property and Resources. A Resource represents an artefact (*e.g.*, scripts, binaries, configuration files, etc.) adopted to manage the deployment life-cycle (*e.g.*, download, configure, install, and start, stop). The three main types of CloudMLElements are Component, Communication, and Containment.

3.1 Components

A Component represents a reusable type of component of a cloud-based application. A Component can be an ExternalComponent, meaning that it is managed by an external Provider (*e.g.*, an Amazon Beanstalk container, see Figure 6), or an InternalComponent, meaning that it is managed by the PaaSage platform (*e.g.*, a Servlet container or SENSAPP, see Figure 7). This mechanism enables supporting both IaaS and PaaS solutions (R_5). The property location of ExternalComponent represents the geographical location of the data centre hosting (*e.g.*, location="eu", short for Europe).

An ExternalComponent can be a VM (*e.g.*, a virtual machine running GNU/-Linux, see Figure 8). The properties minCores, maxCores, minRam, maxRam, minStorage, and maxStorage depict the lower and upper bounds of virtual compute cores, RAM, and storage, respectively, of the required virtual machine (*e.g.*, minCores=1, minRam=1024). The property OS represents the operating system to be run by the virtual machine (*e.g.*, OS="ubuntu").

Example

Figure 6, 7, and 8 show excerpts of an ExternalComponent, an InternalComponent, and a VM, respectively, specified using an EMF tree-based editor (see Section 2).

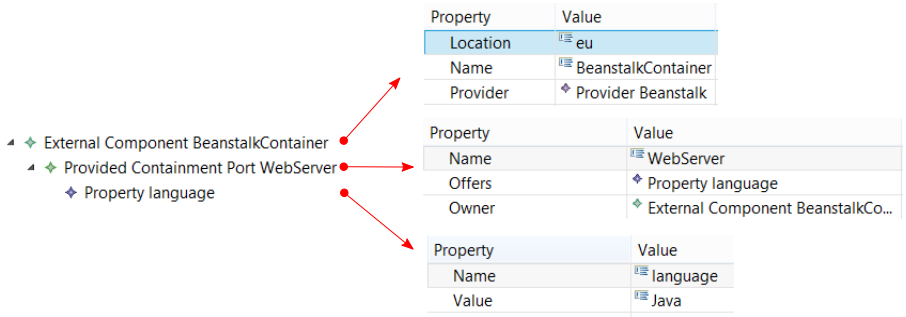

Figure 6: A sample ExternalComponent

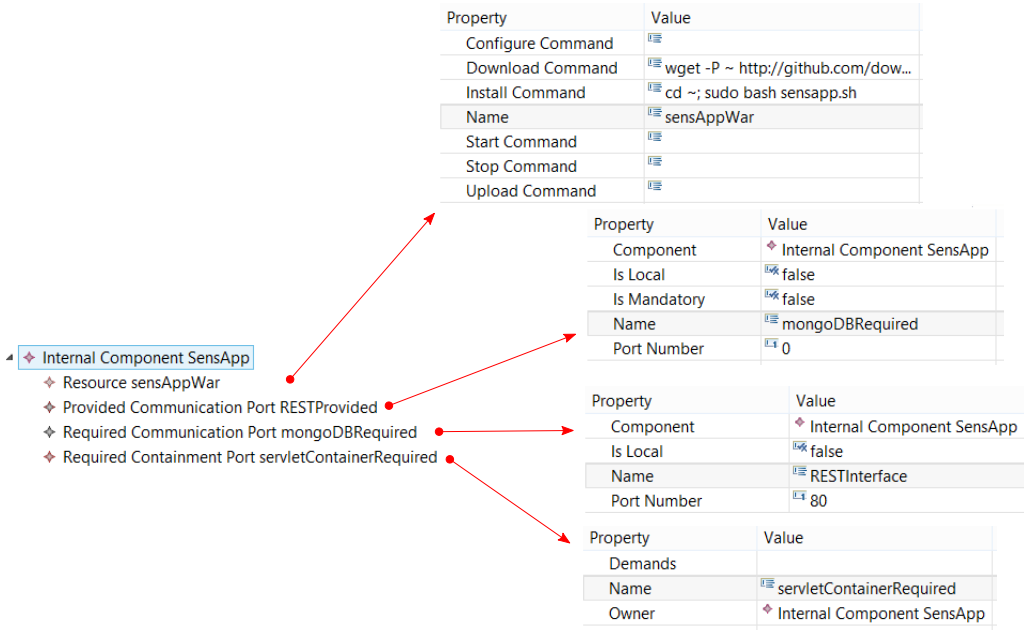

Figure 7: A sample InternalComponent

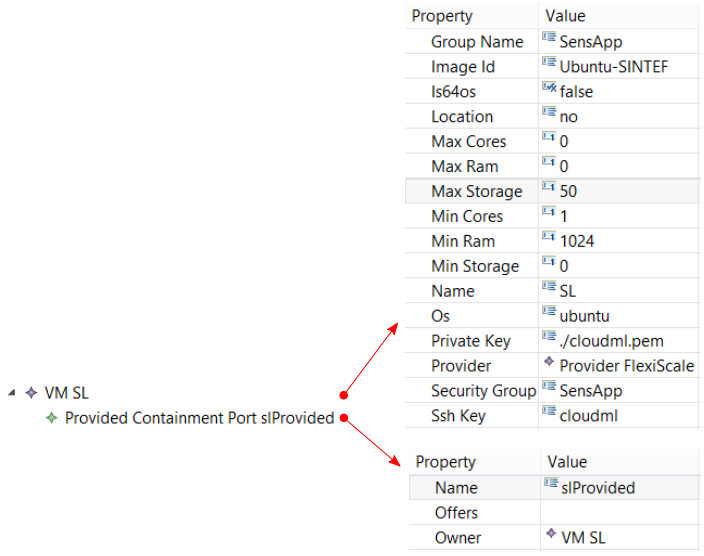

Figure 8: A sample VM

3.2 Communications

A CommunicationPort represents a communication interface of a component. A CommunicationPort can be a ProvidedCommunicationPort, meaning that it provides a feature to another component (*e.g.*, SENSAPP provides a REST interface), or a RequiredCommunicationPort, meaning that it consumes a feature from another component (*e.g.*, SENSAPP ADMIN requires a SENSAPP REST interface). Only internal components can have a RequiredCommunicationPort since they are managed by the PaaSage platform. The property isLocal represents that the component requesting the feature and the component providing the feature have to be deployed on the same external component (*e.g.*, SENSAPP and MongoDB have to be deployed on the same virtual machine, see Figure 7). The property isMandatory of RequiredPort represents that the InternalComponent depends on this feature (*e.g.*, SENSAPP depends on MongoDB and hence MongoDB has to be deployed before SENSAPP, see Figure 7).

A Communication represents a reusable type of communication binding between a Required- and a ProvidedCommunicationPort (*e.g.*, SENSAPP communicates with the SENSAPP ADMIN through HTTP on port 80, see Figure 9). A Communication can be associated with Resources specifying how to configure the components so that they can communicate with each other.

Example

Figure 9 shows an excerpt of a Communication specified using an EMF tree-based editor (see Section 2).

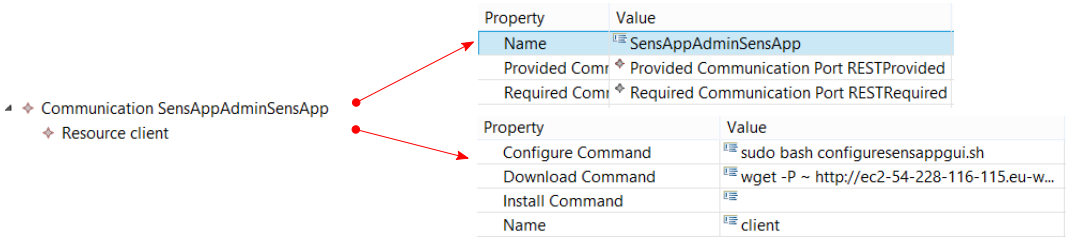

Figure 9: A sample Communication

3.3 Containments

A ContainmentPort represents a containment interface of a component. A ContainmentPort can be a ProvidedContainmentPort, meaning that it provides an execution environment to another component (*e.g.*, a virtual machine running

GNU/Linux provides an execution environment to a Servlet container), or a RequiredContainmentPort, meaning that an internal component consumes an execution environment from another component (*e.g.*, SENSAPP requires a Servlet container, see Figure 7).

A Containment represents a reusable type of containment binding between Required- and a ProvidedContainmentPort (*e.g.*, a Servlet container is contained by a virtual machine running GNU/Linux, see Figure 10). A Containment can be associated with Resources specifying how to configure the components so that the contained component can be deployed on the container component.

Example

Figure 10 shows an excerpt of a Containment specified using an EMF tree-based editor (see Section 2).

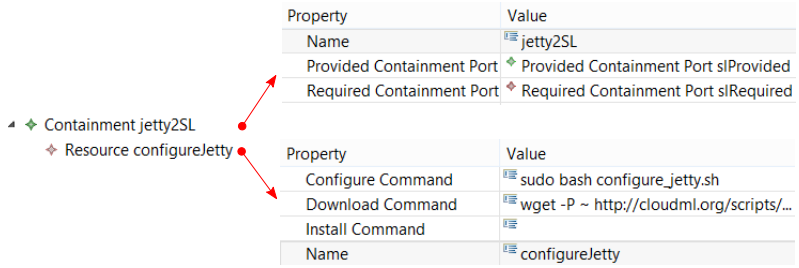

Figure 10: A sample Containment

3.4 Component, Communication, and Containment instances

The types presented above can be instantiated in order to form an assembly of components that specifies a provisioning and deployment model.

Example

Figure 11 shows an instantiation of the sample VM type running GNU/Linux presented above (see Figure 8). Please note that when instantiating a composite type (*e.g.*, the sl1 VMInstance of SL VM type), all of its dependent types also have to be instantiated along with it (*e.g.*, the slProvided1 ContainmentPortInstance of slProvided ContainmentPort type).

The complete serialisation of the SENSAPP example in XMI format specified using the EMF tree-based editor, is available in Appendix A.

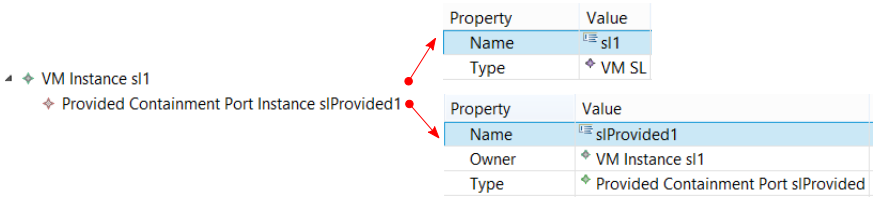

Figure 11: A sample VMInstance

4 Saloon

PaaSage uses profiles of cloud providers in the life-cycle phase of configuration and deployment for matching the provisioning and deployment models with the compatible cloud providers (*cf.* D2.1.1 [25]). For this purpose, we have adapted and extended Saloon [21, 20, 19], which consists of a language and a framework for specifying application requirements and user goals of multi-cloud applications and selecting compatible cloud providers by leveraging upon feature models [3] and ontologies [10].

Saloon uses four metamodels: Feature Metamodel, Ontology Metamodel, Mapping Metamodel, and Type Metamodel. Figure 12 illustrates the relationships between these metamodels.

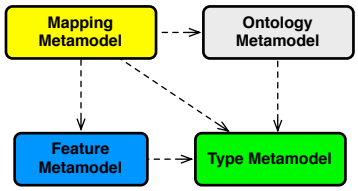

Figure 12: Saloon Metamodels overview

The Feature Metamodel enables the specification of feature models (FMs) characterising cloud providers. The Ontology Metamodel enables the definition of a cloud ontology that encompasses different concepts from cloud providers, application requirements, and user goals. The Mapping Metamodel defines the relationships between the FMs and the cloud ontology, allowing the matching of cloud providers with application requirements and user goals. Finally, the Type Metamodel defines basic types required by the other metamodels, such as string, integer, and float. In the following, we provide a description of the most important classes and corresponding properties in these metamodels as well as sample models conforming to these metamodels.

4.1 Feature

Figure 13 shows the Saloon Feature Metamodel.

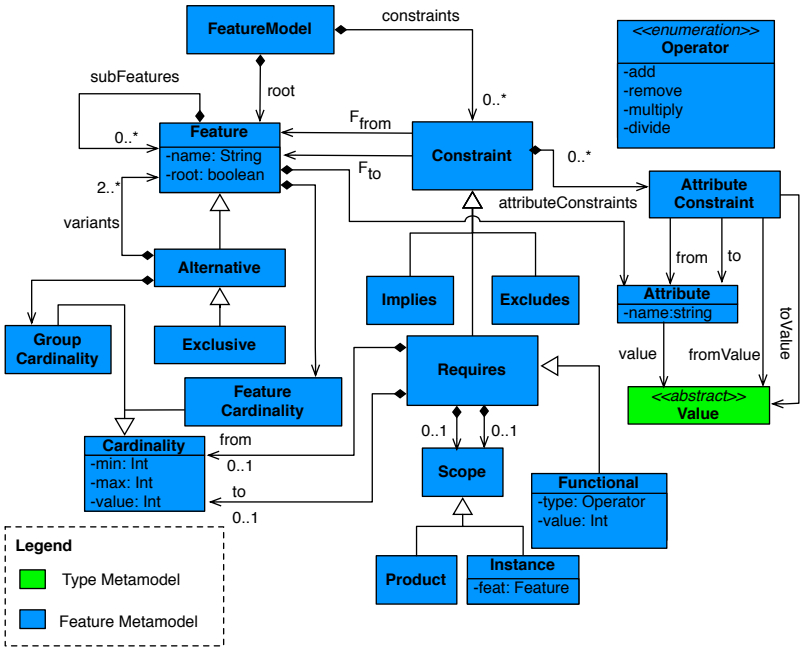

Figure 13: Saloon Feature Metamodel

A FeatureModel has a root Feature and a set of Constraints. A Feature has a Feature Cardinality. The properties min and max represent the lower and upper bound of the cardinality, respectively, while the property value represents a value in this range. A Feature can also have subfeatures and be specialised to Alternative, meaning that at least one feature in the group should be selected, or Exclusive, meaning that exactly one feature should be selected. In the Saloon Feature Metamodel, Alternatives can also have a different Group Cardinality with arbitrary lower and upper bounds (*e.g.*, if the Alternative consists of a group of five choices, the Group Cardinality of 3..5 denotes that at least three features in the group have to be selected).

A Constraint represents a typical restriction in binary feature models [12]. A Constraint can be an Implies constraint, meaning that a given feature requires another feature when selected (*i.e.*, both features have to be together in a valid configuration), or an Excludes constraint, meaning that one feature excludes another one when selected (*i.e.*, both features can not be together in a valid configuration). A Constraint can also be a Requires constraint, enabling the specification of restrictions of the form:

$$\boxed{[x', x'']A \rightarrow [y', y'']B} \text{ with } x', x'', y', y'' \in \mathbb{N}, \text{ and } x' \leq x'', y' \leq y''$$

This restriction denotes that if cardinality of feature A is between x' and x'', the cardinality of feature B must be in $[y', y'']$. A Requires constraint can be specialised to a Functional constraint, enabling the specification of restrictions of the form:

$$\boxed{[x', x'']A \rightarrow +[y]B} \text{ with } x', x'', y \in \mathbb{N}, \text{ and } x' \leq x''$$

This constraint denotes that a feature A with cardinality between $[x', x'']$ requires y more instances of feature B in a valid configuration. A Requires constraint can also be specialised to a Attribute Constraint, enabling the specification of restrictions of the form:

$$\boxed{(A).c = X \rightarrow (B).d = Y}$$

This constraint denotes that if the attribute *c* of A has X as value, then the attribute *d* of B needs Y as value.

Example

Figure 14 shows an excerpt of the FM for Amazon EC2[9] specified using an EMF tree-based editor (see Section 2), while Figure 15 depicts the same model using the FODA notation [12].

In Figure 15, Amazon EC2 represents the root feature having attributes deploymentModel and serviceModel. The Virtual Machine feature is mandatory since its cardinality is 1..*. This feature has attributes vmMemorySize, vmSize, vmOS, vmCpuCores and vmStorage. Each attribute has a corresponding domain (*e.g.*, the vmOS attribute has an enumeration as domain with values Ubuntu, WindowsServer and RedHatEntrepriseLinux).

The Location and Pricing Model features are also mandatory, while the Services feature is optional (cf. Figure 15).

The FM for Amazon EC2 also includes some Attribute Constraints, *e.g.*:

$$\boxed{\text{(Virtual Machine).vmSize= M} \rightarrow \text{(Virtual Machine).vmCpuCores=1}}$$

This constraint denotes that a value M for the attribute vmSize requires a value 1 for the attribute vmCpuCores.

[9] `http://aws.amazon.com/ec2/`

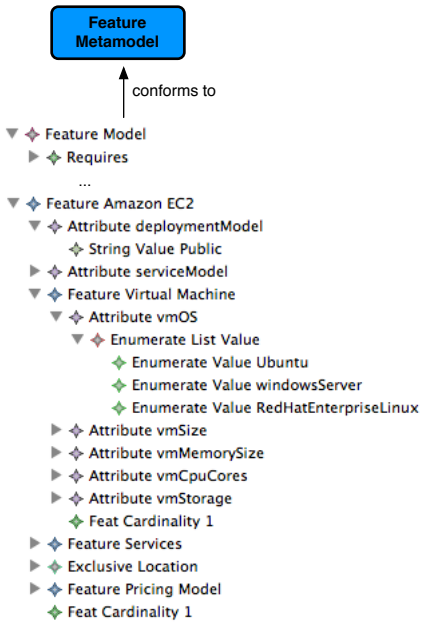

Figure 14: Amazon EC2 Feature Model (Excerpt)

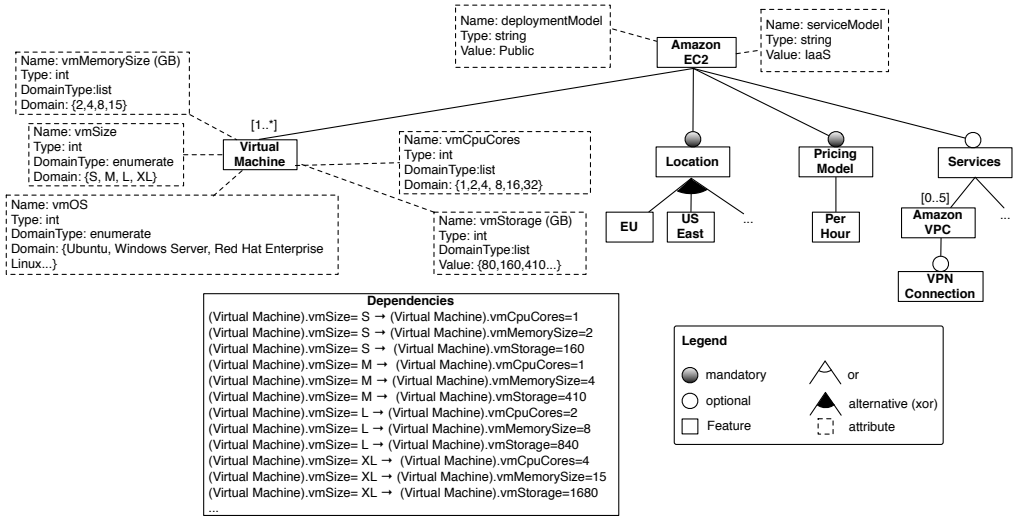

Figure 15: Amazon EC2 Feature Diagram (Excerpt)

4.2 Ontology

Figure 16 depicts the Saloon Ontology Metamodel.

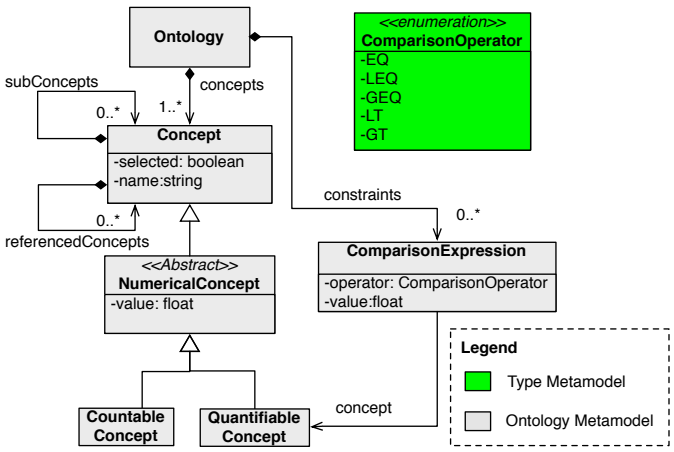

Figure 16: Saloon Ontology Metamodel

An Ontology has one or more Concepts and zero or more constraints on some of them. A Concept can have related sub-concepts that specialise the parent concept (*e.g.*, Tomcat and Jetty are sub-concepts of Application Server). A Concept can also have referenced concepts that provide more information about the referencing concept (*e.g.*, OS and Memory concepts provide more information about the Virtual Machine concept). A Concept can be specialised to a CountableConcept, meaning that the concept can be counted (*e.g.*, Virtual Machine), or QuantifiableConcept, meaning that the concept can be measured (*e.g.*, Memory).

A ComparisonExpression represents a constraint on values of QuantifiableConcepts. The property operator, *i.e.*, $=$, $\leq$, $\geq$, $<$ or $>$ is used to define the comparison with a given value.

Example

In PaaSage, we have defined a Saloon Ontology considering the offerings from providers such as Amazon EC2, ElasticHosts[10], Heroku[11] and Windows Azure[12]. Figure 17 shows an excerpt of this ontology. Concepts in orange represent concrete concepts, *i.e.*, concepts that do not have sub-concepts. The Virtual Machine is an example of CountableConcept. It refers to the OS and Resource concepts. The different resources Memory, CPU and Disk, in contrast, are examples of QuantifiableConcepts.

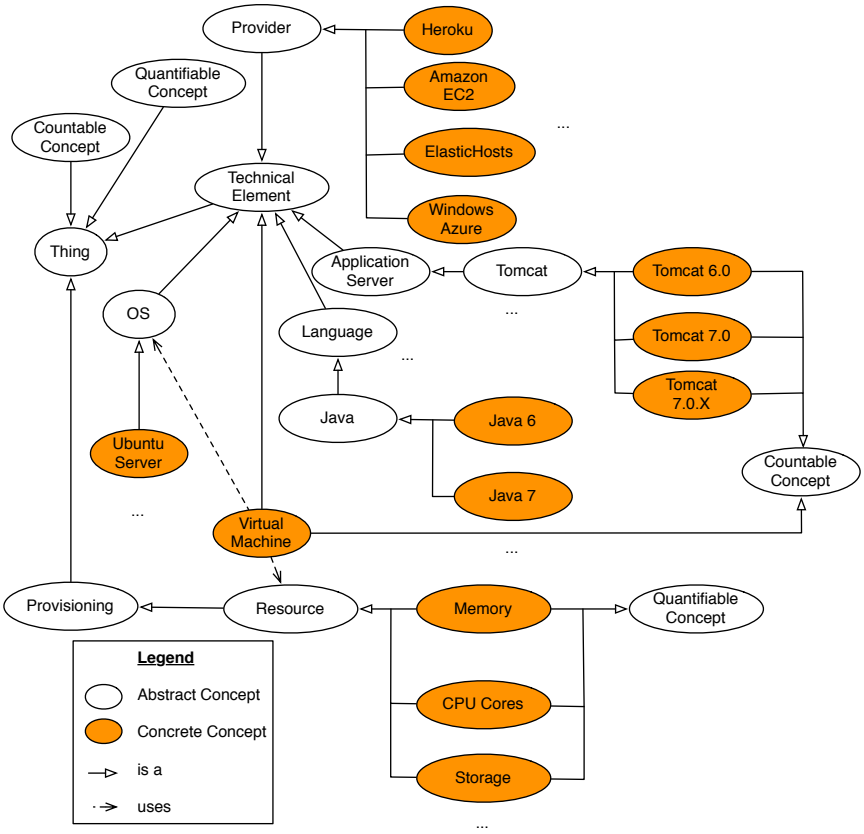

Figure 17: Cloud Ontology Diagram (Excerpt)

Figure 31 in Appendix B presents the whole ontology diagram.

[10] http://www.elastichosts.com/
[11] https://www.heroku.com/
[12] https://www.windowsazure.com/

4.3 Mapping

Figure 18 depicts the Saloon Mapping Metamodel.

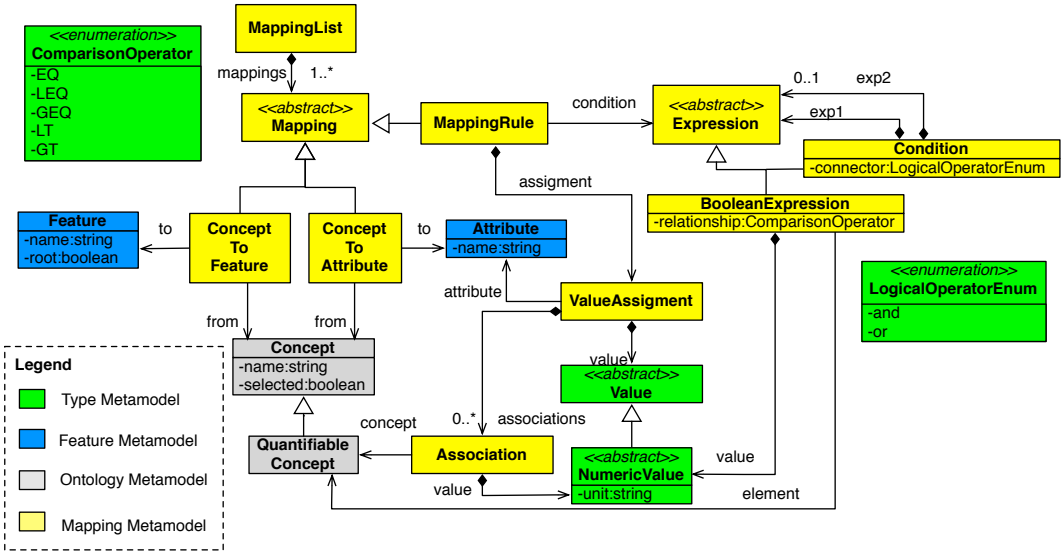

Figure 18: Saloon Mapping Metamodel

As mentioned, this metamodel defines the relationships between the FMs and the cloud ontology, allowing the matching of cloud providers with application requirements and user goals. A MappingList has multiple Mappings from ConceptToFeature or from ConceptToAttribute. Therefore, each Mapping has a related Concept and a Feature or Attribute.

A MappingRule enables the specification of complex mappings. It has an Expression, indicating when the mapping must be applied. An Expression can be a Condition or a BooleanExpression. A Condition consists of BooleanExpressions and/or other Conditions, which are connected using the and and or logical operators. A BooleanExpression defines a comparison between a QuantifiableConcept and a NumericValue with the $=$, $\leq$, $\geq$, $<$ and $>$ operators. A MappingRule also has a ValueAssigment for an Attribute that is executed if the Expression is true.

Example

As each cloud provider defines its own terminology to describe its offerings, a mapping model is required for each one of them. Figure 19 shows an excerpt of the mapping for Amazon EC2. This mapping model only defines ConceptToFeature (*e.g.*, Virtual Machine to Virtual Machine) and ConceptToAttribute (*e.g.*, Disk to vmStorage).

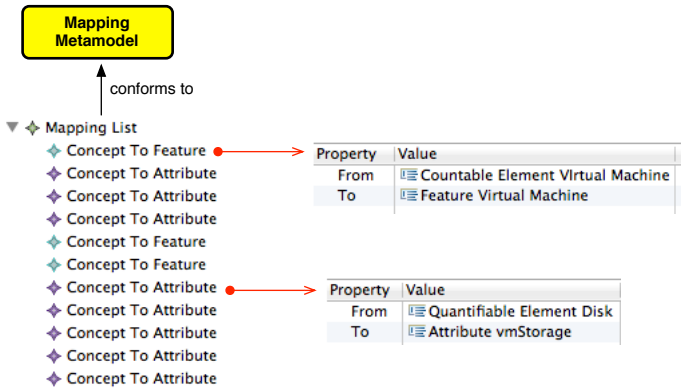

Figure 19: Amazon EC2 Mapping Model (Excerpt)

4.4 Type

Figure 20 depicts the Saloon Type Metamodel. As mentioned, the Type Metamodel defines basic types required by the other metamodels. In particular, this metamodel includes basic types such as string, integer, and float, as well as more complex types such as enumeration, list, and range values. Logic and comparison operators are also included as enumeration types.

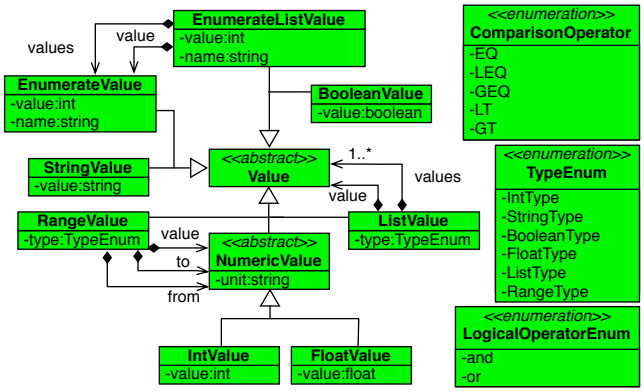

Figure 20: Saloon Type Metamodel

5 WS-Agreement

PaaSage uses SLAs in the life-cycle phases of configuration and deployment for finding the most suitable provisioning and deployment model for multi-cloud applications, as well as in the life-cycle phase of execution for monitoring the QoS (*cf.* D2.1.1 [25]). For this purpose, we have adopted Web Services Agreement (WS-Agreement) [1], which consists of a language and a protocol for advertising the capabilities of service providers, creating SLAs based on templates, and monitoring SLAs at run-time.

In the following, we provide a description of the most important classes and corresponding properties in the WS-Agreement metamodel as well as sample models conforming to this metamodel.

5.1 Agreements

Unlike the other DSLs adopted in PaaSaage, WS-Agreement abstract syntax is defined by an XML schema [1]. In order to have all the DSLs adopted in the same technical space, this schema was transformed to a corresponding metamodel in Ecore.

Figure 21 shows the WS-Agreement metamodel. An AgreementContextType represents the context information associated with an agreement, such as service provider and expiration date. An AgreementType represents the name and identity associated with an AgreementContextType. A AssessmentIntervalType represents a time interval and a count of assessments associated with an agreement. A ServiceLevelObjectiveType represents an SLO the agreement is based on. A KPITargetType represent a key performance indicator (KPI) associated with one or more SLOs. A CompensationType represents a penalty and reward (in term of business value) associated with one or more SLOs. A GuaranteeTermType represents the level of quality of service (QoS) from the service provider in the agreement, and is associated with a business value specific to the importance of the level of quality being met. A TermCompositorType is used as logical AND/OR/XOR operators to logically group GuaranteeTermTypes and/or other TermCompositorTypes underneath it, where the logical group refers to all, one or more, or exactly one level of QoS. This provides a recursive structure to the logical composition of terms.

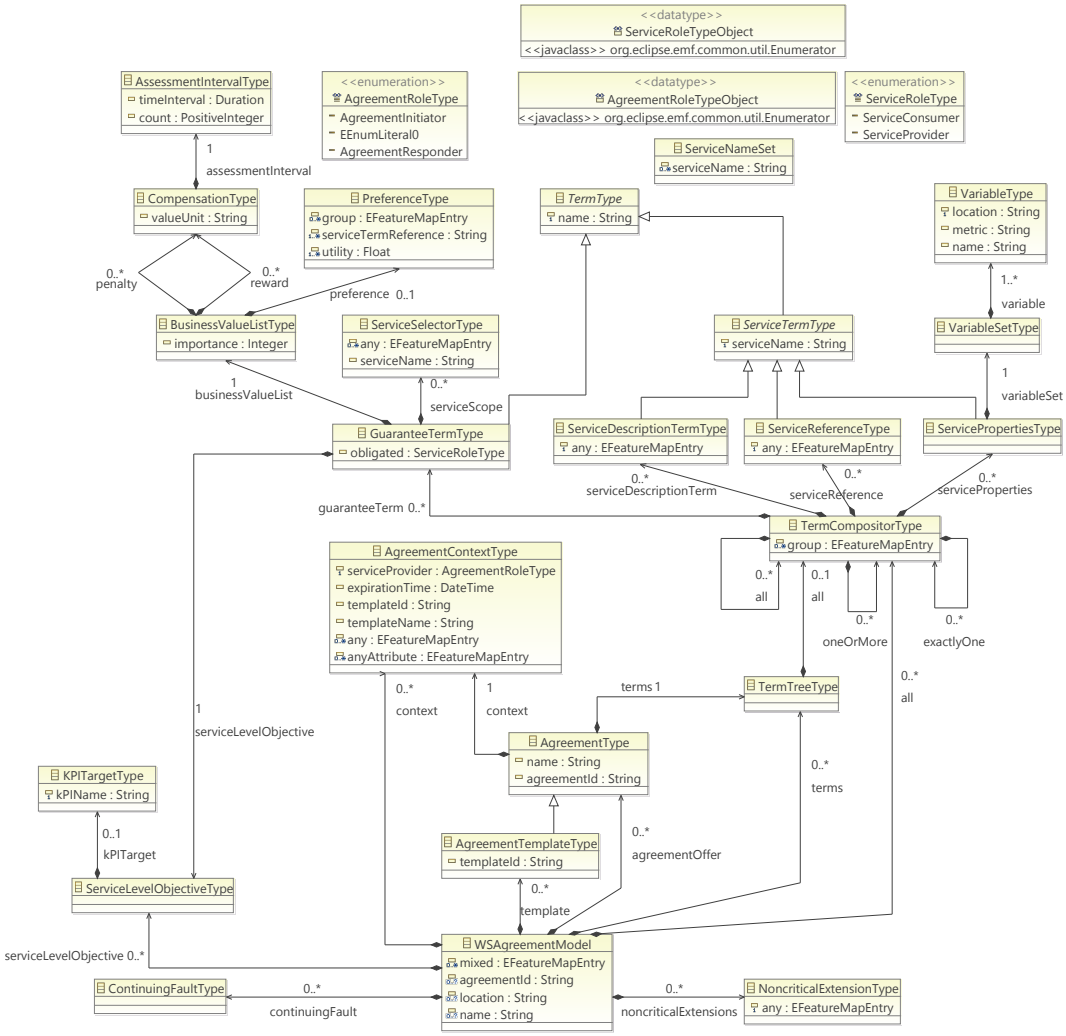

Figure 21: WS-Agreement metamodel

Examples

Figure 22 shows a WS-Agreement model specified using an EMF tree-based editor (see Section 2) that describes the key performance indicators associated with a SLO.

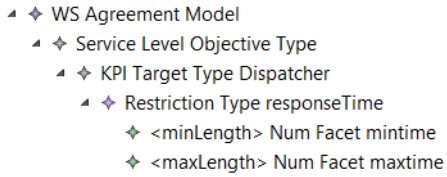

Figure 22: A sample SLO

Figure 23 shows another WS-Agreement model using an EMF tree-based editor (see Section 2) that describes the service description term related to the location of the processing node.

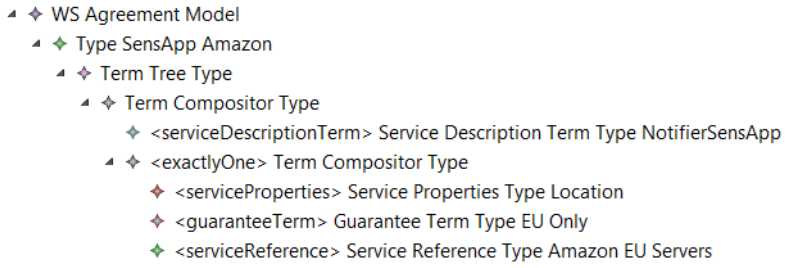

Figure 23: A sample service description term

6 Scalability Rules Language

PaaSage uses scalability rules in the life-cycle phase of execution for guaranteeing QoS levels (*cf.* D2.1.1 [25]). For this purpose, we have developed the Scalability Rules Language (SRL), which consists of a language for specifying noteworthy behavioural patterns of multi-cloud applications, along with the corresponding actions to change the provisioning and deployment model in response to these patterns.

When the reaction is scoped within a single cloud, it will be managed by the Executionware. Alternatively, it will be managed by the Upperware. The complex actions managed by the Upperware may include bursting the application to an additional cloud, scaling the application across cloud boundaries, or changing the application configuration followed by its re-deployment. In order to enact these changes, the Upperware interacts with the Executionware.

In general, all actions associated with the scalability rules are triggered on event patterns. In order to identify patterns in the application behaviour, component instances need to be monitored. As a consequence, SLR provides mechanisms which can be used to: express which components of an application will be monitored by which sensors or metric aggregators (aggregating measurements through formulas involving one or more metrics), define patterns on the monitoring data, and express actions that have to be executed when a pattern is matched.

SRL is based on scalability capabilities of existing cloud platforms and middleware. In particular, Amazon CloudWatch[13] and Cloudify's Automatic Scaling Rules[14] have served as a primary source of inspiration. Nevertheless, SRL goes beyond these two mechanisms for various reasons: it is cross-cloud capable and does not depend on the capabilities offered by each cloud provider. Moreover, it enables combining existing metrics and linking them to computations and event patterns. The metric description used in SRL as well as parts of the terminology are taken from OWL-Q [13].

In the following, we provide a description of the most important classes and corresponding properties in the SRL metamodel as well as sample models conforming to this metamodel.

An SRLModel consists of ScalabilityRules, which represent generic scalability rules. A ScalabilityRule is associated with other elements in the metamodel through the references: relatedTo, when a scalability rule is related to a specific event, and actions, when a scalability rule may trigger one or more scalability actions. Since the SRL metamodel spans over different information aspects, we have split its description in sub-sections.

[13] http://aws.amazon.com/cloudwatch/

[14] http://getcloudify.org/guide/2.7/developing/scaling_rules.html

6.1 Events

Figure 24 shows the portion of the metamodel that is related to events.

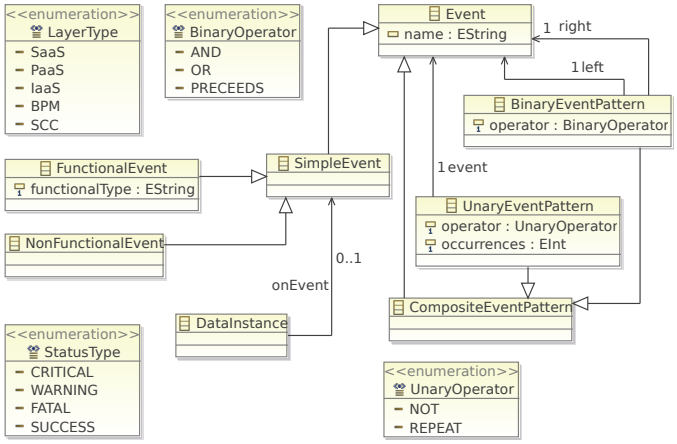

Figure 24: Events metamodel

An Event can be a SimpleEvent or a CompositeEventPattern. A SimpleEvent can be either functional (*e.g.*, a failure of a component instance), or non-functional (*e.g.*, a violation of an SLO). While currently not supported by the metamodel, we are investigating how to extend scalability rules so that they can provide own actions for functional events.

A CompositeEventPattern can be a BinaryEventPattern or a UnaryEventPattern. A BinaryEventPattern connects two other Events by a binary operator. These include common, logical operators such as AND and OR, but also time-based operators, such as PRECEDS, which represents that an event has to occur prior to another one. For instance, the condition A AND $(B$ OR $C)$ can be expressed as a BinaryEventPattern X_1 that comprises a SimpleEvent A and another BinaryEventPattern X_2 connected by the AND operator. X_2, in turn, comprises two SimpleEvents B and C connected by the OR operator. A UnaryEventPattern refers just to one event along with an operator that is applied on the event and enables expressing cases where, *e.g.*, the negation of this event or the REPEATition of this event is required. The property occurrences of UnaryEventPattern represents the number of repetitions for the latter case.

A DataInstance represents the actual data associated with the events that occurred in the system (*e.g.*, the actual value measured, the component that produced the event, etc., see Figure 26). Moreover, the status property represents the status of the event. This property provides useful insight for the Upperware when performing off-line evaluation of the application performance as well as enables the evaluation/assessment of the events. The property layer represents

the layer in the cloud stack where the event has occurred. IaaS or PaaS indicate that the event relates to IaaS or PaaS services used by an application, respectively, while SaaS, indicates that the event relates to the the application as a whole, or to a third-party SaaS (*e.g.*, Amazon Simple Email Service[15]). Additional (sub-)layers have been included in order to further distinguish different types of SaaS services: SCC (short for Service Composition) indicates that service composition is concerned, while BPM (short for Business Process Management) indicates that business processes are concerned.

6.2 Scheduling and Conditions

Figure 25 shows the portion of the metamodel that is related to scheduling and conditions.

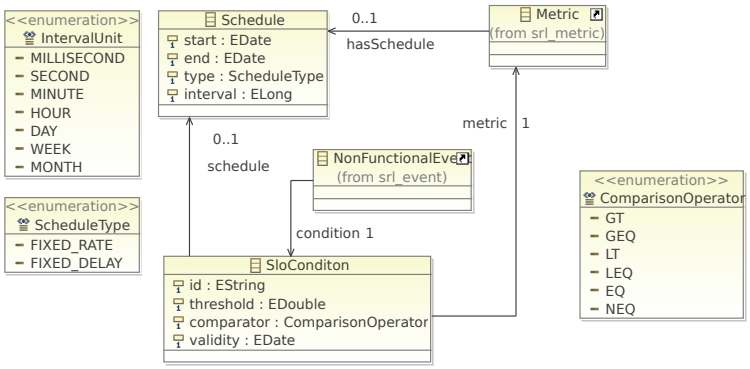

Figure 25: Scheduling and Conditions metamodel

A Schedule represents any aspect of the operations or measurements that need to be executed on a regular, timely basis, such as when an operation/measurement will be run and when the scheduling shall end. The property type represents whether successive runs happen at a fixed rate or with a fixed delay. Moreover, the property intervalUnit represents the time unit used for a Schedule's interval.

Simple, non-functional events (see Section 6.1) refer to SloConditions, which are derived from the SLOs defined using WS-Agreement (see Section 5). An SloCondition is used to compare the measurement of a Metric (see Section 6.3) represented by the property value of DataInstance (see Section 6.1) to a threshold represented by the property threshold of SloCondition. The comparison between the measured value and threshold is performed according to a (binary) ComparisonOperator. The ComparisonOperator enumerates all possible comparison operators that can be used, *i.e.*, greater than or less than (including and

[15] `http://aws.amazon.com/ses/`

excluding equality) as well as (in)equality. The point in time when an SloCondition is evaluated is a choice of configuration. This can be done based either on every new measurement created for the associated event, represented by the boolean property newValue in SloCondition, or based on time, in which case the SloCondition has to be associated with a Schedule.

6.3 Patterns and Metrics

Figure 26 shows the portion of the metamodel that is related to patterns and metrics.

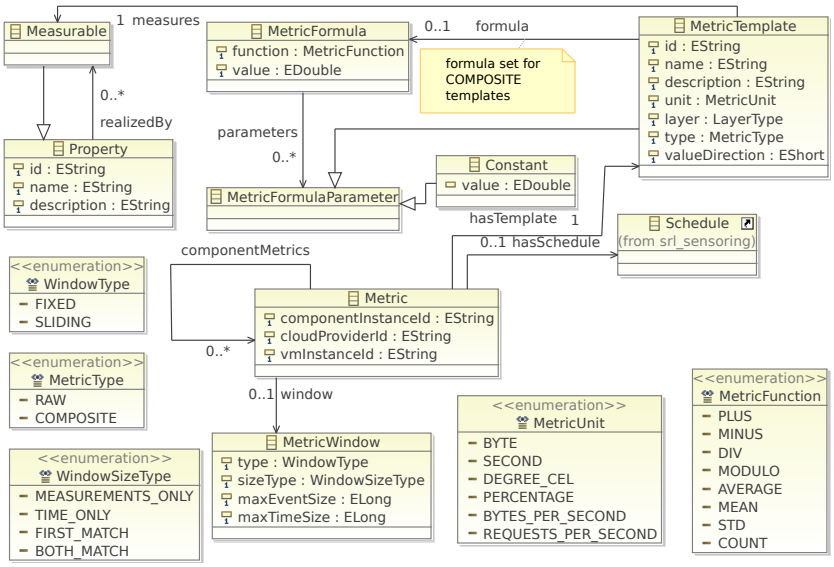

Figure 26: Patterns and Metrics metamodel

A Metric represents a generic metric that can refer to either raw measurements or aggregations on them. A MetricTemplate represents common metric information that might have been repeated if encapsulated directly in Metric. Hence, multiple Metrics can refer to the very same MetricTemplate. In contrast to the repeated information, a MetricTemplate does not specify dynamic details such as how often the respective metric measurements are produced and which component instances on which cloud are concerned. The property type represents the kind of template and hence its associated Metrics. RAW indicates that the associated Metric refers to raw measurments, while COMPOSITE indicates that the metric is computed from other metrics. In the latter case, a MetricTemplate is associated with a MetricFormula, which represents the function

that computes a metric from other metrics. A MetricTemplate is also associated with Measurable, which represents the property that the MetricTemplate (actually its associated metrics) measures.

For its computing task, a MetricFormula may be associated to one or more MetricFormulaParameters, which can be Constants or other MetricTemplates. The parameters are used as input to the function for computing the composite metric. For instance, the computation of an availability metric may divide another metric such as *uptime* with a specific constant value. With respect to non-constant parameters, the computation for a metric m proceeds from m to the respective template mt, then to formula and parameters, and finally matches the non-constant parameters with the contents of the m.componentMetrics set (*i.e.*, the metric components).

A Metric may be associated with a MetricWindow, which represents how many DataInstances will be temporary stored and used to perform computations. The window size may be defined by a time period, or by a fixed number of events, or a combination of both. In the latter case, it may be sufficient to wait for either the first property to be fulfilled, or for both. The property sizeType represents the strategy to be used for this purpose. Moreover, the property type represents what happens when the window size has been reached. SLIDING indicates that the windows is slid by dropping superfluous elements, while FIXED indicates that the window is cleared.

A Metric may also be associated to a Schedule. In case of COMPOSITE metrics, the schedule defines when and how often the metric will be evaluated by applying the associated MetricFormula. Please note that this reference is indirect through the respective MetricTemplate. In case of RAW metrics, the schedule defines how often the value is measured by the respective sensor.

Figure 27 shows further aspects of RAW metrics.

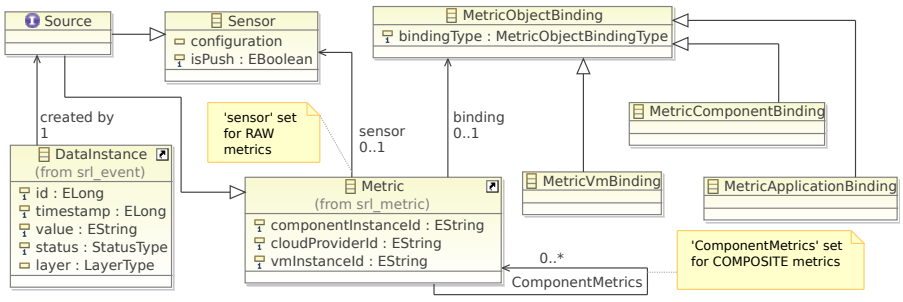

Figure 27: RAW metrics metamodel

A RAW metric is associated with one Sensor, which produces the actual measurement. Details regarding which component/object is actually measured

are exclusively covered by the MetricObjectBinding associated with the Metric. In particular, different components can be measured depending on the respective type of MetricObjectBinding, such as the application as a whole, one or more of its component instances, or one or more of the underlying virtual machine instances in a particular cloud.

Please note that the modelling does not state whether values are polled from the sensor by the metric (rather the service responsible for the collection of the metric measurements) or pushed by the sensor. The measurements by sensors can include the status of a component. For instance, this is important with respect to integration of the HyperFlow engine where the workflow engine may require reconfiguration of the application depending on the stage of the workflow (*cf.* D2.1.1e [26] and D5.1.1 [7]).

6.4 Actions

Figure 28 illustrates the portion of the metamodel that is related to actions.

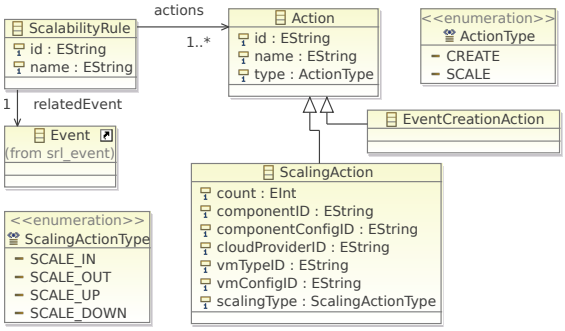

Figure 28: Actions metamodel

A triggered ScalabilityRule leads to performing of one or more Actions, which can be ScalingActions or EventCreationActions.

A ScalingAction represents which entities of an application will be changed and how. The scalingType property represents how the affected entities will be changed. SCALE_OUT, SCALE_IN, SCALE_UP, and SCALE_DOWN indicate the corresponding type of changes. The other properties of ScalingAction reference the affected entities and also indicate their number (*e.g.*, to add a specific number of instances of a specific component). The affected entities are actually associated with their respective definitions in the corresponding CLOUDML model (see Section 3) and may refer to component instances, virtual machine instances, and their respective configurations. In this way, a ScalingAction specifies all the necessary details in order to execute the four action types envisioned in the project.

An EventCreationAction, in contrast, only specifies that the application has reached a state where the current provisioning and deployment model may need to be checked by the Upperware. For instance, this is the case for the HyperFlow workflow engine that is to be integrated into the Executionware (*cf.* D5.1.1 [7]).

6.5 Examples

In this section, we present how two common requirements are represented in sample SRL models. The first example operates at the level of the application as a whole and considers the actions to take when availability drops below or response time exceeds a certain threshold. The second example operates at the level of a single component, namely a Couchbase distributed database, and maps empirical knowledge about the database gained through experiments to a scalability rule.

Availability and Response Time

This sample SRL model is analogous to the sample models in the SENSAPP running example from D2.1.1 [25] and D4.1.1 [14]. Here, we assume that an expert has provided the following scalability rule in prose:

> Scale out when the average application response time goes beyond $300ms$ or its availability falls below 99.9%.

Figure 29 shows an excerpt of the model of this rule, while the following paragraphs discuss the individual entities. Here, we assume that SENSAPP has an application identifier of 12345.

The requirements leads to a single BinaryEventPattern associated with two non-functional SimpleEvents connected by an OR operator. Each of the SimpleEvents is associated with an SloCondition. The respective SLO conditions specify the thresholds of 99.9% and $30ms$, respectively.

The events above lead to two high-level metrics: Metric1 represents the response time and Metric2 represents the availability of the application. Both metrics are associated with their respective MetricTemplates. Response time is a RAW metric, as the response time is directly measured by a sensor. Availability, in contrast, is a COMPOSITE metric, as it is computed by dividing Metric3, which represents the uptime, with a constant Constant1, which represents the total monitoring time. In order to compute this metric, Metric2 has attached a MetricFormula. We do not show the formula in this example, but we do so in the next one.

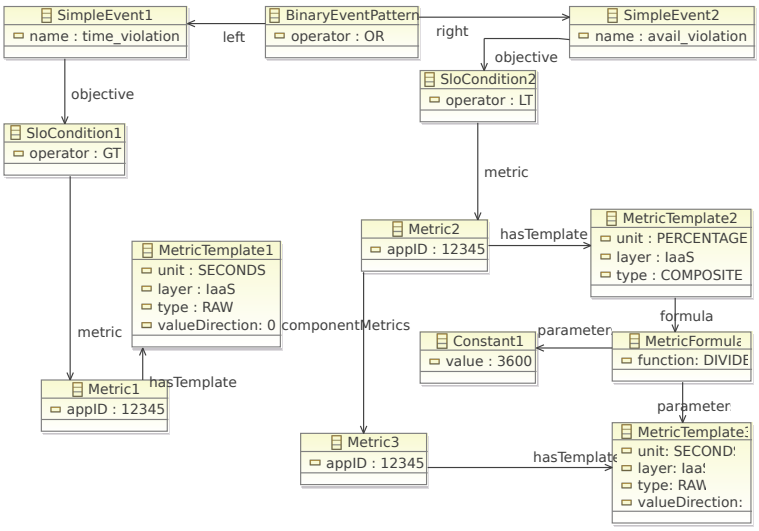

Figure 29: A sample SRL model for availability and response time (Excerpt)

The metrics above lead to a single rule that is associated with a single Action. Yet, it is unknown to the rule processing how the scale out described in prose should take place. Therefore, the action will be an EventCreationAction that leaves the right decisions to the Upperware (*cf.* D1.6.1 [11]). Again, we do not show the action in this example.

Database

This sample SRL model scales a component when all instances of a single component in a particular cloud have a CPU load beyond 50% and at the same time at least one instance has a load beyond 85%. The average over $5min$ and $1min$ will be computed for the 50% and 85% thresholds, respectively. The sensors will be queried once per second. We found this pattern particularly useful for scaling out the Couchbase[16] distributed database [29].

Figure 29 shows an excerpt of the model of this rule, while the following paragraphs discuss the individual entities. In order to avoid clutter, this figure omits some composite metrics, which, however, are symmetric to the ones that are shown; it also omits the events created for this example. Here, we assume that Couchbase has a component identifier 99-99. Furthermore, we assume that the cloud on which Cloudbased is deployed has a cloud identifier 88-88. Couchbase is scaled to 3 instances with component instance identifiers 1, 2, 3. A CPU sensor is associated with each of these instances.

[16] `http://www.couchbase.com/`

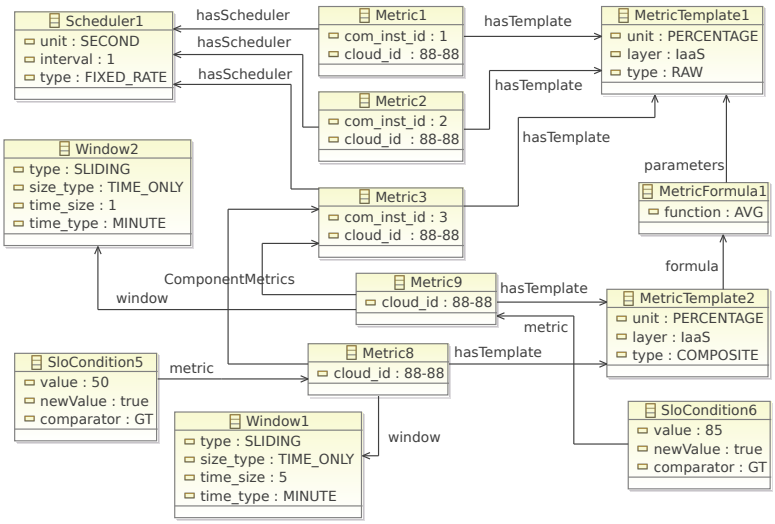

Figure 30: A sample SRL model model for Couchbase scalability rule (Excerpt)

	NonFunctionalEvent		BinaryEventPattern
#e1	avg 50% load for 1 in 5 min	#e7	#e1 AND #e2
#e2	avg 50% load for 2 in 5 min	#e8	#e3 AND #e7
#e3	avg 50% load for 3 in 5 min	#e9	#e4 OR #e5
#e4	avg 85% load for 1 in 1 min	#e10	#e6 OR #e9
#e5	avg 85% load for 2 in 1 min	#e11	#e10 AND #e8
#e6	avg 85% load for 3 in 1 min		

Table 1: Events created for database scalability rule

The requirements lead to six simple, non-functional Events (see Table 1). In particular, these Events correspond to the condition that either of the three instances exceed the 50% and 85% threshold, respectively. The overall event connects the SimpleEvent by either AND or OR operators.

The events above leads to a RAW metric per monitored sensor, *i.e.* to three symmetric Metrics, with a single, common MetricTemplate. The layer of this MetricTemplate is IaaS. Each of the Metrics is associated with the same Schedule gathering new data once per second. Any of these RAW metrics is part of two COMPOSITE metrics that share a common MetricTemplate. Furthermore, two Windows are shared among the respective COMPOSITE metrics. Figure 30 only shows the three RAW metrics (Metric1–Metric3) together with two COMPOSITE metrics (Metric8 and Metric9). The four other metrics (Metric4–Metric7) are symmetric to Metric8 and Metric9, but refer to Metric2 and Metric1, respectively.

	SloCondition				
#sloc1	value	50	#sloc4	value	85
	comparator	GT		comparator	GT
	newValue	true		newValue	true
	metric	Metric4		metric	Metric7
#sloc2	value	85	#sloc5	value	50
	comparator	GT		comparator	GT
	newValue	true		newValue	true
	metric	Metric5		metric	Metric8
#sloc3	value	50	#sloc6	value	85
	comparator	GT		comparator	GT
	newValue	true		newValue	true
	metric	Metric6		metric	Metric9

Table 2: SloConditions created for database scalability rule

The requirements lead to six SloConditions (see Table 2). In particular, three conditions check for a 50% threshold and three conditions checking for an 85% threshold. Whenever the metrics associated with these SloConditions generate values that violate the SloCondition, the non-functional event associated with the SloCondition is triggered.

The occurrence of a sufficient subset of non-functional events activates the outer BinaryEventPatter #e11. The ScalabilityRules for this example require that when event #e11 occurs, a single SCALE_OUT action for Couchbase 99–99 on cloud 88–88 is triggered, that adds an additional instance to the database cluster.

7 Java APIs and CDO

As mentioned, the DSLs adopted in PaaSage are represented by metamodels in Ecore (see Section 2). This enables to specify models using an EMF tree-based editor as well as to programmatically manipulate and persist them through Java APIs. Listing 1 shows an excerpt of the Java code for creating a CLOUDML model for the SENSAPP example that is equivalent to the CLOUDML model specified using the EMF tree-based editor (see Section 3). The full version of the Java code is available for reference in Appendix C.

Listing 1 shows the creation of an `ExternalComponent` equivalent to the one in Figure 6. The classes that are instantiated and initialised in the code have been automatically generated by the EMF generator model based on the CLOUDML metamodel in Ecore. All class instances are obtained using the `CloudmlFactory` object specific for the CLOUDML metamodel. This object provides a set of methods which are used to make sure that the model objects are appropriately instantiated.

Listing 1: A sample `Provider` definition

```
ExternalComponent beanstalkContainer = CloudmlFactory.eINSTANCE.
    createExternalComponent();
beanstalkContainer.setName("BeanstalkContainer");
beanstalkContainer.setProvider(beanstalk);
beanstalkContainer.setLocation("eu");

ProvidedContainmentPort webServer = CloudmlFactory.eINSTANCE.
    createProvidedContainmentPort();
webServer.setName("webServer");
webServer.setOwner(beanstalkContainer);

Property webServerLanguage = CloudmlFactory.eINSTANCE.createProperty()
    ;
webServerLanguage.setName("language");
webServerLanguage.setValue("Java");

webServer.getProperties().add(webServerLanguage);
beanstalkContainer.getProvidedContainmentPorts().add(webServer);
```

Listing 2 shows the creation of a `VM` equivalent to the one in Figure 8.

Listing 2: A sample `VM` definition

```
// create a SL VM
VM sl = CloudmlFactory.eINSTANCE.createVM();
sl.setName("SL");
sl.setMinCores(1);
sl.setMaxCores(0);
sl.setMinRam(1024);
sl.setMaxRam(0);
sl.setMinStorage(50);
sl.setMaxStorage(0);
sl.setLocation("no");
sl.setOs("ubuntu");
sl.setSshKey("cloudml");
sl.setSecurityGroup("SensApp");
sl.setGroupName("SensApp");
sl.setPrivateKey("./cloudml.pem");
sl.setImageId("Ubuntu-SINTEF");
sl.setIs64os(false);
sl.setProvider(flexiScale);
```

Listing 3 shows the creation an `InternalComponent` equivalent to the one in Figure 7. This `InternalComponent` has an associated `Resource` representing a WAR file along with the corresponding life-cycle control scripts

(*e.g.*, download and install commands). It also has a Servlet container `RequiredContainmentPort`, a MongoDB `RequiredCommunicationPort`, and an HTTP `ProvidedCommunicationPort`.

Listing 3: A sample `InternalComponent` definition

```
// create a SensApp InternalComponent
InternalComponent sensApp = CloudmlFactory.eINSTANCE.
    createInternalComponent();
sensApp.setName("SensApp");

Resource sensAppWar = CloudmlFactory.eINSTANCE.createResource();
sensAppWar.setDownloadCommand(""
+ "wget␣-P␣~␣http://github.com/downloads/SINTEF-9012/sensapp/sensapp.
    war;␣"
+ "wget␣-P␣~␣http://ec2-54-228-116-115.eu-west-1.compute.amazonaws.com
    /scripts/linux/ubuntu/sensapp/sensapp.sh");
sensAppWar.setInstallCommand("cd␣~;␣sudo␣bash␣sensapp.sh");
sensAppWar.setName("sensAppWar");

RequiredContainmentPort servletContainerRequired = CloudmlFactory.
    eINSTANCE.createRequiredContainmentPort();
servletContainerRequired.setName("servletContainerRequired");
servletContainerRequired.setOwner(sensApp);

RequiredCommunicationPort mongoDBRequired = CloudmlFactory.eINSTANCE.
    createRequiredCommunicationPort();
mongoDBRequired.setName("mongoDBRequired");
mongoDBRequired.setIsLocal(true);
mongoDBRequired.setIsMandatory(true);
mongoDBRequired.setPortNumber(0);
mongoDBRequired.setComponent(sensApp);

ProvidedCommunicationPort restProvided = CloudmlFactory.eINSTANCE.
    createProvidedCommunicationPort();
restProvided.setName("RESTProvided");
restProvided.setIsLocal(false);
restProvided.setPortNumber(8080);
restProvided.setComponent(sensApp);

sensApp.getResources().add(sensAppWar);
sensApp.getProvidedCommunicationPorts().add(restProvided);
sensApp.getRequiredCommunicationPorts().add(mongoDBRequired);
sensApp.setRequiredContainmentPort(servletContainerRequired);
```

Listing 4 shows the creation of a `Communication` binding between the SENSAPP and the SENSAPP ADMIN `InternalComponents`, equivalent to the one in Figure 9.

Listing 4: A sample `Communication` definition

```
Communication sensAppAdminSensApp = CloudmlFactory.eINSTANCE.
    createCommunication();
sensAppAdminSensApp.setName("SensAppAdminSensApp");
sensAppAdminSensApp.setProvidedCommunicationPort(restProvided);
sensAppAdminSensApp.setRequiredCommunicationPort(restRequired);
```

Listing 5 shows the creation of a `Containment` binding between the Jetty `InternalComponent` and the SL `ExternalComponent`, equivalent to the one in Figure 10.

Listing 5: A sample `Containment` definition

```
Containment jetty2Sl = CloudmlFactory.eINSTANCE.createContainment();
jetty2Sl.setName("jetty2SL");
jetty2Sl.setProvidedContainmentPort(slProvided);
jetty2Sl.setProvidedContainmentPort(slProvided);

Resource configureJetty = CloudmlFactory.eINSTANCE.createResource();
configureJetty.setName("configureJetty");
configureJetty.setConfigureCommand("sudo␣bash␣configure_jetty.sh");
configureJetty.setDownloadCommand("wget␣-P␣~␣http://cloudml.org/
    scripts/linux/ubuntu/sensappAdmin/configure_jetty.sh");

jetty2Sl.getResources().add(configureJetty);
```

Listing 6 shows the process of saving a CLOUDML model in a CDO repository. As mentioned, CDO uses a set of APIs which are designed after the JDBC APIs. In order to save a model we need to first create a session and obtain a transaction over it. This example adopts a local database that is accessed using a TCP connector from the Net4j framework[17], a partner project used within CDO. Once the transaction is obtained, the CLOUDML model is associated with the `CDOResource` responsible for its persistence, and the transaction is committed.

Listing 6: Saving a CLOUDML model in a CDO repository

```
// initialise and activate a container
final IManagedContainer container = ContainerUtil.createContainer();
Net4jUtil.prepareContainer(container);
TCPUtil.prepareContainer(container);
CDONet4jUtil.prepareContainer(container);
container.activate();

// create a Net4j TCP connector
final IConnector connector = (IConnector) TCPUtil.getConnector(
    container, "localhost:2036");

// create the session configuration
CDONet4jSessionConfiguration config = CDONet4jUtil.
    createNet4jSessionConfiguration();
config.setConnector(connector);
config.setRepositoryName("CloudMLCDORepository");

// create the actual session with the repository
CDONet4jSession cdoSession = config.openNet4jSession();

// obtain a transaction object
CDOTransaction transaction = cdoSession.openTransaction();
```

[17] `https://www.eclipse.org/modeling/emf/?project=net4j`

```
// create a CDO resource object
CDOResource resource = transaction.getOrCreateResource("/
    sensAppResource");

// associate the \cloudml model to the resource
resource.getContents().add(model);

// commit the transaction to persist the model
transaction.commit();
```

Listing 7 shows the process of loading and modifying a CLOUDML model. In this example, an SL VM is moved from the Flexiant to the AWS-EC2 cloud.

Listing 7: Loading and modifying a CLOUDML model in a CDO repository

```
// open a new transaction
CDOTransaction transaction = cdoSession.openTransaction();

// load the existing resource of SensApp
CDOResource resource = transaction.getResource("/sensAppResource");

VM vm = null;

// find the Flexiant cloud
assertTrue(resource.getContents().get(0) instanceof CloudMLModel);
CloudMLModel model = (CloudMLModel) resource.getContents().get(0);

EList<Cloud> clouds = model.getClouds();
Cloud flexiantCloud = null;
for (int i = 0; i < clouds.size(); i++) {
  flexiantCloud = clouds.get(i);
  if (flexiantCloud.getName().equalsIgnoreCase("Flexiant")) {

    // remove the first of the External Components of the Flexiant
        cloud
    vm = (VM) flexiantCloud.getExternalComponents().get(0);
        flexiantCloud.getExternalComponents().remove(0);
        break;
  }
}

// find the AWS-EC2 cloud
Cloud awsCloud = null;
for (int i = 0; i < clouds.size(); i++) {
  awsCloud = clouds.get(i);
  if (awsCloud.getName().equalsIgnoreCase("AWS-EC2")){
    // add the External Component from Flexiant to AWS-EC2
    awsCloud.getExternalComponents().add(vm);
    break;
  }
}

// commit the transaction to persist the updated model
transaction.commit();
```

The examples above show the Java code for programmatically saving, loading, and modifying a CLOUDML model in a CDO repository. The Java code for programmatically saving, loading, and modifying Saloon, WS-Agreement, and SLR models is analogous, since all DSLs adopted in PaaSage are represented by metamodels in Ecore, which allows for using the same Java APIs.

8 Metadata Database

The Metadata Database stores all information manipulated by the PaaSage platform. Hence, the Metadata Database schema has to cover all concepts in the metamodels of the DSLs presented above, along with additional concepts that are not currently in these metamodels, such as metadata about configuration, deployment, and execution models (*cf.* D4.1.1 [14], Sections 3.1.2-3.1.7). In this respect, the Metadata Database schema can be regarded as a super-set of the metamodels of the DSLs.

In order to guarantee that the Metadata Database schema covers all concepts in the metamodels of each DSL, O-R mappings have to be specified and realised. At month 18, we have specified these mappings graphically (*cf.* D4.1.1 [14], Sections 3.1.2-3.1.7). The realisation of these mapping is part of future work (see Section 10).

9 Related Work

In the cloud community, libraries such as jclouds[18] or DeltaCloud[19] provide generic APIs abstracting over the heterogeneous APIs of IaaS providers, thus reducing cost and effort of deploying multi-cloud applications. While these libraries effectively foster the deployment of cloud-based applications across multiple cloud infrastructures, they remain code-level solutions, which make design changes difficult and error-prone. More advanced frameworks such as Cloudify[20], Puppet[21], or Chef[22] provide DSLs that facilitate the specification and enactment of provisioning, deployment, monitoring, and adaptation of cloud-based applications, without being language-dependent. As for the research community, the mOSAIC [27] project tackles the vendor lock-in problem by providing an API for provisioning and deployment of multi-cloud applications. This solution is also limited to the code level. The Topology and Orchestration Specification

[18] `http://www.jclouds.org`
[19] `http://deltacloud.apache.org/`
[20] `http://www.cloudifysource.org/`
[21] `https://puppetlabs.com/`
[22] `http://www.opscode.com/chef/`

for Cloud Applications (TOSCA) [18] is a specification developed by the OASIS consortium, which provides a language for specifying the components comprising the topology of cloud-based applications along with the processes for their orchestration. By contrast with PaaSage, the aforementioned approaches only focus on the management of cloud-based applications deployed on IaaS environments.

The literature encompasses several approaches to the management of cloud-based applications deployed on PaaS environments. Sellami *et al.* [28] propose a model-driven approach to PaaS-independent provisioning and management of cloud-based applications. This approach includes a language for modelling provisioning and deployment, as well as a REST API for enacting them. The Cloud4SOA EU project [5] provides a framework for facilitating the matchmaking, management, monitoring, and migration of cloud-based applications on PaaS environments. By contrast with PaaSage, these approaches focus on one cloud delivery model only (*i.e.*, either IaaS or PaaS, but not both). In addition, their models are not causally connected to the running system, and may become irrelevant as soon as the running system is changed. The approaches proposed in the CloudScale [4] and Reservoir [22] projects suffer similar limitations.

The work of Shao *et al.* [30] was a first attempt to build a models@run-time platform for the cloud, but remains restricted to monitoring, without providing support for enactment of provisioning and deployment. To the best of our knowledge, PaaSage is thus the first attempt to reconcile cloud management solutions with modelling practices through the use of models@run-time.

The flexibility, expressiveness, and power of SRL compared to commercial scalability rules languages, but also to other languages such as SYBL [6], leads to a certain complexity in the model. While the complexity is well-justified, the language can support the production of simple expressions where some level of detail can be hidden. This simplification will be performed throughout the course of the project.

10 Conclusions and Future Work

In this deliverable, we have provided an initial version of the technical documentation of the DSLs adopted in PaaSage. In particular, we have described the modelling concepts, their attributes and their relationships, as well as the rules for combining these concepts to specify valid models that conform to these DSLs. Moreover, we have exemplified how to specify models through an Eclipse editor as well as how to programmatically manipulate and persist them through CDO.

In the future, we will aggregate the metamodels presented in this deliverable, along with additional concepts currently captured by the Metadata Database only (*e.g.* users, organisations, and roles) into an extensive CAMEL metamodel. This metamodel will link the various concepts from the DSLs in order to have a uniform and integrated CAMEL language. Moreover, we will realise the mapping between the Metadata Database and the CAMEL metamodel through one of the approaches provided by CDO. In particular, we plan to adopt either a DB or a Hibernate store provided by CDO, and exploit as much as possible the internal mappings of these stores, possibly customising them by means of annotations or Hibernate mapping files (see Section 2). This will ensure that the changes in the CAMEL metamodel are automatically reflected in the Metadata Database schema. Finally, we will validate these mappings to ensure that the Metadata Database persists appropriate information.

Please note the capabilities of the DSLs presented in this deliverable reflect our understanding of the requirements of PaaSage at month 18. These requirements will be developed iteratively throughout the course of the project. Therefore, an important task is to adapt the capabilities of the DSLs to the changing requirements, and adapt the Metadata Database schema accordingly. In this respect, the research partners in PaaSage will provide feedback on whether the elements of each DSL are adequate to develop the components of the PaaSage platform. Similarly, the industrial partners in PaaSage will provide feedback on whether the elements of each DSL are satisfactory for modelling the use cases.

References

[1] 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.

[2] Colin Atkinson and Thomas Kühne. "Rearchitecting the UML infrastructure". In: *ACM Transactions on Modeling and Computer Simulation* 12.4 (2002), pp. 290–321. DOI: 10.1145/643120.643123.

[3] David Benavides, Sergio Segura and Antonio Ruiz Cortés. "Automated analysis of feature models 20 years later: A literature review". In: *Inf. Syst.* 35.6 (2010), pp. 615–636. DOI: 10.1016/j.is.2010.01.001.

[4] Gunnar Brataas, Erlend Stav, Sebastian Lehrig, Steffen Becker, Goran Kopčak and Darko Huljenic. "CloudScale: scalability management for cloud systems". In: *ICPE 2013: 4th ACM/SPEC International Conference on Performance Engineering*. ACM, 2013, pp. 335–338. ISBN: 978-1-4503-1636-1. DOI: 10.1145/2479871.2479920.

[5] *Cloud4SOA EU project*. URL: http://www.cloud4soa.eu/.

[6] Georgiana Copil, Daniel Moldovan, Hong Linh Truong and Schahram Dustdar. "SYBL: An Extensible Language for Controlling Elasticity in Cloud Applications". In: *CCGrid 2013: 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE Computer Society, 2013, pp. 112–119. ISBN: 978-1-4673-6465-2. DOI: 10.1109/CCGrid.2013.42.

[7] Jörg Domaschka, Panagiotis Garefalakis, Damianos Metalidis, Chrysostomos Zeginis, Bartosz Balis, Dariusz Król, Craig Sheridan, Kuan Lu, Edwin Yaqub and Anthony Sulistio. *D5.1.1/D5.3.1 – Prototype Executionware, Prototype New Execution Engines*. PaaSage project deliverable. Oct. 2013.

[8] 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.

[9] Nicolas Ferry, Alessandro Rossini, Franck Chauvel, Brice Morin and Arnor Solberg. "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems". In: *CLOUD 2013: IEEE 6th International Conference on Cloud Computing*. Ed. by Lisa O'Conner. IEEE Computer Society, 2013, pp. 887–894. ISBN: 978-0-7695-5028-2. DOI: 10.1109/CLOUD.2013.133.

[10] Thomas R. Gruber. "A translation approach to portable ontology specifications". In: *Knowledge Acquisition* 5.2 (June 1993), pp. 199–220. ISSN: 1042-8143. DOI: 10.1006/knac.1993.1008.

[11] Keith Jeffery and Tom Kirkham. *D1.6.1 – Initial Architecture Design*. PaaSage project deliverable. Oct. 2013.

[12] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak and A. Spencer Peterson. *Feature-Oriented Domain Analysis (FODA) - Feasibility Study*. Tech. rep. The Software Engineering Institute, 1990. URL: http://www.sei.cmu.edu/reports/90tr021.pdf.

[13] Kyriakos Kritikos and Dimitris Plexousakis. “OWL-Q for Semantic QoS-based Web Service Description and Discovery”. In: *SMR^2 2007: Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web*. Ed. by Tommaso Di Noia, Rubén Lara, Axel Polleres, Ioan Toma, Takahiro Kawamura, Matthias Klusch, Abraham Bernstein, Massimo Paolucci, Alain Leger and David L. Martin. Vol. 243. CEUR Workshop Proceedings. CEUR, 2007.

[14] Kyriakos Kritikos et al. *D4.1.1 – Prototype Metadata Database and Social Network*. PaaSage project deliverable. Mar. 2014.

[15] Thomas Kühne. “Matters of (meta-)modeling”. In: *Software and Systems Modeling* 5.4 (2006), pp. 369–385. DOI: 10.1007/s10270-006-0017-9.

[16] Object Management Group. *Unified Modeling Language Specification*. 2.4.1. http://www.omg.org/spec/UML/2.4.1/. Aug. 2011.

[17] *OMG Model-Driven Architecture*. URL: http://www.omg.org/mda/.

[18] Derek Palma and Thomas Spatzier. *Topology and Orchestration Specification for Cloud Applications (TOSCA)*. Tech. rep. Organization for the Advancement of Structured Information Standards (OASIS), June 2013. URL: http://docs.oasis-open.org/tosca/TOSCA/v1.0/cos01/TOSCA-v1.0-cos01.pdf.

[19] 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.

[20] Clément Quinton, Daniel Romero and Laurence Duchien. “Cardinality-based feature models with constraints: a pragmatic approach”. In: *SPLC 2013: 17th International Software Product Line Conference*. Ed. by Tomoji Kishi, Stan Jarzabek and Stefania Gnesi. ACM, 2013, pp. 162–166. ISBN: 978-1-4503-1968-3. DOI: 10.1145/2491627.2491638.

[21] Clément Quinton, Romain Rouvoy and Laurence Duchien. “Leveraging Feature Models to Configure Virtual Appliances”. In: *CloudCP 2012: 2nd International Workshop on Cloud Computing Platforms*. ACM, 2012, 2:1–2:6. ISBN: 978-1-4503-1161-8. DOI: 10.1145/2168697.2168699.

[22] B. Rochwerger et al. “The reservoir model and architecture for open federated cloud computing”. In: *IBM Journal of Research and Development* 53.4 (July 2009), pp. 535–545. ISSN: 0018-8646. DOI: 10.1147/JRD.2009.5429058.

[23] Alessandro Rossini and the PaaSage consortium. *D2.1.3 – CloudML Implementation Documentation - Final version*. PaaSage project deliverable. Oct. 2015 (To appear).

[24] Alessandro Rossini, Adrian Rutle, Yngve Lamo and Uwe Wolter. "A formalisation of the copy-modify-merge approach to version control in MDE". In: *Journal of Logic and Algebraic Programming* 79.7 (2010), pp. 636–658. DOI: 10.1016/j.jlap.2009.10.003.

[25] Alessandro Rossini, Arnor Solberg, Daniel Romero, Jörg Domaschka, Kostas Magoutis, Lutz Schubert, Nicolas Ferry and Tom Kirkham. *D2.1.1 – CloudML Guide and Assesment Report*. PaaSage project deliverable. Oct. 2013.

[26] Alessandro Rossini et al. *D2.1.1e – CloudML Guide and Assesment Report (Extended)*. PaaSage project deliverable. Nov. 2013.

[27] Calin Sandru, Dana Petcu and Victor Ion Munteanu. "Building an Open-Source Platform-as-a-Service with Intelligent Management of Multiple Cloud Resources". In: *UCC 2012: IEEE 5th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2012, pp. 333–338. ISBN: 978-1-4673-4432-6. DOI: 10.1109/UCC.2012.54.

[28] Mohamed Sellami, Sami Yangui, Mohamed Mohamed and Samir Tata. "PaaS-Independent Provisioning and Management of Applications in the Cloud". In: *CLOUD 2013: IEEE 6th International Conference on Cloud Computing*. Ed. by Lisa O'Conner. IEEE Computer Society, 2013, pp. 693–700. ISBN: 978-0-7695-5028-2. DOI: 10.1109/CLOUD.2013.105.

[29] Daniel Seybold. "Design und Implementierung eines skalierenden Database-as-a-Service Systems (in German)". Mastersthesis VS-M05-2014. Institute for Distributed Systems, University of Ulm, Apr. 2014.

[30] Jin Shao, Hao Wei, Qianxiang Wang and Hong Mei. "A Runtime Model Based Monitoring Approach for Cloud". In: *CLOUD 2010: IEEE 3rd International Conference on Cloud Computing*. IEEE Computer Society, 2010, pp. 313–320. ISBN: 978-1-4244-8207-8. DOI: 10.1109/CLOUD.2010.31.

[31] Clemens Szyperski. *Component software: beyond object-oriented programming (2nd edition)*. Pearson Education, 2011. ISBN: 978-0321753021.

A XMI Serialisation of the SENSAPP Example

Listing A.1: XMI Serialisation of the SensApp Example

```
<?xml version="1.0" encoding="UTF-8"?>
<net.cloudml:CloudMLModel xmi:version="2.0" xmlns:xmi="http://www.omg.
    org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:net.cloudml="http://cloudml.net" name="SensAppModel">
  <providers name="Beanstalk" credentials="./credentials_beanstalk"/>
  <providers name="FlexiScale" credentials="./credentials_flexiscale"/
      >
  <providers name="AmazonEC2" credentials="./credentials_aws"/>
  <components xsi:type="net.cloudml:ExternalComponent" name="
      BeanstalkContainer" provider="Beanstalk" location="eu">
    <providedContainmentPorts name="webServer" owner="
        BeanstalkContainer" offers="language">
      <properties name="language" value="Java"/>
    </providedContainmentPorts>
  </components>
  <components xsi:type="net.cloudml:InternalComponent" name="SensApp">
    <resources name="sensAppWar" downloadCommand="wget␣-P␣~␣http://
        github.com/downloads/SINTEF-9012/sensapp/sensapp.war;␣wget␣-P␣~
        ␣http://ec2-54-228-116-115.eu-west-1.compute.amazonaws.com/
        scripts/linux/ubuntu/sensapp/sensapp.sh" installCommand="cd␣~;␣
        sudo␣bash␣sensapp.sh"/>
    <providedCommunicationPorts name="RESTProvided" component="SensApp
        " portNumber="8080"/>
    <requiredCommunicationPorts name="mongoDBRequired" component="
        SensApp" isMandatory="true"/>
    <requiredContainmentPort name="servletContainerRequired" owner="
        SensApp"/>
  </components>
  <components xsi:type="net.cloudml:InternalComponent" name="
      SensAppAdmin">
    <resources name="sensAppAdminWar" downloadCommand="wget␣-P␣~␣http:
        //ec2-54-228-116-115.eu-west-1.compute.amazonaws.com/resources/
        sensappAdmin/SensAppGUI.tar;␣wget␣-P␣~␣http://ec2
        -54-228-116-115.eu-west-1.compute.amazonaws.com/scripts/linux/
        ubuntu/sensappAdmin/startsensappgui.sh␣;␣wget␣-P␣~␣http://ec2
        -54-228-116-115.eu-west-1.compute.amazonaws.com/scripts/linux/
        ubuntu/sensappAdmin/sensappgui.sh␣;␣wget␣-P␣~␣http://ec2
        -54-228-116-115.eu-west-1.compute.amazonaws.com/resources/
        sensappAdmin/localTopology.json;␣wget␣http://ec2
        -54-228-116-115.eu-west-1.compute.amazonaws.com/resources/
        sources.list;␣sudo␣mv␣sources.list␣/etc/apt/sources.list"
        installCommand="cd␣~;␣sudo␣bash␣sensappgui.sh" startCommand="cd
        ␣~;␣sudo␣bash␣startsensappgui.sh"/>
    <requiredCommunicationPorts name="RESTRequired" component="
        SensAppAdmin" portNumber="8080"/>
    <requiredContainmentPort name="beanstalkRequired" owner="SensApp"/
        >
  </components>
  <components xsi:type="net.cloudml:InternalComponent" name="JettySC">
    <resources name="jettyBin" downloadCommand="wget␣-P␣~␣http://ec2
        -54-228-116-115.eu-west-1.compute.amazonaws.com/scripts/linux/
        ubuntu/jetty/jetty.sh" installCommand="cd␣~;␣sudo␣bash␣jetty.sh
        "/>
    <providedContainmentPorts name="servletContainerProvided" owner="
        JettySC"/>
```

```
  <requiredContainmentPort name="slRequired" owner="JettySC"/>
</components>
<components xsi:type="net.cloudml:InternalComponent" name="MongoDB">
  <resources name="mongoDBBin" downloadCommand="wget␣-P␣~␣http://ec2
      -54-228-116-115.eu-west-1.compute.amazonaws.com/scripts/linux/
      ubuntu/mongoDB/mongoDB.sh" installCommand="cd␣~;␣sudo␣bash␣
      mongoDB.sh"/>
  <providedCommunicationPorts name="mongoDB" component="MongoDB"/>
  <requiredContainmentPort name="mlRequired" owner="MongoDB"/>
</components>
<components xsi:type="net.cloudml:VM" name="ML" provider="AmazonEC2"
     location="eu-west-1b" minRam="4096" minCores="2" maxStorage="50"
     os="ubuntu" securityGroup="SensApp" sshKey="cloudml" privateKey=
    "/Users/aronnax/SINTEF/cloudml/bin/cloudml.pem" groupName="
    sensapp">
  <providedContainmentPorts name="mlProvided" owner="ML"/>
</components>
<components xsi:type="net.cloudml:VM" name="SL" provider="FlexiScale
    " location="no" minRam="1024" minCores="1" maxStorage="50" os="
    ubuntu" is64os="false" imageId="Ubuntu-SINTEF" securityGroup="
    SensApp" sshKey="cloudml" privateKey="./cloudml.pem" groupName="
    SensApp">
  <providedContainmentPorts name="slProvided" owner="SL"/>
</components>
<communications name="SensAppMongoDB" requiredCommunicationPort="
    mongoDBRequired" providedCommunicationPort="mongoDB"/>
<communications name="SensAppAdminSensApp" requiredCommunicationPort
    ="RESTRequired" providedCommunicationPort="RESTProvided">
  <resources name="client" downloadCommand="wget␣-P␣~␣http://ec2
      -54-228-116-115.eu-west-1.compute.amazonaws.com/scripts/linux/
      ubuntu/sensappAdmin/configuresensappgui.sh" configureCommand="
      cd␣~;␣sudo␣bash␣configuresensappgui.sh"/>
</communications>
<clouds name="Flexiant"/>
<clouds name="AWS-EC2"/>
<componentInstances xsi:type="net.cloudml:InternalComponentInstance"
     name="SensApp1" type="SensApp">
  <providedCommunicationPortInstances name="RESTProvided1" type="
      RESTProvided" componentInstance="SensApp1"/>
  <requiredCommunicationPortInstances name="mongoDBRequired1" type="
      mongoDBRequired" componentInstance="SensApp1"/>
  <requiredContainmentPortInstance name="servletContainerRequired1"
      owner="SensApp1" type="servletContainerRequired"/>
</componentInstances>
<componentInstances xsi:type="net.cloudml:InternalComponentInstance"
     name="SensAppAdmin1" type="SensAppAdmin">
  <requiredCommunicationPortInstances name="RESTRequired1" type="
      RESTRequired" componentInstance="SensAppAdmin1"/>
  <requiredContainmentPortInstance name="beanstalkRequired1" owner="
      SensAppAdmin1" type="beanstalkRequired"/>
</componentInstances>
<componentInstances xsi:type="net.cloudml:InternalComponentInstance"
     name="JettySC1" type="JettySC">
  <providedContainmentPortInstances name="servletContainerProvided1"
       owner="JettySC1" type="servletContainerProvided"/>
  <requiredContainmentPortInstance name="slRequired1" owner="
      JettySC1" type="slRequired"/>
</componentInstances>
<componentInstances xsi:type="net.cloudml:InternalComponentInstance"
     name="MongoDB1" type="MongoDB">
```

```
    <providedCommunicationPortInstances name="mongoDB1" type="mongoDB"
         componentInstance="MongoDB1"/>
    <requiredContainmentPortInstance name="mlRequired1" owner="
        MongoDB1" type="mlRequired"/>
  </componentInstances>
  <componentInstances xsi:type="net.cloudml:ExternalComponentInstance"
       name="beanstalkContainer1" type="BeanstalkContainer">
    <providedContainmentPortInstances name="webServer1" owner="
        beanstalkContainer1" type="webServer"/>
  </componentInstances>
  <communicationInstances name="sensAppMongoDB1" type="SensAppMongoDB"
       requiredCommunicationPortInstance="mongoDBRequired1"
      providedCommunicationPortInstance="mongoDB1"/>
  <communicationInstances name="sensAppAdminSensApp1" type="
      SensAppAdminSensApp" requiredCommunicationPortInstance="
      RESTRequired1" providedCommunicationPortInstance="RESTProvided1">
    <resources name="client" downloadCommand="wget␣-P␣~␣http://ec2
        -54-228-116-115.eu-west-1.compute.amazonaws.com/scripts/linux/
        ubuntu/sensappAdmin/configuresensappgui.sh" configureCommand="
        sudo␣bash␣configuresensappgui.sh"/>
  </communicationInstances>
  <containmentInstances name="sensApp2SC1"
      providedContainmentPortInstance="servletContainerProvided1"
      requiredContainmentPortInstance="servletContainerRequired1" type=
      "sensApp2SC"/>
  <containmentInstances name="jetty2SL1"
      providedContainmentPortInstance="slProvided1"
      requiredContainmentPortInstance="slRequired1" type="jetty2SL"/>
  <containmentInstances name="sensAppAdmin2Beanstalk1"
      providedContainmentPortInstance="webServer1"
      requiredContainmentPortInstance="beanstalkRequired1" type="
      sensAppAdmin2Beanstalk"/>
  <containmentInstances name="mongoDB2ML1"
      providedContainmentPortInstance="mlProvided1"
      requiredContainmentPortInstance="mlRequired1" type="mongoDB2ML"/>
  <containments name="sensApp2SC" providedContainmentPort="
      servletContainerProvided" requiredContainmentPort="
      servletContainerRequired"/>
  <containments name="jetty2SL" providedContainmentPort="slProvided"
      requiredContainmentPort="slRequired">
    <resources name="configureJetty" downloadCommand="wget␣-P␣~␣http:
        //cloudml.org/scripts/linux/ubuntu/sensappAdmin/configure_jetty
        .sh" configureCommand="sudo␣bash␣configure_jetty.sh"/>
  </containments>
  <containments name="sensAppAdmin2Beanstalk" providedContainmentPort=
      "webServer" requiredContainmentPort="beanstalkRequired"/>
  <containments name="mongoDB2ML" providedContainmentPort="mlProvided"
       requiredContainmentPort="mlRequired"/>
  <vmInstances name="ml1" type="ML" publicAddress="">
    <providedContainmentPortInstances name="mlProvided1" owner="ml1"
        type="mlProvided"/>
  </vmInstances>
  <vmInstances name="sl1" type="SL" publicAddress="">
    <providedContainmentPortInstances name="slProvided1" owner="sl1"
        type="slProvided"/>
  </vmInstances>
</net.cloudml:CloudMLModel>
```

B Cloud Ontology Diagram

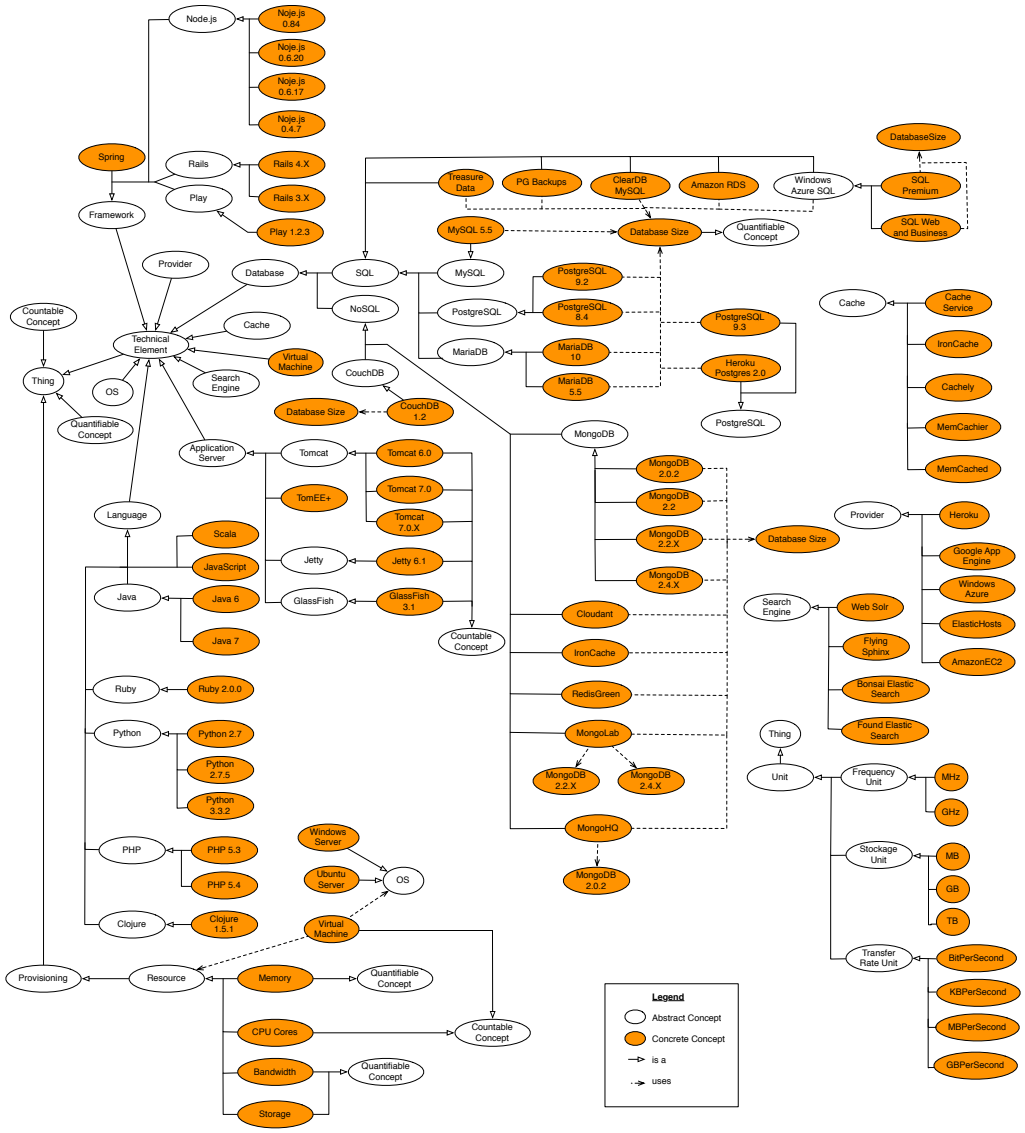

Figure 31: Saloon Cloud Ontology Diagram

C Full Java Code of the SENSAPP Example

Listing C.1: Full SensApp Example Code

```
// initialise and activate a container
final IManagedContainer container = ContainerUtil.createContainer();
Net4jUtil.prepareContainer(container);
TCPUtil.prepareContainer(container);
CDONet4jUtil.prepareContainer(container);
container.activate();

// create a Net4j TCP connector
final IConnector connector = (IConnector) TCPUtil.getConnector(
    container, "localhost:2036");

// create the session configuration
CDONet4jSessionConfiguration config = CDONet4jUtil.
    createNet4jSessionConfiguration();
config.setConnector(connector);
config.setRepositoryName("CDORepositoryName");

// create the actual session with the repository
CDONet4jSession cdoSession = config.openNet4jSession();

// obtain a transaction object
CDOTransaction transaction = cdoSession.openTransaction();

// create a CDO resource object
CDOResource resource = transaction.getOrCreateResource("/
    sensAppResource");

// complete mapping of the SensApp example
CloudMLModel model = CloudmlFactory.eINSTANCE.createCloudMLModel();
model.setName("SensAppCPSM");

Provider beanstalk = CloudmlFactory.eINSTANCE.createProvider();
beanstalk.setName("Beanstalk");
beanstalk.setCredentials("./credentials_beanstalk");

Provider flexiScale = CloudmlFactory.eINSTANCE.createProvider();
flexiScale.setName("FlexiScale");
flexiScale.setCredentials("./credentials_flexiscale");

Provider amazonEC2 = CloudmlFactory.eINSTANCE.createProvider();
amazonEC2.setName("AWS-EC2");
amazonEC2.setCredentials("./credentials_amazon");

ExternalComponent beanstalkContainer = CloudmlFactory.eINSTANCE.
    createExternalComponent();
beanstalkContainer.setName("BeanstalkContainer");
beanstalkContainer.setProvider(beanstalk);
beanstalkContainer.setLocation("eu");

ProvidedContainmentPort webServer = CloudmlFactory.eINSTANCE.
    createProvidedContainmentPort();
webServer.setName("webServer");
webServer.setOwner(beanstalkContainer);
```

```
Property webServerLanguage = CloudmlFactory.eINSTANCE.createProperty()
    ;
webServerLanguage.setName("language");
webServerLanguage.setValue("Java");

webServer.getProperties().add(webServerLanguage);
beanstalkContainer.getProvidedContainmentPorts().add(webServer);

InternalComponent sensApp = CloudmlFactory.eINSTANCE.
    createInternalComponent();
sensApp.setName("SensApp");

Resource sensAppWar = CloudmlFactory.eINSTANCE.createResource();
sensAppWar.setDownloadCommand(""
    + "wget␣-P␣~␣http://github.com/downloads/SINTEF-9012/sensapp/
        sensapp.war;␣"
    + "wget␣-P␣~␣http://ec2-54-228-116-115.eu-west-1.compute.amazonaws
        .com/scripts/linux/ubuntu/sensapp/sensapp.sh"
    );
sensAppWar.setInstallCommand("cd␣~;␣sudo␣bash␣sensapp.sh");
sensAppWar.setName("sensAppWar");

ProvidedCommunicationPort restProvided = CloudmlFactory.eINSTANCE.
    createProvidedCommunicationPort();
restProvided.setName("RESTProvided");
restProvided.setIsLocal(false);
restProvided.setPortNumber(8080);
restProvided.setComponent(sensApp);

RequiredCommunicationPort mongoDBRequired = CloudmlFactory.eINSTANCE.
    createRequiredCommunicationPort();
mongoDBRequired.setName("mongoDBRequired");
mongoDBRequired.setIsLocal(true);
mongoDBRequired.setIsMandatory(true);
mongoDBRequired.setPortNumber(0);
mongoDBRequired.setComponent(sensApp);

RequiredContainmentPort servletContainerRequired = CloudmlFactory.
    eINSTANCE.createRequiredContainmentPort();
servletContainerRequired.setName("servletContainerRequired");
servletContainerRequired.setOwner(sensApp);

sensApp.getResources().add(sensAppWar);
sensApp.getProvidedCommunicationPorts().add(restProvided);
sensApp.getRequiredCommunicationPorts().add(mongoDBRequired);
sensApp.setRequiredContainmentPort(servletContainerRequired);


InternalComponent sensAppAdmin = CloudmlFactory.eINSTANCE.
    createInternalComponent();
sensAppAdmin.setName("SensAppAdmin");

Resource sensAppAdminWar = CloudmlFactory.eINSTANCE.createResource();
sensAppAdminWar.setName("sensAppAdminWar");
sensAppAdminWar.setDownloadCommand(""
    + "wget␣-P␣~␣http://ec2-54-228-116-115.eu-west-1.compute.amazonaws
        .com/resources/sensappAdmin/SensAppGUI.tar;␣"
    + "wget␣-P␣~␣http://ec2-54-228-116-115.eu-west-1.compute.amazonaws
        .com/scripts/linux/ubuntu/sensappAdmin/startsensappgui.sh␣;␣"
```

```
    + "wget␣-P␣~␣http://ec2-54-228-116-115.eu-west-1.compute.amazonaws
        .com/scripts/linux/ubuntu/sensappAdmin/sensappgui.sh␣;␣"
    + "wget␣-P␣~␣http://ec2-54-228-116-115.eu-west-1.compute.amazonaws
        .com/resources/sensappAdmin/localTopology.json;␣"
    + "wget␣http://ec2-54-228-116-115.eu-west-1.compute.amazonaws.com/
        resources/sources.list;␣"
    + "sudo␣mv␣sources.list␣/etc/apt/sources.list"
    );
sensAppAdminWar.setInstallCommand("cd␣~;␣sudo␣bash␣sensappgui.sh");
sensAppAdminWar.setStartCommand("cd␣~;␣sudo␣bash␣sensappgui.sh");

RequiredCommunicationPort restRequired = CloudmlFactory.eINSTANCE.
    createRequiredCommunicationPort();
restRequired.setName("RESTRequired");
restRequired.setIsLocal(false);
restRequired.setIsMandatory(false);
restRequired.setPortNumber(8080);
restRequired.setComponent(sensAppAdmin);

RequiredContainmentPort beanstalkRequired = CloudmlFactory.eINSTANCE.
    createRequiredContainmentPort();
beanstalkRequired.setName("beanstalkRequired");
beanstalkRequired.setOwner(sensAppAdmin);

sensAppAdmin.getResources().add(sensAppAdminWar);
sensAppAdmin.getRequiredCommunicationPorts().add(restRequired);
sensAppAdmin.setRequiredContainmentPort(beanstalkRequired);

InternalComponent jettySC = CloudmlFactory.eINSTANCE.
    createInternalComponent();
jettySC.setName("JettySC");

Resource jettyBin = CloudmlFactory.eINSTANCE.createResource();
jettyBin.setName("jettyBin");
jettyBin.setDownloadCommand("wget␣-P␣~␣http://ec2-54-228-116-115.eu-
    west-1.compute.amazonaws.com/scripts/linux/ubuntu/jetty/jetty.sh");
jettyBin.setInstallCommand("cd␣~;␣sudo␣bash␣jetty.sh");

ProvidedContainmentPort servletContainerProvided = CloudmlFactory.
    eINSTANCE.createProvidedContainmentPort();
servletContainerProvided.setName("servletContainerProvided");
servletContainerProvided.setOwner(jettySC);

RequiredContainmentPort slRequired = CloudmlFactory.eINSTANCE.
    createRequiredContainmentPort();
slRequired.setName("slRequired");
slRequired.setOwner(jettySC);

jettySC.getResources().add(jettyBin);
jettySC.getProvidedContainmentPorts().add(servletContainerProvided);
jettySC.setRequiredContainmentPort(slRequired);

InternalComponent mongoDB = CloudmlFactory.eINSTANCE.
    createInternalComponent();
mongoDB.setName("MongoDB");

Resource mongoDBBin = CloudmlFactory.eINSTANCE.createResource();
mongoDBBin.setName("mongoDBBin");
```

```
mongoDBBin.setDownloadCommand("wget␣-P␣~␣http://ec2-54-228-116-115.eu-
    west-1.compute.amazonaws.com/scripts/linux/ubuntu/mongoDB/mongoDB.
    sh");
mongoDBBin.setInstallCommand("cd␣~;␣sudo␣bash␣mongoDB.sh");

ProvidedCommunicationPort mongoDBPP = CloudmlFactory.eINSTANCE.
    createProvidedCommunicationPort();
mongoDBPP.setName("mongoDB");
mongoDBPP.setIsLocal(true);
mongoDBPP.setPortNumber(0);
mongoDBPP.setComponent(mongoDB);

RequiredContainmentPort mlRequired = CloudmlFactory.eINSTANCE.
    createRequiredContainmentPort();
mlRequired.setName("mlRequired");
mlRequired.setOwner(mongoDB);

mongoDB.getResources().add(mongoDBBin);
mongoDB.getProvidedCommunicationPorts().add(mongoDBPP);
mongoDB.setRequiredContainmentPort(mlRequired);

VM ml = CloudmlFactory.eINSTANCE.createVM();
ml.setName("ML");
ml.setMinCores(4);
ml.setMaxCores(8);
ml.setMinRam(4096);
ml.setMaxRam(0);
ml.setMinStorage(50);
ml.setMaxStorage(100);
ml.setLocation("eu-west-1b");
ml.setOs("ubuntu");
ml.setSshKey("cloudml");
ml.setSecurityGroup("SensApp");
ml.setGroupName("sensapp");
ml.setPrivateKey("/Users/aronnax/SINTEF/cloudml/bin/cloudml.pem");
ml.setIs64os(true);
ml.setProvider(amazonEC2);

ProvidedContainmentPort mlProvided = CloudmlFactory.eINSTANCE.
    createProvidedContainmentPort();
mlProvided.setName("mlProvided");
mlProvided.setOwner(ml);

ml.getProvidedContainmentPorts().add(mlProvided);

VM sl = CloudmlFactory.eINSTANCE.createVM();
sl.setName("SL");
sl.setMinCores(1);
sl.setMaxCores(0);
sl.setMinRam(1024);
sl.setMaxRam(0);
sl.setMinStorage(50);
sl.setMaxStorage(0);
sl.setLocation("no");
sl.setOs("ubuntu");
sl.setSshKey("cloudml");
sl.setSecurityGroup("SensApp");
sl.setGroupName("SensApp");
sl.setPrivateKey("./cloudml.pem");
```

```
sl.setImageId("Ubuntu-SINTEF");
sl.setIs64os(false);
sl.setProvider(flexiScale);

ProvidedContainmentPort slProvided = CloudmlFactory.eINSTANCE.
    createProvidedContainmentPort();
slProvided.setName("slProvided");
slProvided.setOwner(sl);

sl.getProvidedContainmentPorts().add(slProvided);

Communication sensAppMongoDb = CloudmlFactory.eINSTANCE.
    createCommunication();
sensAppMongoDb.setName("SensAppMongoDB");
sensAppMongoDb.setProvidedCommunicationPort(mongoDBPP);
sensAppMongoDb.setRequiredCommunicationPort(mongoDBRequired);

Communication sensAppAdminSensApp = CloudmlFactory.eINSTANCE.
    createCommunication();
sensAppAdminSensApp.setName("SensAppAdminSensApp");
sensAppAdminSensApp.setProvidedCommunicationPort(restProvided);
sensAppAdminSensApp.setRequiredCommunicationPort(restRequired);

Containment sensApp2SC = CloudmlFactory.eINSTANCE.createContainment();
sensApp2SC.setName("sensApp2SC");
sensApp2SC.setProvidedContainmentPort(servletContainerProvided);
sensApp2SC.setRequiredContainmentPort(servletContainerRequired);

Containment jetty2Sl = CloudmlFactory.eINSTANCE.createContainment();
jetty2Sl.setName("jetty2SL");
jetty2Sl.setProvidedContainmentPort(slProvided);
jetty2Sl.setProvidedContainmentPort(slProvided);

Resource configureJetty = CloudmlFactory.eINSTANCE.createResource();
configureJetty.setName("configureJetty");
configureJetty.setConfigureCommand("sudo␣bash␣configure_jetty.sh");
configureJetty.setDownloadCommand("wget␣-P␣~␣http://cloudml.org/
    scripts/linux/ubuntu/sensappAdmin/configure_jetty.sh");

jetty2Sl.getResources().add(configureJetty);

Containment sensAppAdmin2Beanstalk = CloudmlFactory.eINSTANCE.
    createContainment();
sensAppAdmin2Beanstalk.setName("sensAppAdmin2Beanstalk");
sensAppAdmin2Beanstalk.setProvidedContainmentPort(webServer);
sensAppAdmin2Beanstalk.setRequiredContainmentPort(beanstalkRequired);

Containment mongoDb2Ml = CloudmlFactory.eINSTANCE.createContainment();
mongoDb2Ml.setName("mongoDB2ML");
mongoDb2Ml.setProvidedContainmentPort(mlProvided);
mongoDb2Ml.setRequiredContainmentPort(mlRequired);

Resource client = CloudmlFactory.eINSTANCE.createResource();
client.setName("client");
client.setDownloadCommand(""
    + "wget␣-P␣~␣http://ec2-54-228-116-115.eu-west-1.compute.amazonaws
        .com/scripts/linux/ubuntu/sensappAdmin/configuresensappgui.sh"
    );
client.setConfigureCommand("cd␣~;␣sudo␣bash␣configuresensappgui.sh");
```

```
sensAppAdminSensApp.setRequiredPortResource(client);

sensAppAdminSensApp.getResources().add(client);

InternalComponentInstance sensApp1 = CloudmlFactory.eINSTANCE.
    createInternalComponentInstance();
sensApp1.setName("sensApp1");
sensApp1.setType(sensApp);

ProvidedCommunicationPortInstance restProvided1 = CloudmlFactory.
    eINSTANCE.createProvidedCommunicationPortInstance();
restProvided1.setName("RESTProvided1");
restProvided1.setType(restProvided);
restProvided1.setComponentInstance(sensApp1);

RequiredCommunicationPortInstance mongoDbRequired1 = CloudmlFactory.
    eINSTANCE.createRequiredCommunicationPortInstance();
mongoDbRequired1.setName("mongoDBRequired1");
mongoDbRequired1.setType(mongoDBRequired);
mongoDbRequired1.setComponentInstance(sensApp1);

RequiredContainmentPortInstance servletContainerRequired1 =
    CloudmlFactory.eINSTANCE.createRequiredContainmentPortInstance();
servletContainerRequired1.setName("servletContainerRequired1");
servletContainerRequired1.setOwner(sensApp1);
servletContainerRequired1.setType(servletContainerRequired);

sensApp1.getProvidedCommunicationPortInstances().add(restProvided1);
sensApp1.getRequiredCommunicationPortInstances().add(mongoDbRequired1)
    ;
sensApp1.setRequiredContainmentPortInstance(servletContainerRequired1)
    ;

InternalComponentInstance sensAppAdmin1 = CloudmlFactory.eINSTANCE.
    createInternalComponentInstance();
sensAppAdmin1.setName("sensAppAdmin1");
sensAppAdmin1.setType(sensAppAdmin);

RequiredCommunicationPortInstance restRequired1 = CloudmlFactory.
    eINSTANCE.createRequiredCommunicationPortInstance();
restRequired1.setName("RESTRequired");
restRequired1.setType(restRequired);
restRequired1.setComponentInstance(sensAppAdmin1);

RequiredContainmentPortInstance beanstalkRequired1 = CloudmlFactory.
    eINSTANCE.createRequiredContainmentPortInstance();
beanstalkRequired1.setName("beanstalkRequired1");
beanstalkRequired1.setOwner(sensAppAdmin1);
beanstalkRequired1.setType(beanstalkRequired);

sensAppAdmin1.getRequiredCommunicationPortInstances().add(
    restRequired1);
sensAppAdmin1.setRequiredContainmentPortInstance(beanstalkRequired1);

InternalComponentInstance jettySc1 = CloudmlFactory.eINSTANCE.
    createInternalComponentInstance();
jettySc1.setName("jettySC1");
jettySc1.setType(jettySC);
```

```
ProvidedContainmentPortInstance servletContainerProvided1 =
    CloudmlFactory.eINSTANCE.createProvidedContainmentPortInstance();
servletContainerProvided1.setName("servletContainerProvided1");
servletContainerProvided1.setOwner(jettySc1);
servletContainerProvided1.setType(servletContainerProvided);

RequiredContainmentPortInstance slRequired1 = CloudmlFactory.eINSTANCE
    .createRequiredContainmentPortInstance();
slRequired1.setName("slRequired1");
slRequired1.setOwner(jettySc1);
slRequired1.setType(slRequired);

jettySc1.getProvidedContainmentPortInstances().add(
    servletContainerProvided1);
jettySc1.setRequiredContainmentPortInstance(slRequired1);

InternalComponentInstance mongoDb1 = CloudmlFactory.eINSTANCE.
    createInternalComponentInstance();
mongoDb1.setName("MongoDB1");
mongoDb1.setType(mongoDB);

ProvidedCommunicationPortInstance mongoDb1PPI = CloudmlFactory.
    eINSTANCE.createProvidedCommunicationPortInstance();
mongoDb1PPI.setName("mongoDB1");
mongoDb1PPI.setType(mongoDBPP);
mongoDb1PPI.setComponentInstance(mongoDb1);

RequiredContainmentPortInstance mlRequired1 = CloudmlFactory.eINSTANCE
    .createRequiredContainmentPortInstance();
mlRequired1.setName("mlRequired1");
mlRequired1.setOwner(mongoDb1);
mlRequired1.setType(mlRequired);

mongoDb1.getProvidedCommunicationPortInstances().add(mongoDb1PPI);
mongoDb1.setRequiredContainmentPortInstance(mlRequired1);

ExternalComponentInstance beanstalkContainer1 = CloudmlFactory.
    eINSTANCE.createExternalComponentInstance();
beanstalkContainer1.setName("beanstalkContainer1");
beanstalkContainer1.setType(beanstalkContainer);

ProvidedContainmentPortInstance webServer1 = CloudmlFactory.eINSTANCE.
    createProvidedContainmentPortInstance();
webServer1.setName("webServer1");
webServer1.setOwner(beanstalkContainer1);
webServer1.setType(webServer);

beanstalkContainer1.getProvidedContainmentPortInstances().add(
    webServer1);

VMInstance sl1 = CloudmlFactory.eINSTANCE.createVMInstance();
sl1.setName("PaaSage-SL1");
sl1.setType(sl);

ProvidedContainmentPortInstance slProvided1 = CloudmlFactory.eINSTANCE
    .createProvidedContainmentPortInstance();
slProvided1.setName("slProvided1");
slProvided1.setOwner(sl1);
slProvided1.setType(slProvided);
```

```
sl1.getProvidedContainmentPortInstances().add(slProvided1);

VMInstance ml1 = CloudmlFactory.eINSTANCE.createVMInstance();
ml1.setName("PaaSage-ML1");
ml1.setType(ml);

ProvidedContainmentPortInstance mlProvided1 = CloudmlFactory.eINSTANCE
    .createProvidedContainmentPortInstance();
mlProvided1.setName("mlProvided1");
mlProvided1.setOwner(ml1);
mlProvided1.setType(mlProvided);

ml1.getProvidedContainmentPortInstances().add(mlProvided1);

Cloud flexiant = CloudmlFactory.eINSTANCE.createCloud();
flexiant.setName("Flexiant");
flexiant.getExternalComponents().add(sl);

Cloud aws_ec2 = CloudmlFactory.eINSTANCE.createCloud();
aws_ec2.setName("AWS-EC2");
aws_ec2.getExternalComponents().add(ml);

CommunicationInstance sensAppMongoDb1 = CloudmlFactory.eINSTANCE.
    createCommunicationInstance();
sensAppMongoDb1.setName("sensAppMongoDb1");
sensAppMongoDb1.setType(sensAppMongoDb);
sensAppMongoDb1.setProvidedCommunicationPortInstance(mongoDb1PPI);
sensAppMongoDb1.setRequiredCommunicationPortInstance(mongoDbRequired1)
    ;

CommunicationInstance sensAppAdminSensApp1 = CloudmlFactory.eINSTANCE.
    createCommunicationInstance();
sensAppAdminSensApp1.setName("sensAppAdminSensApp1");
sensAppAdminSensApp1.setType(sensAppAdminSensApp);
sensAppAdminSensApp1.setProvidedCommunicationPortInstance(
    restProvided1);
sensAppAdminSensApp1.setRequiredCommunicationPortInstance(
    restRequired1);

ContainmentInstance sensApp2SC1 = CloudmlFactory.eINSTANCE.
    createContainmentInstance();
sensApp2SC1.setName("sensApp2SC1");
sensApp2SC1.setProvidedContainmentPortInstance(
    servletContainerProvided1);
sensApp2SC1.setRequiredContainmentPortInstance(
    servletContainerRequired1);

ContainmentInstance jetty2SL1 = CloudmlFactory.eINSTANCE.
    createContainmentInstance();
jetty2SL1.setName("jetty2SL1");
jetty2SL1.setProvidedContainmentPortInstance(slProvided1);
jetty2SL1.setRequiredContainmentPortInstance(slRequired1);

ContainmentInstance sensAppAdmin2Beanstalk1 = CloudmlFactory.eINSTANCE
    .createContainmentInstance();
sensAppAdmin2Beanstalk1.setName("sensAppAdmin2Beanstalk1");
sensAppAdmin2Beanstalk1.setProvidedContainmentPortInstance(webServer1)
    ;
```

```
sensAppAdmin2Beanstalk1.setRequiredContainmentPortInstance(
    beanstalkRequired1);

ContainmentInstance mongoDb2Ml1 = CloudmlFactory.eINSTANCE.
    createContainmentInstance();
mongoDb2Ml1.setName("mongoDB2ML1");
mongoDb2Ml1.setProvidedContainmentPortInstance(mlProvided1);
mongoDb2Ml1.setRequiredContainmentPortInstance(mlRequired1);

model.getProviders().add(beanstalk);
model.getProviders().add(flexiScale);
model.getProviders().add(amazonEC2);

model.getComponents().add(beanstalkContainer);
model.getComponents().add(sensApp);
model.getComponents().add(sensAppAdmin);
model.getComponents().add(jettySC);
model.getComponents().add(mongoDB);
model.getComponents().add(sl);
model.getComponents().add(ml);

model.getCommunications().add(sensAppMongoDb);
model.getCommunications().add(sensAppAdminSensApp);

model.getClouds().add(flexiant);
model.getClouds().add(aws_ec2);

model.getVmInstances().add(sl1);
model.getVmInstances().add(ml1);

model.getComponentInstances().add(sensApp1);
model.getComponentInstances().add(sensAppAdmin1);
model.getComponentInstances().add(jettySc1);
model.getComponentInstances().add(mongoDb1);
model.getComponentInstances().add(beanstalkContainer1);

model.getCommunicationInstances().add(sensAppMongoDb1);
model.getCommunicationInstances().add(sensAppAdminSensApp1);

model.getContainments().add(sensApp2SC);
model.getContainments().add(jetty2Sl);
model.getContainments().add(sensAppAdmin2Beanstalk);
model.getContainments().add(mongoDb2Ml);

model.getContainmentInstances().add(sensApp2SC1);
model.getContainmentInstances().add(jetty2SL1);
model.getContainmentInstances().add(sensAppAdmin2Beanstalk1);
model.getContainmentInstances().add(mongoDb2Ml1);

resource.getContents().add(model);

try {
  info=transaction.commit();
  System.out.println(info);
} catch (CommitException e) {

  e.printStackTrace();
}
```