PaaSage

Model Based Cloud Platform Upperware

Deliverable D8.2.1

Open Source Prototype System

Version: 1.0

Project Deliverable

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

Philippe Rohou, European Project Coordinator, ERCIM, +33 4 97 15 53 06, +33 6 28 47 40 72

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:	10 April 2014
Document	
Period covered:	Period II
Deliverable number:	D8.2.1
Deliverable title	Open Source Prototype System
Contractual Date of Delivery:	30 September 2014
Actual Date of Delivery:	7 October 2014
Editor (s):	Geir Horn
Author (s):	Daniel Baur, Amin Bsila, Kamil Figiela, Geir Horn, Tom Kirkham, Kyriakos Kritikos, Nikos Parlavantzas, Daniel Romero, Alessandro Rossini, Anthony Sulistio, Robert Viseur
Reviewer (s):	Pierre Guisset
Participant(s):	
Work package no.:	WP8
Work package title:	Exploitation
Work package leader:	Stéphane Waha
Distribution:	Public
Version/Revision:	1.0
Draft/Final:	Final
Total number of pages (including cover):	29

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

This document is a introduction to the PAASAGE software platform and the various components developed for the first PAASAGE prototype. It presents the overall objectives of the software and the global architecture, before presenting the minimal work flow implemented for the first prototype. It is not a complete guide to the PAASAGE software, because the real deliverable documented by this brief guide is a *prototype*, i.e. it is the software itself that is the deliverable.

Intended Audience

This document is written for an application developer who wants to download and test the PAASAGE software for cloud deployment. It aims at providing the necessary *overview* to understand PAASAGE and its objectives, and the context of the developed prototype, without any knowledge required of other PAASAGE deliverables. However, the extensive background necessary to fully understand the components and the implementation is found in the related and referenced PAASAGE deliverables.

Structure of the document

A summary of the PAASAGE's software architecture is given first, before the scope of the first prototype is presented together with the implemented components. Then there is a section describing how to install the PAASAGE virtual machine and the dependencies, and how to run PAASAGE on an application model.

1 Architecture

The PAASAGE perspective is to be a tool for an application developer to master the complex and often error prone task of deploying an application to the Cloud. The application developer could learn the interface and features of one Cloud provider, but it will be very costly to master the development to many providers. It is a real challenge to orchestrate the simultaneous deployment to many different Clouds at the same time as would be the case if some parts of the application can only run on private Cloud resources for confidentiality reasons while one would like to use public Cloud providers for application scalability. The main objective of PAASAGE is to assist the developer with difficult deployment scenarios through *autonomic cloud deployment*. Figure 1 shows a very high level view of PAASAGE and the scope covered by the project.

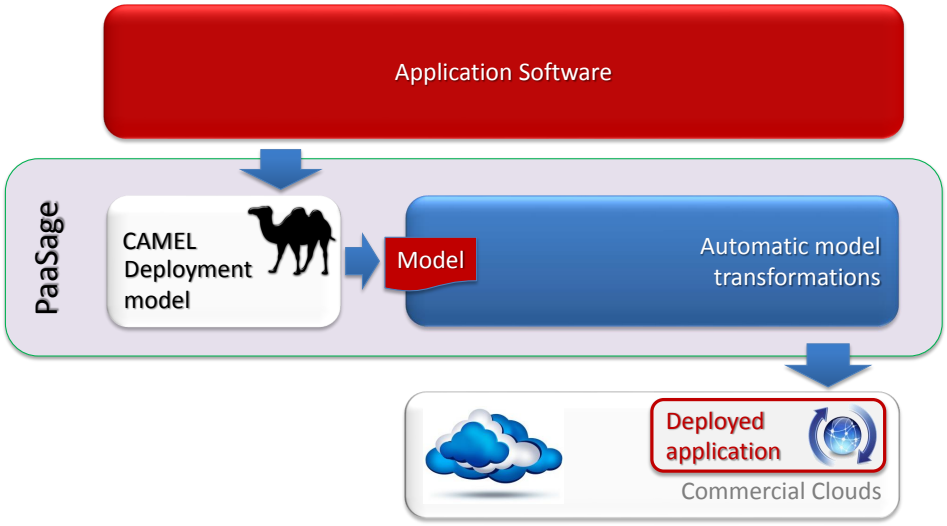

Figure 1: The scope of PAASAGE is to extend the application model with platform annotations and user's goals and preference to a Cloud Application Modelling and Execution Language (CAMEL) model, which is then transformed by PAASAGE to a deployed application in one or more Clouds. Proprietary elements that stays proprietary throughout the use of the open source PAASAGE platform are indicated in red.

To support the developer, PAASAGE needs not only a model of the application to be deployed, but models of the features of the available Cloud platforms, and goals and preferences to be satisfied by the deployment like response times or deployment cost budgets. These different models are specified in *domain specific languages* (DSLs) commonly referred to as the *CAMEL model*. CAMEL integrates the various DSLs using the Eclipse Modelling Framework[1] (EMF) on top of the Connected Data Objects[2] (CDO) persistence solution to maintain model information between different application deployments. The CDO repository is referred to as the *metadata database*, and the intention is that different PAASAGE users will be able to form a "social network" and exchange models, e.g. one user has developed a parametrised model for a particular Cloud provider which can be shared with the other users of the PAASAGE and immediately allows all PAASAGE users to deploy to this Cloud provider. Please refer to the PAASAGE *Deliverable D2.1.2 CloudML Implementation Documentation (First version)* [1] for further details on the modelling concepts, and *Deliverable D4.1.1 & D4.1.3 Prototype Metadata Database and Social Network and Integration Extension* [2] for information on the metadata database.

The application's CAMEL model is first transformed by what is referred to as the *upper ware*. The main purpose of this set of components is to derive a specific deployment *configuration* satisfying all the constraints and goals for the deployment set by the user in the CAMEL model. This implies selecting one or more Cloud providers and generating the necessary deployment scripts. These scripts are then passed to the PAASAGE *execution ware* responsible for instantiating the different parts of the application on the selected Cloud providers

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

and monitor a set of defined metrics in order to make autonomous scalability decisions within the boundaries of the deployment configuration. If the application execution triggers conditions that cannot be satisfied by the current deployment configuration, the execution ware will pass control back to the upper ware to find a better configuration for the current application context. The monitored metrics will subsequently be consolidated as statistical knowledge in the metadata database to guide the decisions on future deployment configurations.

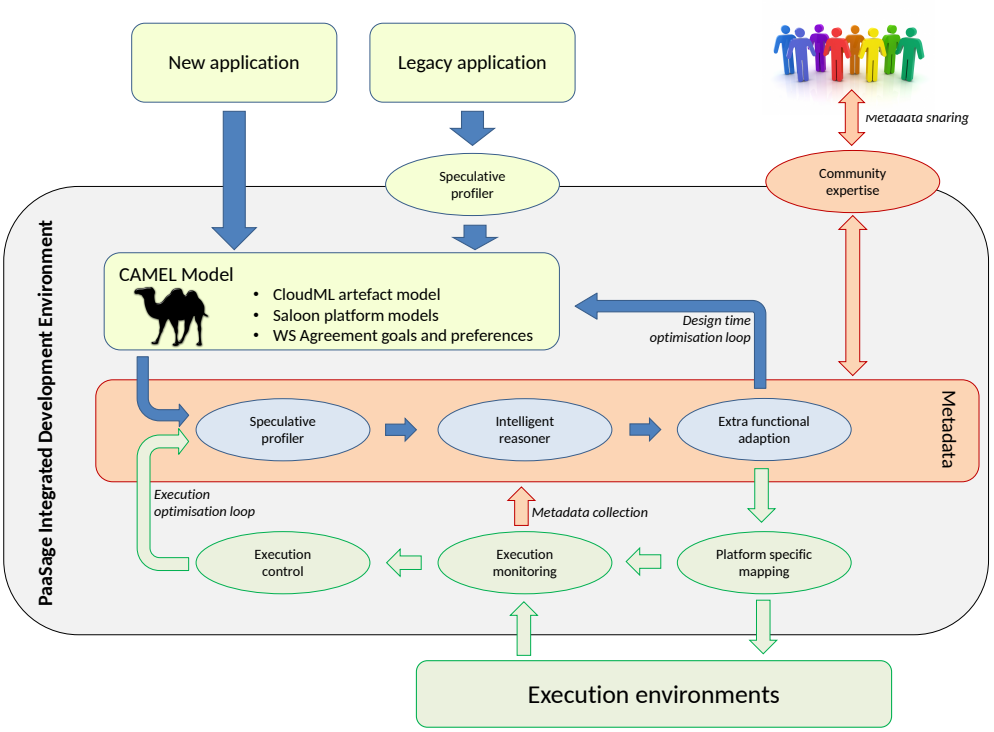

Figure 2: The overall PAASAGE architecture and workflow. Yellow parts are concerned with modelling, blue parts are the upper ware, orange parts are the persistent metadata structure, green parts are the execution ware.

This *feedback loop* controlling the application deployment is depicted in Figure 2 showing the PAASAGE work flow and the different logical parts of the upper ware and the execution ware. The upper ware consists of a *profiler* whose task is to combine information from the different CAMEL DSLs and produce a consistent model based profile of the deployment scenarios. This model is passed on to the *reasoner* part that finds a deployment configuration that satisfies all constraints and optimises[3] the goals set for the deployment by the developer. The *adapter* produces the Cloud platform specific deployment scripts. It also maintains a casual connection between the running application and the model, and reacts to context changes by issuing commands to keep the deployment within the current deployment configuration found by the reasoner. The execution ware has one part taking care of the actual mapping of the deployment configuration onto the Cloud platforms chosen by the reasoner. This includes setting up the necessary resources and installing both the monitoring infrastructure and the application components. The execution is then *monitored* and based on the observed values, the *execution control* will then make autonomous scalability decisions within the boundary conditions given by the deployment configuration, or pass control back to the upper ware to produce a new deployment configuration that better suits the current execution context.

Several components have been identified in order to implement the upper ware in a flexible way. The architecture of the upper ware is shown in Figure 3, and the different components are described in PAASAGE *Deliverable D3.1.1 Prototype Upperware Report* [3]. Similarly, the architecture of the execution ware is shown in Figure 4, and the detailed description of these components can be found in PAASAGE *Deliverable D5.1.1 & D5.3.1 Prototype Executionware, Prototype New Execution Engines* [4] It should be noted that realising the complete architecture is work in progress, and that a subset of these components have been implemented to produce the first prototype, as described in the next section.

[3]Note that PAASAGE will iteratively optimise the found solutions, so the "optimal" solution is here to be understood as the best solution found so far.

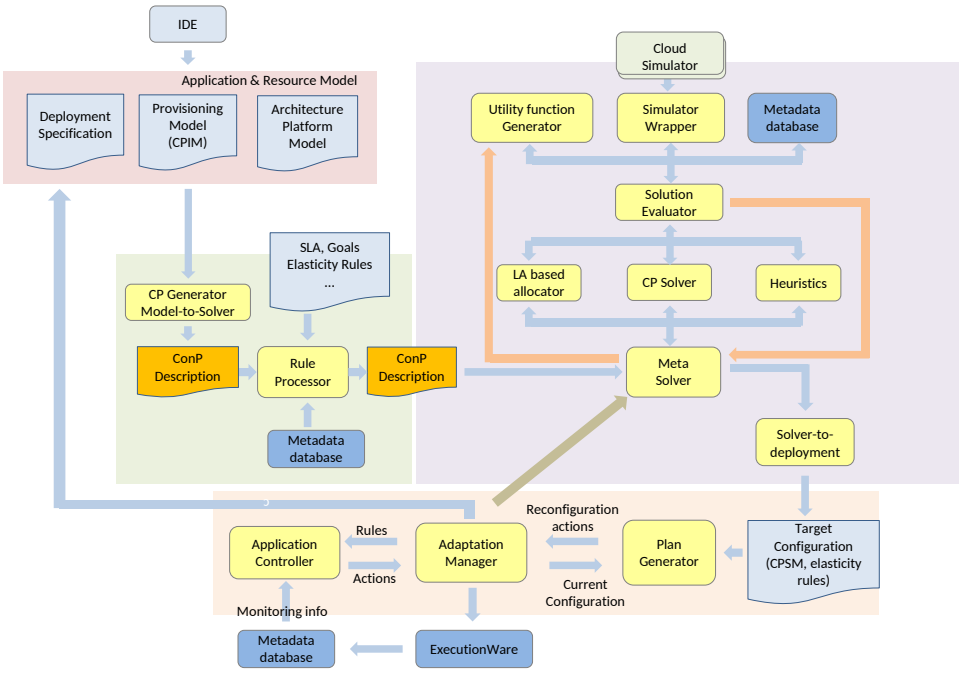

Figure 3: The full architecture of the PAASAGE upper ware with links to the other parts of the PAASAGE work flow.

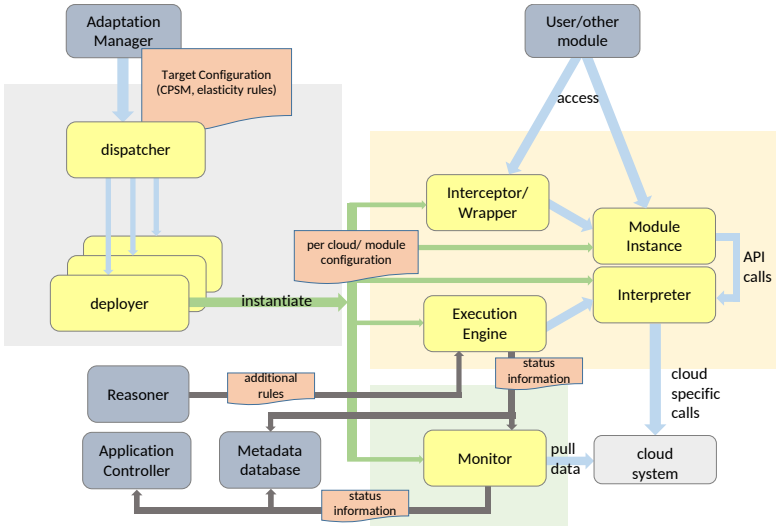

Figure 4: The full architecture of the PAASAGE execution ware with links to the other parts of the PAASAGE work flow.

2 Scope of Prototype

2.1 Introduction

The prototype consists of the components that are strictly necessary to realise the autonomic deployment flow, and even for these components some have reduced functionality. The reason for this is that the focus in the first year of PAASAGE was on defining an architecture for the autonomic deployment system, followed in the second year by the definition of the interfaces and the testing of component integration and their capability to manipulate the involved CAMEL DSLs. The focus has been to establish the PAASAGE platform, and ensure that the use case partners of PAASAGE can start developing and modelling their PAASAGE managed applications.

The components of the prototype release are shown in Figure 5. The components are integrated around the CDO server, which is considered to be fully functional in this release. Even though it can be expanded, the main task is an integration with the monitoring sub-system.

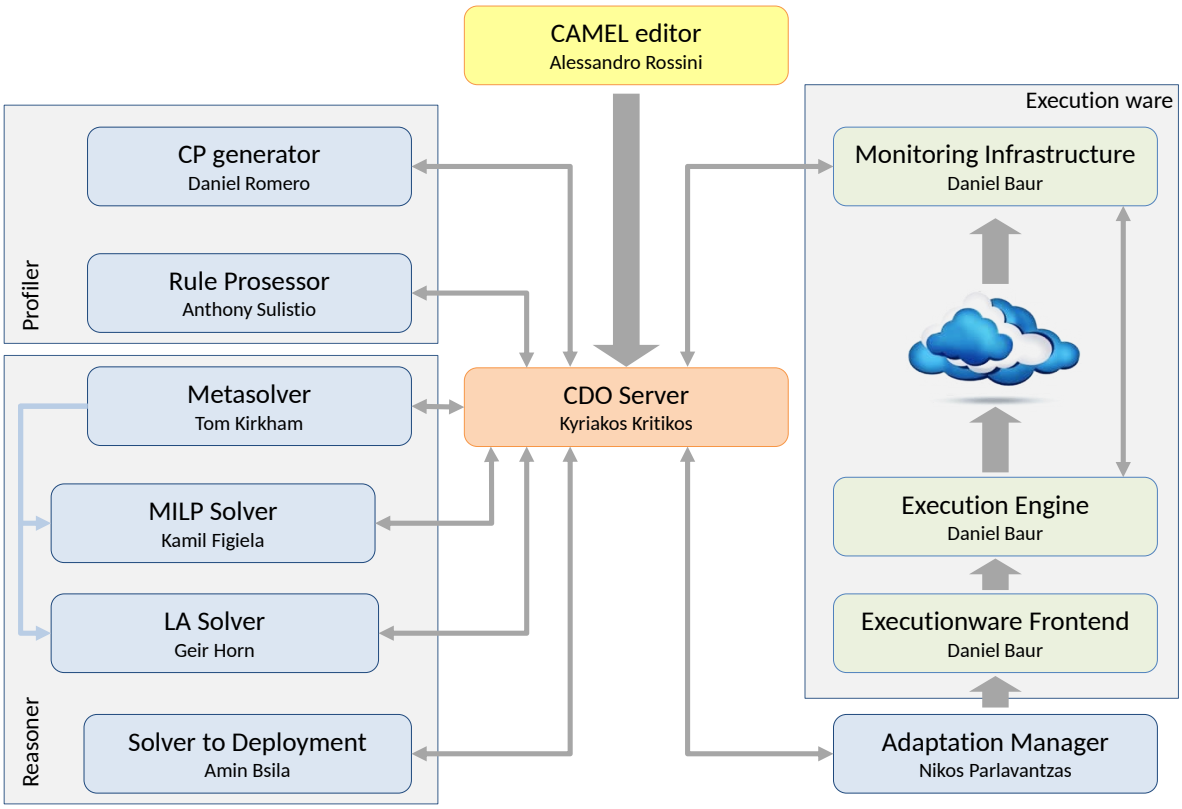

Figure 5: The components of the PAASAGE open source prototype, and their interaction through the CDO server storing the original CAMEL model and all the modifications done to this model by the upperware components. The indicated persons are the lead developers of each component responsible for the component's open source sub-project.

The profiler part is supported by the *CP Generator* and the *Rule Processor*. The first component reads the CAMEL model and converts it into a constraint programming model by defining the variables of the model, their domains, and the constraints that must be satisfied by the deployment. The Rule Processor checks all constraints of the CAMEL model and sets the domains of the variables accordingly. It also removes redundant variables and constraints from the model, e.g. if the model defines that only virtual machines of a given size should be used, only providers offering such virtual machines can be selected and all other providers must be removed from the domain for the 'provider' variable.

The reasoning engine is supported by four components in this release: The *Meta Solver*, the *MILP Solver*, the *LA Solver*, and the *Solver to Deployment*. The Meta Solver analyses the model and selects the solver most

suited for the problem. If the problem is linear in its constraints and utility function, the MILP Solver will be used, otherwise the LA Solver will be used. The two solvers are therefore interchangeable but with different characteristics. In the future it is envisioned that PAASAGE can support other solvers in addition to these two. It could also happen that the Meta Solver will be able to decouple the problem into a linear and a non-linear part, and use the solvers in parallel. The 'utility function' can be any way of evaluating a candidate deployment, for instance a Cloud simulator or even a real world test deployment, as indicated in Figure 3. The Meta Solver consequently informs the solvers how deployment candidates should be evaluated. Once a solution has been found, the Solver to Deployment component converts the solver output to Cloud Provider Specific Models (CPSM) for the providers involved in the proposed deployment.

The adapter part is currently supported only by the *Adaptation Manager* component. It takes the CPSMs, produces and validates a configuration plan, and sends this plan to the execution ware.

The execution ware is supported by three components in this release: The *Front End*, the *Engine*, and the *Monitoring Infrastructure*. The Front End receives the deployment plan from the adapter and enacts the deployment of the application on the selected providers. The control is then passed to the Engine that interacts with the Cloud providers, acquires the virtual machines, configures them and launches the user application on the set of virtual machines. Once the machines are running, the Monitoring Infrastructure takes over and collects sensor data for the running application, triggering re-configurations if necessary.

2.2 Components

CAMEL Editor

The CAMEL Editor allows specifying CAMEL models, which encompass provisioning and deployment templates, requirements and constraints, service-level objectives, scalability rules, and other information required to execute multi-cloud applications, as well as profiles of organisations and cloud providers that will host these multi-cloud applications.
The current implementation of the CAMEL Editor is based on the Eclipse[4] platform and provides a tree-based editor to specify CAMEL models, which are either serialised in XMI format or persisted into the Metadata Database through the CDO Server interface. The tree-based editor can be executed on any desktop operating system supported by Eclipse and the Java Virtual Machine, *i.e.*, Windows, Mac OS X, and GNU/Linux. Both an installation tutorial[5] for the CAMEL Editor and the technical documentation[6] for CAMEL are available.
Note that the current implementation is generated automatically from the CAMEL metamodel by the Eclipse Modelling Framework[7] (EMF). As such, the current implementation relies on existing source code from the Eclipse project and no additional source code besides the one in the CAMEL metamodel has been developed for this component.

Input parameters

No input parameter is consumed by the component. However, the user of the CAMEL Editor has to specify the configuration of the CDO Server when installing.

Output parameters

No output parameter is produced by the component. However, the resulting CAMEL model can either be serialised in XMI or persisted into the Metadata Database.

External dependencies

Library	*Description*	*License*	*Availability*
EMF	Eclipse Modelling Framework	EPL 1.0	www.eclipse.org

Known limitations

This component currently only supports the abstract syntax of CAMEL . Support for a concrete syntax of CAMEL, either in the form of a text-based or graphical syntax, will be added at a later stage in the project.

[4] http://www.eclipse.org
[5] git.cetic.be/paasage/camel/raw/master/documents/CAMELTreeBasedEditorInstallation.pdf
[6] git.cetic.be/paasage/camel/raw/master/documents/CAMELTechnicalDocumentation.pdf
[7] http://www.eclipse.org/modeling/emf/

CP Generator Model-To-Solver

The `CP-Generator Model-To-Solver` component produces a *CP Model* and a *PaaSage Application Model* from a CAMEL Model. The *CP Model* represents the selection of Cloud Providers for an application as a constraint problem, *i.e.*, as a set of variables and constraints. The *PaaSage Application Model* defines relationships between concepts in the CAMEL Model and the *CP Model*. Furthermore, the `CP-Generator Model-To-Solver` preselects Cloud provider candidates according to resources required by an application and described in the CAMEL Model. The component is defined in a maven project. This means that dependencies retrieval, compilation execution and generation of an executable Jar file is done through maven plugins.

Input parameters

ModelId	A string that represents the identifier in CDO Server of a CAMEL Model related to the application being deployed.
OutputPath	A string that represents an absolute file system path where a file containing the GenModeId will be created.

Output parameters

GenModels	A list containing the PaaSage Application (0) and CP (1) Models. The list is stored in CDO Server.
GenModelsId	A string representing the identifier of GenModels. The string is stored in the file system using OutputPath as target.

External dependencies

Library	*Description*	*License*	*Availability*
Saloon PaaSage 1.0	Library for searching valid configurations in Provider Models	MPL2.0	`http://saloon.gforge.inria.fr/repositories/releases/`
log4j 1.2.17	Logging library for Java	Apache License 2.0	`http://logging.apache.org/log4j/1.2/`
JUnit 4.8.2	Framework to write repeatable tests	EPL 1.0	`http://junit.org/`
Commons io 1.4	Library of utilities to assist with developing IO functionality.	Apache License 2.0	`http://commons.apache.org/proper/commons-io/`

Known limitations

The current version process the CAMEL Model part related to Provider Models and Deployment.

CDO Server

CDO is a persistence and distribution framework for EMF-based applications which provides both client and server functionality for the storage, updating and retrieval of models. CDO exhibits various interesting features, such as multi-user or transactional access, parallel evolution, scalability, collaboration, data-integrity and fault-tolerance. To this end, CDO technology was chosen for realizing the Metadata DataBase (MDDB) module of the PaaSage platform.

Based on the above decision, a component encapsulating the functionality of a CDO server has been realized which provides a model repository, backed-up by an underlying database, which is responsible for the storage of models specified in any meta-model of EMF Ecore as well as their querying and retrieval in the form of a java domain object.

Various types of model repositories are supported by a CDO server but the most important ones are the DBStore and HibernateStore. Both types of stores can connect to a variety of databases but they support different querying languages. A DBStore supports SQL while a HibernateStore supports HQL. Both types of stores enforce a particular default mapping from the meta-models for which models need to be stored to the respective database schema. However, this behavior can be modified. In a DBStore, EAnnotations can be used on meta-model elements to explicate the way these elements will be mapped. In a HibernateStore, mappings described through JPA annotations as well as some Hibernate-based extensions can be enforced. Apart from this difference, DBStore supports additional CDO features than a HibernateStore, such as the proper support of branches.

This CDO-server component can be exploited in two possible ways:

1. via the graphical environment of Eclipse where a CDO Session can be opened and then support either CDO transactions or views. CDO transactions are more appropriate for the storage and updating of models, while CDO views are more appropriate for querying the models stored in the CDO server.

2. programmatically via the CDO Client, a component which has been developed in order to interact with the CDO Server to be used in the code of the PaaSage component developers. This component provides an interface through which CDO transactions or views can be opened and closed, models can be queried with one or more languages (depending also on the type of the store realized by the CDO server), models can be stored and models or objects can be deleted. A detailed documentation of this component is available in the PAASAGE repository[8]

To be continued...

[8] git.cetic.be/paasage/cdo_client/raw/master/documents/CDOClientDocumentation.pdf

...CDO Server

This component can be configured in many ways which include the configuration of the underlying database, the port on which to listen for incoming connections by clients, the name of the respective repository and its type. All this information can be configured via a .properties file whose structure and content is quite comprehensive. The properties that can be configured are the following:

- dbtype - The type of the database. Until now, HSQLDB and MySQL are supported as the underlying databases.
- dburl - The URL through which the server can connect to the underlying database
- username - The username for establishing the connection to the database
- password - The password for establishing the connection to the database
- repository - The name of the repository to be created
- storetype - The type of the repository to be created. CDO supports various stores but for now only the DBStore (enabling posing SQL queries in various types of databases) and HibernateStore (enabling posing HQL queries to a variety of databases) are supported.
- port - The number of port to which the server listens for incoming connections by clients.

Input parameters

No input parameter is needed for the component to function. The information necessary is obtained from a .properties file.

Output parameters

As this is a server, no output parameters are produced. Actually, the models stored are entered into databases, so the server updates these databases and the corresponding db files.

External dependencies

Library	*Description*	*License*	*Availability*
Eclipse	Various Modules of the Eclipse Framework	Eclipse Public Licence (EPL)	www.eclipse.org
Javax	Java libraries for annotation and injection	BCL	www.java.com
DOM4j	Java Library for working with XML, XSLT and XPath with full support for DOM, SAX and JAXP	BSD	http://dom4j.sourceforge.net
Hibernate	Hibernate Java Database	LGPL 2.1 or ASL 2.0	hibernate.org
MySQL	MySQL Database	GPL	www.mysql.com

Known limitations

This component can be fully used.

Rule Processor

The Rule Processor receives a list of potential Cloud providers provided by the Profiler based on requirements specified by the application designer via the IDE. The Rule Processor checks these providers against implementation specific rules in the Metadata Database (MDDB). These rules are the base rules of the system and are expressed in terms of performance and data processing constraints associated with the specific instance of the PaaSage platform.
The Rule Processor verifies that the list of possible deployments or cloud providers satisfy all the given constraints from the data in the MDDB. Deployments formed by the Rule Processor take into account the constraints and details of the implementation in the MDDB. For example, the Rule Processor could have a requirement that data is processed in a specific territory. The MDDB could contain SLA specific data from associated service providers that fulfil this rule, and thus it is added to the list of deployments. If during this phase the Rule Processor encounters a requirement that can not be fulfilled by the PaaSage instance, the Rule Processor returns to a error message containing this detail to be fed back to the application designer.

Input parameters

ModelId	A string that represents the identifier in CDO Server of the generated CP model.

Output parameters

Error message	If a requirement that can not be fulfilled.

External dependencies

Library	*Description*	*License*	*Availability*
WSAG4J Client 2.0.0	WS-Agreement for Java framework	BSD license	http://wsag4j.sourceforge.net/site/index.html
log4j 1.2.17	Logging library for Java	Apache License 2.0	http://logging.apache.org/log4j/1.2/
JUnit 4.8.2	Framework to write repeatable tests	EPL 1.0	http://junit.org/

Known limitations

Currently, it prints to stdout the information related to the model stored in the CDO server.

Meta Solver

The MetaSolver Component acts as a gateway / decision point before the process of Solving. Taking the CDO model as refined by the Rule Processor the MetaSolver evaluates the model to determine if the model presents a linear or non linear problem. This process is assited by the use of role in the PaaSage architecture by CPML. If the problem is non linear Solvers are invoked for non linear problems by the metasolver and if the problem is linear the same process is done for linear solvers.
The Solvers currently used are the LA Solver and MILP Solver. It is expected as the project progresses that more Solvers will be added and the MetaSolver will select one or more Solvers per problem. The Meta Solver could also break the problem up and send sub-problems to different solvers. However in its current state the Solver invokes either one of the two solvers with a specific set of shell commands relevant to the appropriate solver. Errors generated by the MetaSolver are written in a Java log.

Input parameters

Parameter 1	The location of the CDO Server.
Parameter 2	The name of the CDO Model to be used, these are input as Java arguments.

Output parameters

Parameter 1	Choice of Solver and message indicating the result of the invocation i.e. pass / fail
Parameter 2	It is expected that the Solvers will generate new output and write directly to the CDO server.

External dependencies

Library	*Description*	*License*	*Availability*
Library 1	Numerical library	MIT	www.numlib.org
Library 2	XML	LGPL2.0	www.openXML.org
Library 3	COIN Mathematical Processing Library	GPL3.0	https://projects.coin-or.org/Cmpl

Known limitations

The component uses CPML to assess if the problem is linear, if not we assume it is non linear rather than having a separate check for this. The MetaSolver is limited to the use of the two solvers of this prototype.

Mixed Integer Linear Program solver

The MILP solver receives the definition of a constraint problem from the CDO server. The model is then translated to the mathematical notation in CMPL. The challenge is not only to translate the model, but also to convert the mathematical expressions into equivalent and valid formats. The CMPL mathematical modelling system is then used to solve the problem using an open source solver, such as Cbc. The optimal values found by CMPL are saved back to CDO.

Input parameters

Model	A string that represents the identifier in CDO Server of the model that is subject to be optimised.

Output parameters

CP solution	Values of variables in the model stored in CDO are updated.
Text	Debug output from CMPL and solver.

External dependencies

Library	*Description*	*License*	*Availability*
CMPL	Coliop/Coin Mathematical Programming Language	GPLv3	http://www.coliop.org
jCMPL	Coliop/Coin Mathematical Programming Language	LGPLv3	http://www.coliop.org
Scala	Scala standard library	BSD-style	http://www.scala-lang.org
Typesafe Scala Logging	Logging library	Apache 2.0	http://github.com/typesafehub/scala-logging
Typesafe Config	Configuration management	Apache 2.0	http://github.com/typesafehub/config
Logback Classic	Logging backend	Dual: EPL 1.0 and LGPL 2.1	http://logback.qos.ch

Known limitations

Only linear problems are supported.

Learning Automata based solver

The Learning Automata (LA) based solver searches for a solution to the constraint deployment problem using Learning Automata to learn the best values of the discrete variables in the model and a non-linear solver to solve for the continuous variables conditioned on the discrete variables. The background and algorithm can be found in PAASAGE deliverable D3.1.1.

Each discrete variable is assigned a learning automaton. Without any *a priori* knowledge about the value of a discrete variable, any assigned value from the variable domain can be as good as every other value, and the learning automation initiates the search with a uniform probability vector over all possible values in the variable's domain. If there is knowledge in the metadata database indicating that some values in the domain may work better than others, then this initialisation will be non-uniform with the value probabilities reflecting the relative goodness of the values in the variable's domain. This will start the search making it more likely to choose previously good assignments for a variable.

The LA will then interact iteratively in a game where each play involves every automaton assigning a value for its discrete variable according to its probability distribution over the variable's domain, and then the value of the utility function is assessed after solving for the continuous variables. The objective is to *learn* the assignments that maximises the utility in the long run. The problem is stochastic because the utility function can be based on measured values that may change over time, e.g. the cost of a virtual machine with a Cloud provider. Thus, evaluating the utility twice for the same assignment of variables, may give two different utility values.

The benefit of this approach is that the solver can keep on running, always returning the best known solution as soon as a solution better than the current is found. The solver will receive updated metric values from the execution ware, and immediately take them into consideration in the next play of the assignment game. The found solution will therefore be adaptive to the current execution context of the application, and application deployment can take place with an initial solution and then be adapted by the adapter component as better deployment configurations become available.

The CAMEL model of the application is converted into a set of variables, their domains and initial values are retrieved from the CDO server, if such values are available. The constraints are also taken from the CDO model. The variables, the constraints, and the utility function is then compiled and linked with the solver. This implies that there is one solver for each model, and if the model changes, the currently running solver must be stopped and recreated for the possibly new variables, constraints, and utility function. This process is described in deliverable D3.1.1.

Input parameters

IP address	A textual IP address for the *Metrics Collector* that receives the measurements from the running application and updates the metric values in the metadata database. It also publishes the measured values to any solver subscribing to these updates.
ModelId	A string that represents the identifier in CDO Server of a Camel Model related to the application being deployed.

Output parameters

None	The solver will write back to the CDO server the assigned variable values and the corresponding utility as soon as a *feasible* solution satisfying all constraints is found.

External dependencies

Library	*Description*	*License*	*Availability*
NLopt	Open-source library for non-linear optimisation, providing a common interface for a number of different free optimization routines available online as well as original implementations of various other algorithms. Provided as a standard package for most Linux distributions	LGPL	http://ab-initio.mit.edu/wiki/index.php/NLopt

To be continued...

...Learning Automata based solver

Theron	A lightweight C++ concurrency library based on the Actor Model	MIT	http://www.theron-library.com/
ZeroMQ	A message queue library providing sockets that carry atomic messages across various transports like in-process, inter-process, TCP, and multicast. Version 3 or higher is required and this is provided as a standard package with most Linux distributions	LGPv3	http://zeromq.org/
zmqpp	C++ binding for ZeroMQ as a 'high-level' library that hides most of the C-style interface the core ZeroMQ provides. Must be cloned from the Git repository	MIT	https://github.com/zeromq/zmqpp

Known limitations

The source code is released as part of the prototype and the interface has been tested to ensure that the solver is correctly built for a given model. However, the metric collector part is not implemented as a part of this prototype, and the solver has therefore not been tested as part of this release.
The utility function must currently be created manually, and it is a topic for the research in the next years of PAASAGE how a good utility function can be derived automatically from the information in the CAMEL model.

Solver to deployment			
`Solver-to-deployer` is a glue layer between the *Reasoner* and the *Adapter*. It participates to lowering the dependencies of solvers to the remaining of PAASAGE. Upper ware metamodels aim at enabling interactions between the *Profiler* and the *Reasoner* while lowering dependencies to CAMEL. Solvers produce solutions using these upper ware metamodels. The main objective of the `Solver-to-deployment` component is to translate the output of the Solvers into the Deployment Model CPSM.			
Input parameters			
SolutionModelId	A string that represents the identifier in CDO Server of a Camel Model related to the solution being deployed.		
OutputPath	A string that represents an absolute file system path where a file containing the deployable model will be created.		
Output parameters			
GenModels	The CloudML model of the deployable solution.		
External dependencies			
Library	*Description*	*License*	*Availability*
log4j 1.2.17	Logging library for Java	Apache License 2.0	`http://logging.apache.org/log4j/1.2/`
Commons io 1.4	Library of utilities to assist with developing IO functionality.	Apache License 2.0	`http://commons.apache.org/proper/commons-io/`
Known limitations			
The current version generates a CloudML deployable model.			

Adaptation manager

The purpose of the `Adaptation Manager` is to transform the currently running application configuration into a target configuration in an efficient and safe way. The component operates as follows: (1) it loads the target deployment model from the CDO server, (2) it produces and validates a reconfiguration plan, and (3) it executes the reconfiguration plan through interacting with the execution ware. Interactions with the execution ware are based on a REST API.

Input parameters

ModelId	A string that represents the identifier in the CDO Server of the Camel Model representing the target deployment.
URL	The URL of the Executionware Frontend

Output parameters

REST-Interface	Uses the REST Interface to interact with Executionware

External dependencies

Library	*Description*	*License*	*Availability*
log4j 1.2.17	Logging library for Java	Apache License 2.0	`http://logging.apache.org/log4j/1.2/`
minimal-json 0.9.1	JSON parser and writer for Java	Apache License 2.0	`https://github.com/ralfstx/minimal-json`
Http Components 4.3.3	Tools for HTTP and associated protocols	Apache License 2.0	`http://hc.apache.org/`
JUnit 4.11	Framework to write repeatable tests	EPL 1.0	`http://junit.org/`
Commons CLI 1.2	Parsing command line options	Apache License 2.0	`commons.apache.org/cli/`

Known limitations

The component currently performs initial application deployment, rather than dynamic reconfiguration. Moreover, there is currently no plan validation.

Executionware Frontend

The Executionware Frontend is the entry point for the upper ware. It provides a REST-based interface which can be used by the upper ware to instruct the underlying Execution Engine. The provided graphical Web-Interface may also be used manually to configure the Execution Engine and provide additional information required for the deploying process. It configures the Execution Engine via low-level configuration files.

Input parameters

Web Interface	The Executionware Frontend offers a web interface for configuring the basic parameters for the Execution Engine. The web interface is also used for entering user and cloud specific information like user accounts.
REST-Interface	The REST-Interface offers the capabilities of the Execution Engine to the Adaption Manager of the Upperware.

Output parameters

File System & Command Line	The Executionware Frontend communicates with the Execution Engine by using the file system (configuration files) and the command line.
REST-Interface	The REST-Interface provides the Execution Engine with the information stored within the Executionware Frontend.

External dependencies

Library	*Description*	*License*	*Availability*
Hibernate Entitymanager	Database	LGPL 2.1	http://www.hibernate.org
Apache Commons	Helper	Apache License 2.0	http://commons.apache.org/
Apache Commons	Helper	Apache License 2.0	http://commons.apache.org/
MariaDB Java Client	Database	LGPL 2.1	https://mariadb.com
Zip4j	Helper	Apache License 2.0	http://www.lingala.net/zip4j/
Java Hamcrest	Testing	BSD	www.hamcrest.org
Halbuilder	JSON/REST	Apache License 2.0	http://gotohal.net/

Known limitations

As the Executionware Frontend is a frontend to the Execution Engine, its limitations are given by the Execution Engine. All features currently offered by the Execution Engine are supported by the frontend.

Execution Engine

The Execution Engine is responsible for the orchestration of the different clouds, meaning that it aquires the needed virtual machines of the defined cloud providers and installs the application on them. The Execution Engine uses a modified version of the open-source software Cloudify for this purpose. It is configured by the Executionware Frontend via low-level configuration files and instructed via the command line.

Input parameters

File System	The Execution Engine receives its configuration from the Executionware Frontend via the file system. The communication is based on *.groovy files in the DSL of cloudify.
Command Line	The Execution Engine receives commands via command line calls executed by the Executionware Frontend.

Output parameters

Cloud API	The Execution Engine calls the different cloud provider APIs via an abstraction layer supporting several cloud middlewares.
Installer	The applications are installed via SSH-connections to the created virtual machines.

External dependencies

Library	*Description*	*License*	*Availability*
Cloudify 2.7	Cloud Orchestration	Apache License 2.0	http://getcloudify.org/

Known limitations

The current prototype of the Execution Engine has the following limitations:

1. The Execution Engine is currently able to handle the deployment of one application, which however may consist of several (possibly interdependent) components.
2. The Execution Engine is able to deploy the application to a single cloud. This means that cross-cloud deployment is currently not supported. Multi-cloud functionality is available though.
3. The Execution Engine's deployment is tested with OpenStack and Flexiant Cloud Orchestrator. While the support of other cloud providers is provided by its abstraction layer, the support remains untested to the current time.
4. The Execution Engine is able to execute the initial deployment. Also internal scaling (scale-in, scale-out) of single application components is supported. Yet, the platform lacks sophisticated support for redeployment.

Monitoring Infrastructure

The monitoring infrastructure is responsible for monitoring the metrics defined in the application model and to communicate them to the other components of the PaaSage architecture via the MDDB. The monitoring infrastructure consists of several components:

TSDB The time series database (TSDB) is distributed database suited for processing the large amount of sensor data collected by the agents. It stores the collected data for use of the other components.

Agents The monitoring agents are responsible for collecting the sensor data. They implement probes for monitoring basic data about the execution environement on the virtual machine. In addition they offer an interface for the reporting of application specific data. The thereby collected sensor data is sent to the TSDB.

Collector The collector is responsible for interlinking the time-series database with the meta-data database (MDDB).

Input parameters

Sensors	The sensors implemented in the monitoring agents aquire basic monitoring data of the execution environement on the virtual machines.
Sensor Interface	The sensor interface offers an interface where applications can report application-specific monitoring information.

Output parameters

Monitoring Data	The collector provides the MDDB with the monitoring data stored in the TSDB.

External dependencies

Library	*Description*	*License*	*Availability*
kairosdb	Time-Series Database	Apache License 2.0	https://github.com/kairosdb
kairosdb-client	Client for kairosdb	LGPL2.0	https://github.com/kairosdb
Cloudify 2.7	Cloud Orchestration	Apache License 2.0	http://getcloudify.org/

Known limitations

While the prototype is fully functional regarding the monitoring workflow, it only implements sensors for basic monitoring data.

3 Installation

The installation of the PAASAGE components has been verified on computers supporting the following minimal configuration:

- A modern Linux distribution. Note however that the PAASAGE prototype has only been tested under Ubuntu 14.04; or
- MacOSX 10.9 Maverick;
- In both cases at least 4GB RAM is minimum, 8GB RAM is recommended.

From a functional point of view, nothing prevents the distribution of various components on separate virtual machines (VMs). All components were developed with a share-nothing[9] concept in mind so that each one can run separately. However, as PAASAGE is in progress, the chef recipes have not yet been adapted to allow such a deployment and, at the time of writing, chef deploys all components on the same Ubuntu 14.04 Virtual Machine:

1. Install Git on the machine

```
sudo apt-get install git
```

2. Download and install ChefDK[10]
3. Download and install VirtualBox and VirtualBox Extension Pack[11] according to the operating system. Note that VirtualBox is provided as a package for Oracle Linux, Debian based Linux, and RPM based distributions like Fedora and Red Hat Enterprise Linux (RHEL). For these distributions only a repository file needs to be installed, and then VirtualBox can be installed and kept updated with the standard package manager on the system.
4. Download and install Vagrant[12] according to the operating system.
5. Install Vagrant plugins

```
vagrant plugin install vagrant-vbox-snapshot
```

6. Configure Virtualbox network:
   a) Open VirtualBox → File → Settings → Networks → Host Only
   b) Edit the vboxnet0 so that: IPv4 address = 10.19.65.1 ; Netmask = 255.255.255.0; DHCP server is enabled (see Figure X)
7. Clone the PAASAGE Git repository[13]. Currently this repository is located on a Git server provided by the PAASAGE partner CETIC. The following shows how it can be installed to a folder *WorkingCopies* on the host work station.

```
cd $HOME/WorkingCopies
git clone ssh://git@git.cetic.be:61011/paasage/wp6_integration.git
git checkout kitchen
export WP6_INTEG=$HOME/WorkingCopies/wp6_integration
cd $WP6_INTEG/admin
bundle install
```

8. Verify the list of PAASAGE Virtual Machines (VMs) that can be deployed by the following command

[9] http://en.wikipedia.org/wiki/Shared_nothing_architecture
[10] http://www.getchef.com/downloads/chef-dk/ubuntu/
[11] https://www.virtualbox.org/wiki/Downloads
[12] http://www.vagrantup.com
[13] The PAASAGE consortium is currently in the process of moving the code to a publicly available repository. Please contact the PAASAGE's Open Source Manager Geir Horn at Geir.Horn@mn.uio.no for status and access details.

```
kitchen list
```

9. The workstation is now ready to launch the PaaSage platform VMs. Trigger the deployment by the following commands, and note that this will download and install a lot of things, so please be patient for the first time

```
cd $WP6_INTEG/admin
kitchen converge full_paasage_platform
kitchen login full_paasage_platform
```

10. After finishing working with PAASAGE, the platform can be exited and destroyed by the following command

```
kitchen destroy full_paasage_platform
```

4 Use of the prototype

Once the PAASAGE platform is up, using to the installation instructions of the previous section, PAASAGE can be executed in order to process an application deployment. In this respect, a "master script" has been developed, which aims to instantiate the various components of the PAASAGE work flow.

In future versions, the starting point to launch a PAASAGE deployment run will be integrated in a graphical front-end. However, at the time of writing, the simplest way to achieve this is to log into the PAASAGE platform and execute the so called master script using the following commands.

```
kitchen login full_paasage_platform
cd /etc/paasage/
./masterscript.sh APPLICATION_MODEL_URL
```

The script takes one parameter: the Uniform Resource Locator (URL) to the application model given as an Extensible Markup Language (XML) Metadata Interchange (XMI) file in CAMEL format. Once executed, the script downloads the XMI file, and loads it into the metadata database. Then, the various PaaSage components are sequentially called in order to process the model and deploy the application.

5 Licence and conditions

5.1 The PAASAGE platform license

PaaSage is a committed open source project and all code necessary for running the platform must be available as open source, even if it constitutes development prior to PaaSage. Many of the PAASAGE components are enhancements and innovations involving existing open source modules and projects. The PAASAGE members wish to favour the creation of source code commons, but avoid the problems of "virality" that cause incompatibility problems between software components. The following principles have therefore been set for the adoption of an open source license in PAASAGE:

1. The chosen license should be compatible with the licenses currently in use by the PAASAGE partners for their open source projects being enhanced through PAASAGE. The currently used licenses are: Apache[14], LGPL3.0[15] and Eclipse[16].

2. The license should protect the investment we have done in PAASAGE and should therefore be "weak copyleft"[17], i.e. if someone improves the PAASAGE platform code, these improvements should be released back as open source for others to use.

[14] http://opensource.org/licenses/Apache-2.0
[15] http://opensource.org/licenses/LGPL-3.0
[16] http://opensource.org/licenses/EPL-1.0
[17] http://en.wikipedia.org/wiki/Copyleft

3. The chosen license should not restrict commercial use of PAASAGE, and should permit PAASAGE software to be integrated with commercial closed source software whether it will be simple use of the PAASAGE platform or linking other libraries with PAASAGE.

The open source cloud computing projects landscape is characterised by the weight of initiatives hosted by Apache Foundation. The Apache license is also used by other organisations such as Appscale[18] or Red Hat[19]. As a consequence, the Apache license is often used in "Infrastructure as a Service" (IaaS) and "Platform as a Service" (PaaS) projects. Although it is widely used in open source cloud project, the permissive Apache license does not satisfy the constraints expressed for the PAASAGE project.

An additional requirement for the selection of a common PAASAGE license was that it should be well know in the developer communities to facilitate easy adoption of the PAASAGE code base. The choice was therefore confined to the the list of recommended licenses that is published by Open Source Initiative[20] leading to a choice among four licences:

GNU Lesser General Public License (LGPL) Version 2.1 suffers a lack of clarity due to the distinction between dynamic or static linkage. The last version, version 3.0, clarify the point by removing that distinction. The LGPL is compatible with the widely used GPL.

Mozilla Public License (MPL[21] was created by Netscape in order to protect the Mozilla projects and to simplify the use of third-parties modules that are under various free and proprietary licences. The MPL 1.1 is incompatible with the widely used GPL. The new MPL 2.0 was written with the compatibility problem in mind.[22] The MPL 2.0 license can be compatible with MPL 1.1 and GNU licenses, which is acknowledged[23] by the Free Software Foundation (FSF).

Common Public License (CPL) or Eclipse Public License (EPL) The EPL is an evolution of the CPL[24]. It is a file-based weak copyleft license that is associated to the Eclipse projects. The EPL is incompatible with the widely used GPL.

Common Development and Distribution License (CDDL) [25] was created by Sun Microsystems. It is inspired by MPL. The added value of the license compared to the widely used MPL is unclear.

The LGPL 3.0 license is stronger in terms of reciprocity and responsibility to contribute to the development. However some companies could be afraid by the "GNU" label. The MPL 2.0 license is easier for the creation of combined works that contain files with various licenses. In consequence, MPL 2.0 has been adopted unanimously as the common licence for PAASAGE.

Note that this does not prevent partners of PAASAGE or users of PAASAGE to replace PAASAGE components with commercial components for better performance as per the third principle above. It also does not prevent partners to the PAASAGE project to sell packaged versions of PAASAGE, or provide services on the PAASAGE platform.

The chosen open source license must be referenced in the source code. The developers have to indicate the license in the source code of the software. The original text is written in a file that is named LICENSE or LICENSE.txt[26] in the root directory of the source code. Each source file should contain the following text as part of the file header:

[18] www.appscale.com
[19] www.redhat.com
[20] http://opensource.org/licenses
[21] http://opensource.org/licenses/MPL-2.0
[22] https://www.mozilla.org/MPL/2.0/FAQ.html
[23] https://www.gnu.org/licenses/license-list.html#MPL-2.0
[24] http://www.ibm.com/developerworks/library/os-cpl.html
[25] http://opensource.org/licenses/CDDL-1.0
[26] This file can be copied from https://www.mozilla.org/MPL/2.0/index.txt

```
Copyright (C) 2014 NAME <EMAIL COMPANY.EXT>
OPTIONAL CONTACT DETAILS

This Source Code Form is subject to the terms of the
Mozilla Public License, v. 2.0. If a copy of the MPL
was not distributed with this file, You can obtain one at
http://mozilla.org/MPL/2.0/.
```

5.2 Governance

Governance in of software projects can be defined as *"the complex process that is responsible for the control of project scope, progress, and continuous commitment of developers"* [5]. In particular, the scope and purpose of the "governance model" for an open source project can be defined as follows[27]:

> *"A governance model describes the roles that project participants can take on and the process for decision making within the project. In addition, it describes the ground rules for participation in the project and the processes for communicating and sharing within the project team and community. In other words it is the governance model that prevents an open source project from descending into chaos."*
>
> — Ross Gardler and Gabriel Hanganu

Owing to its importance, the governance of open source projects has attracted the interest of the management science community and software communities alike. In the interest of keeping this document brief, a full survey of the literature is omitted, and the interested reader is referred to the extensive literature reviews in Capra *et al.* [6] and O'Mahony [7].

The traditional view is that highly structured development projects, based on disciplined adherence to defined methodologies and development plans is the best way to increase development efficiency. However, over the last decade this view has been challenged by agile methodologies based on informal governance and autonomous coordination among the developers. Open source projects projects receive contributions from a community of developers over which traditional project management methodologies cannot be applied. Governance of open source projects and agile software development therefore has much in common. However, as a research project PAASAGE has clear plans and responsibilities, and clear deliverables. The governance model must reconcile these two perspectives on the PAASAGE's code base.

The good news is that the extensive study of 75 major open source projects undertaken Capra *et al.* leads them to conclude that a larger degree of openness in the governance leads to better code produced, however at the expense that the development effort is higher [6]. This can intuitively be explained by the fact that open governance requires better defined and complete modules interacting through well defined interfaces, but it takes longer to develop such complete components.

O'Mahony has identified five core principles that participants to open source projects value as essential to good community management [7]. Owing to the fact that the PAASAGE open source project will be the only place where the project code will be developed after month 24 of the project, these principles will be adhered to on a sliding scale ranging from full project control until now up to full community control two years after the end of the project, i.e. in four years from now.

The governance model will be revised annually. The PAASAGE project has appointed Geir Horn from the University of Oslo to be the Open Source Manager until the end of the project, and he is responsible for the annual revisions of the governance model. The initial model described here will be in effect during the first year as an open source project, i.e. until October 2015, and confirms as follows with O'Mahony's five principles:

Independence is currently not possible as the code is strictly linked to the European research project PAASAGE, and it is backed by the organisations contractually co-funded as part of this project. However, a research project is not a legal entity and no agreements exists or shall exist among the PAASAGE partners

[27] http://oss-watch.ac.uk/resources/governancemodels

that will allow them to keep control over the open source PAASAGE project beyond the project period. Thus the PAASAGE open source project will gradually gain independence from the PAASAGE research project as more developers from the community participates to the maintenance and evolution of the code base.

Pluralism is already in place as each of the prototype components and each of the components of the full PAASAGE architecture have different owners as indicated in Figure X. Some of these components exists as open source projects with separate communities outside of PAASAGE. There is consequently no single organisation or person that can claim the ownership of the code base.

Representation will be detailed in the future versions of the governance models. Currently, each component in the PAASAGE Open Source Project has a named developer assigned and this developer represents other external developers in the PAASAGE Hackers' meetings deciding on future development directions.

Decentralised decision making is ensured by the developers assigned as responsible for their component leading the development of the component and deciding on the implementation details of the component jointly with other external community developers contributing to the component. The fundamental regulation is that the developer who contributes the most, will also have the most to say over the components future evolution.

Autonomous participation is ensured since the PAASAGE developers now working on the code are named individuals, albeit paid by their organisations under the PAASAGE research project contract. External developers are granted the rights to download, test, and use the PAASAGE code base. The PAASAGE team of developers commit to react timely to any bugs detected by external code users. Furthermore, individual developers are invited to contribute to improving the PAASAGE code base; however, until October 2015, contributions will be monitored by the component owner who autonomously decides whether the provided contribution can be included into the component. It is envisaged that after October 2015 external developers will be accepted as part of the PAASAGE development team by the Hackers' meeting after nomination by the component owner, after which there will be no distinction between the type of developer involved and working to maintain and enhance the PAASAGE open source code base.

6 Conclusion

This brief guide documents the components developed for the PAASAGE prototype, and gives instructions on how to install and run the prototype software, which is the actual deliverable.

References

[1] A. Rossini, N. Nikolov, D. Romero, J. Domaschka, K. Kritikos, T. Kirkham and A. Solberg, "D2.1.2 – CloudML Implementation Documentation (First version)", PaaSage project deliverable, Apr. 2014.

[2] K. Kritikos, M. Korozi, B. Kryza *et al.*, "D4.1.1 – Prototype Metadata Database and Social Network", PaaSage project deliverable, Mar. 2014.

[3] A. Bsila, N. Ferry, K. Figiela *et al.*, "D3.1.1 – model based cloud platform upperware", PaaSage project deliverable, Mar. 2014.

[4] J. Domaschka, A. Sulisto, P. Garefalakis, D. Metalidis, C. Zeginis, C. Sheridan, K. Lu, E. Yaqub, B. Balis and D. Król, "D5.1.1/D5.1.3 – Prototype Executionware/Prototype New Execution Engines)", PaaSage project deliverable, Mar. 2014.

[5] Patrick S. Renz, *Project Governance - Implementing Corporate Governance and Business Ethics in Non-profit Organizations*, ser. Contributions to Economics. Physica Verlag Heidelberg, 260 pp., ISBN: 978-3-7908-1927-4.

[6] Eugenio Capra, Chiara Francalanci and Francesco Merlo, "An empirical study on the relationship between software design quality, development effort and governance in open source projects", *IEEE Transactions on Software Engineering*, vol. 34, no. 6, pp. 765–782, Nov. 2008, ISSN: 0098-5589. DOI: 10.1109/TSE.2008.68.

[7] Siobhán O'Mahony, "The governance of open source initiatives: what does it mean to be community managed?", *Journal of Management & Governance*, vol. 11, no. 2, pp. 139–150, 1st May 2007, ISSN: 1385-3457, 1572-963X. DOI: 10.1007/s10997-007-9024-7.