PaaSage

Model Based Cloud Platform Upperware

Deliverable D8.2.2

Open Source Prototype System 2

Version: 0.9

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:	20 April 2016
Document	
Period covered:	Period IV
Deliverable number:	D8.2.2
Deliverable title	Open Source Prototype System 2
Contractual Date of Delivery:	30 September 2016
Actual Date of Delivery:	14 November 2016
Editor (s):	Geir Horn
Author (s):	Daniel Baur, Shirly Crompton, Kamil Figiela, Dennis Hoppe, Tom Kirkham, Kyriakos Kritikos, Nikos Parlavantzas, Christian Perez, Daniel Romero, Alessandro Rossini, Arnab Sinha, Robert Viseur
Reviewer (s):	Dennis Hoppe
Participant(s):	
Work package no.:	WP8
Work package title:	Exploitation
Work package leader:	Frédéric Fleurial Monfils
Distribution:	Public
Version/Revision:	0.9
Draft/Final:	Pre-Final
Total number of pages (including cover):	48

DISCLAIMER

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

Contents

Executive Summary

This document is the final guide to the PaaSage software platform and the various components developed in PaaSage. It presents the overall objectives of the software and the global architecture, before presenting the work flow implemented for deploying cross Cloud applications.

Intended Audience

the PaaSage software for cross cloud application deployment is a tool for DevOps[1], which can be defined as a development methodology with a set of practices aimed at bridging the gap between Development and Operations, emphasising communication and collaboration, continuous integration, quality assurance and delivery with automated deployment [1], as illustrated in Figure 1 below[2].

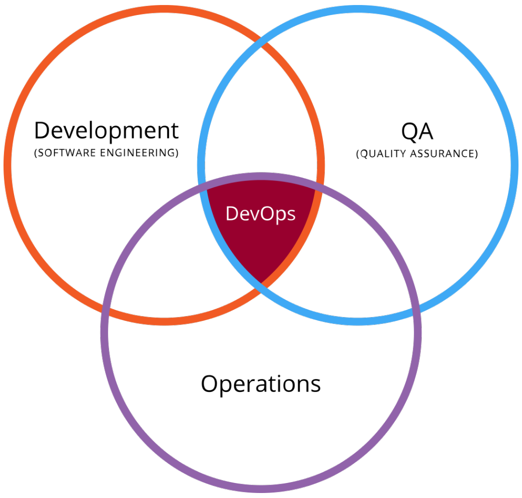

Figure 1: A DevOps is as a person working at the intersection of applicaton developer communities, operational deployment, and quality assurance and operational monitoring.

This document aims at providing the necessary *overview* to understand PaaSage and its objectives, and the context of the developed software, 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 the implemented components. The document also priovides insight into the Open Source Software governance model, and how a developer my contribute to the PaaSage platform in the futrue.

[1]https://kartar.net/2010/02/what-devops-means-to-me.../
[2]https://en.wikipedia.org/wiki/DevOps

1 Architecture

PaaSage is tool for DevOps to master the complex and often error prone task of deploying an application to the Cloud. The DevOps could learn the interface and features of *one* Cloud provider, but it will be very costly to master the development to *different* providers, and it could easily be a nightmare to deploy a complex distributed application *across* several Cloud 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 2 shows a very high level view of PaaSage and the scope covered by the project.

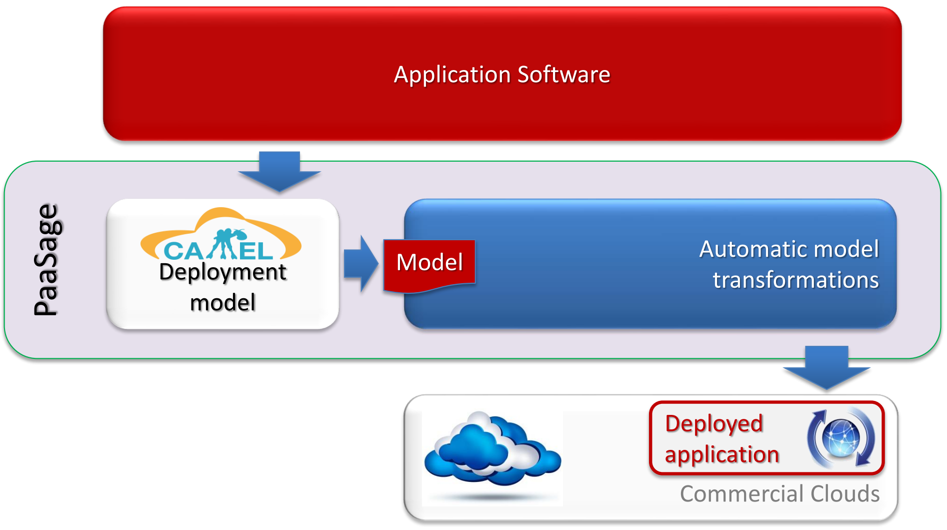

Figure 2: The scope of PaaSage is to extend the application model with platform annotations and user's goals and preference to a Cloud Application Modelling and Execution Language (CAMEL) model, which is then transformed by PaaSage to a deployed application in one or more Clouds. Even though the PaaSage platform is open source, proprietary elements indicated in red will stay proprietary throughout *the use* of PaaSage.

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[3] (EMF) on top of the Connected Data Objects[4] (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 D2.1.3 CAMEL documentation* [2] for further details on the modelling concepts, and *Deliverable D4.1.2 Product database and social network system* [3] 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

[3] http://www.eclipse.org/modeling/emf/
[4] http://www.eclipse.org/cdo/

execution ware responsible for instantiating the different parts of the application on the selected Cloud providers 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.

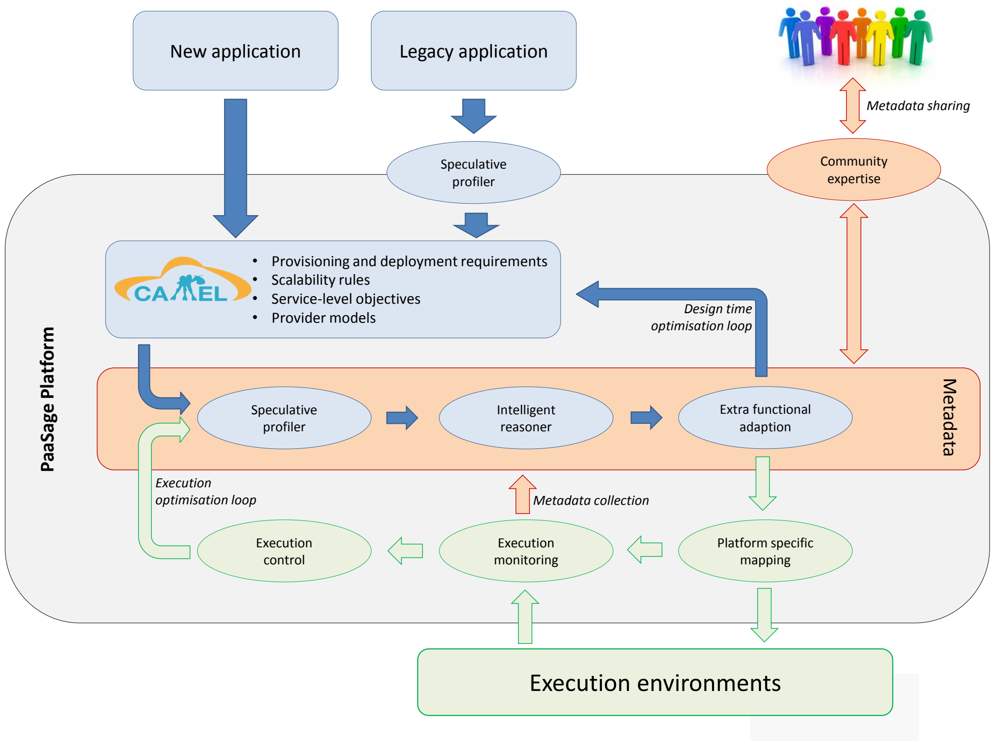

Figure 3: The overall PAASAGE architecture and workflow. The blue parts in the upper flow are the upper ware, orange parts are the persistent metadata structure, green parts are the execution ware.

This *feedback loop* controlling the application deployment is depicted in Figure 3 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[5] 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 4, and the different components are described in PAASAGE *Deliverable D3.1.2 Product Upperware* [4]. Similarly, the architecture of the execution ware is shown in Figure 5, and the detailed description of these components can be found in PAASAGE *Deliverable D5.1.2 Product Executionware* [5].

[5]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.

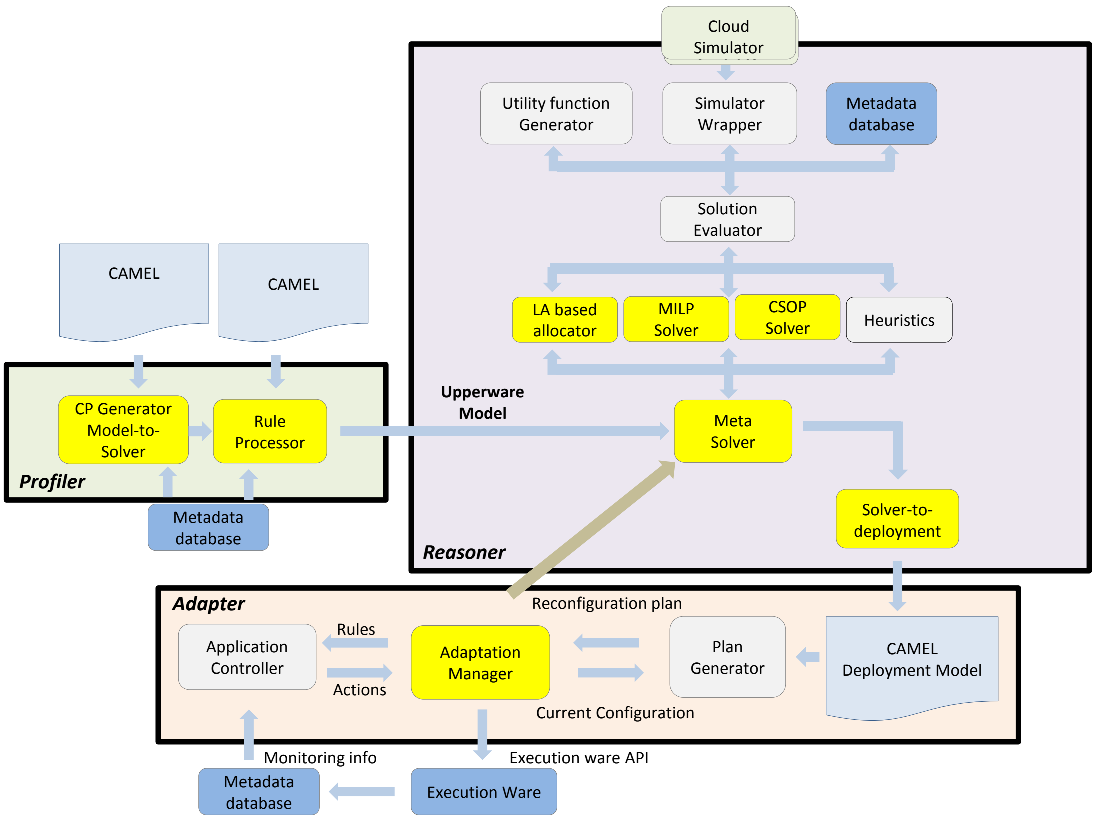

Figure 4: The full architecture of the PAASAGE upper ware with links to the other parts of the PAASAGE work flow.

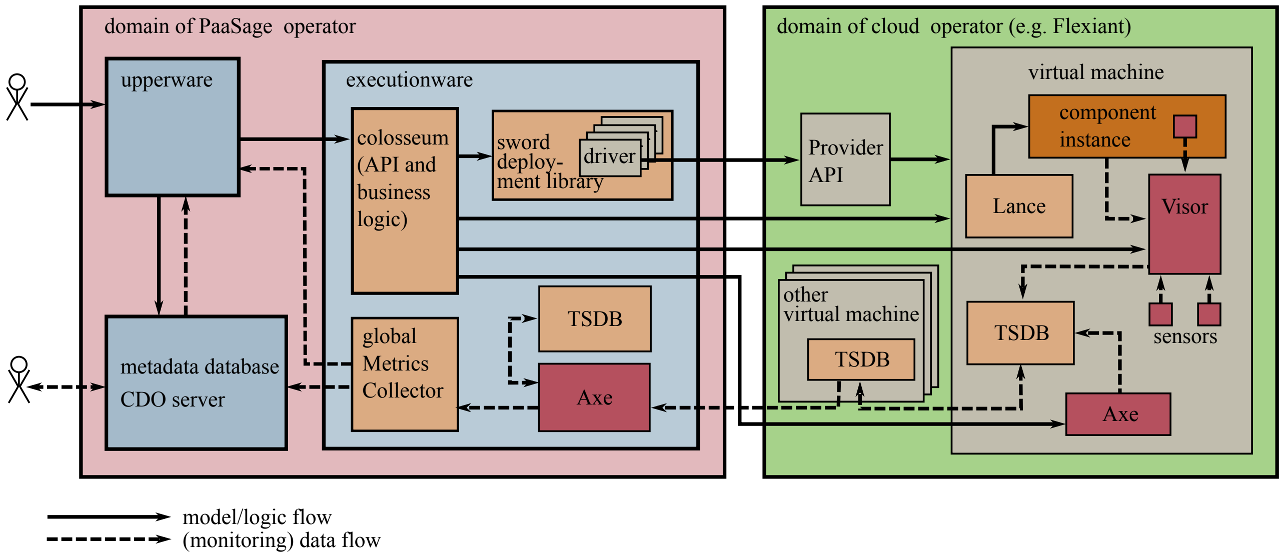

Figure 5: The full architecture of the PAASAGE execution ware with links to the other parts of the PAASAGE work flow. TSDB = Time Series Data Base.

2 Scope of Implementation

2.1 Introduction

The PaaSage software consists of the components that are necessary to realise the autonomic deployment flow described by the PaaSage architecture. The architecture is highly distributed, and the various components are contributed by several PaaSage project partners. Furthermore, PaaSage has been highly influential on work in other projects outside of PaaSage:

- The CAMEL[6] development was initially joint work with the MODAClouds[7]. CAMEL is now used in CACTOS[8] aiming at improving data centre operation; CloudSocket[9] aiming at semantic deployment of buisness processes to Cloud workflows; and ARCADIA[10], aiming to facilitate self-combining software systems built from distributed micro-services combined in asyclic service graphs.

  The feature set of CAMEL is a superset of the deployment specification offered by the Topology and Orchestration Specification for Cloud Applications (TOSCA[11]) standardised by the Organization for the Advancement of Structured Information Standards (OASIS[12]), and this has led to CAMEL now being proposed as a run-time extension to TOSCA.

- The execution ware of PaaSage is joint development with the CACTOS and CloudSocket.

The result of this enlargement of the PaaSage development is that both CAMEL and the execution ware mainly exist in repositories separate from PaaSage, and that this deliverable is limited to describe the PaaSage interface components and the PaaSage contributions to these joint efforts with other projects.

The main components of the PaaSage released platform are shown in Figure 6. The components are integrated around the CDO server that stores the initial CAMEL models received from the application DevOps, and then stores each better solution the reasoner can find for the current application context. The deployed model is updated by the Adapter based on any changes done by the execution ware. In this way the CDO server will persist the whole execution history of the application and the reconfigurations and adaptations made during the application's lifetime.

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 five components in this release: The *Meta Solver*, the *MILP Solver*, the *CP 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 CP Solver or the LA Solver will be used. The three solvers are therefore interchangeable but with different characteristics. In the future it is envisioned that PaaSage can support other solvers in addition to these three. 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 4. 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.

Changing metrics will in general lead to a change in the solution found by a solver if the changed metric is used in the solver's objective function or in the set of constraints. Only the LA Solver is capable of optimise

[6] http://camel-dsl.org/
[7] http://www.modaclouds.eu/
[8] http://www.cactosfp7.eu/
[9] https://www.cloudsocket.eu/
[10] http://www.arcadia-framework.eu/
[11] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca
[12] https://www.oasis-open.org/

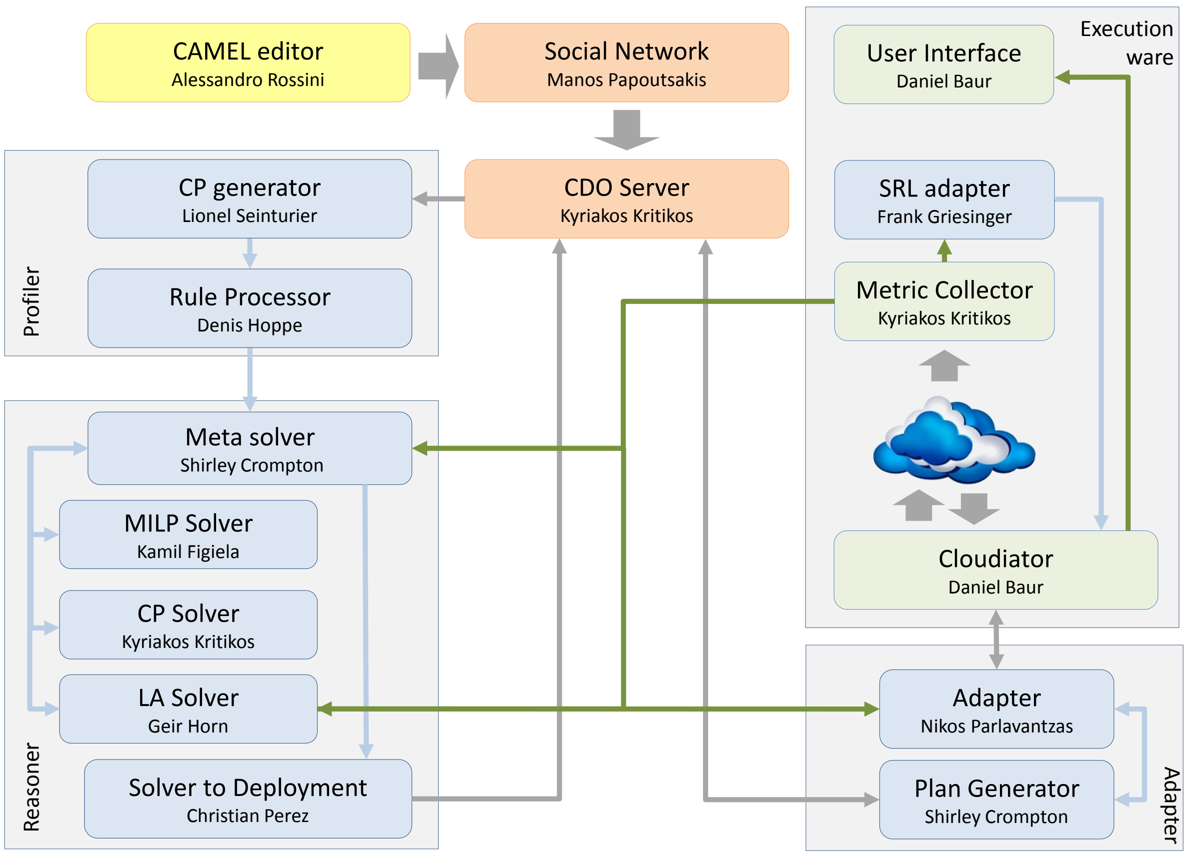

Figure 6: The components of the PAASAGE platform. The indicated persons are the project leaders of the open source sub-projects for the various components and responsible for post PAASAGE debugging and continued development.

over changing objective functions and constraints. Hence it will receive the updated metric value directly from the execution ware. For the other two solvers, the Meta Solver will recive the updated metric and use this as a fixed value parameter in the objective function passed to the solver for the concerned solver to find a solution for the current execution context. If the solver is not able to find a better solution for the current context before the next metric update, the solver will be terminated and restarted on the new problem defined by the received metric values as only these will reflect the *current* context.

The *Plan Generator* takes the CPSMs, produces and validates a configuration plan, and sends this plan to the *Adapter* component. If the application is already running, this plan will describe the necessary changes to adapt the running configuration to the better configuration provided by the reasoner for the current execution context. The adapter is then faced with the difficult task of assessing if it is worth doing the adaptation of the running configuration. Should it decide to adapt the deployed application, it will then interface with the execution ware to enact the necessary changes.

The execution ware has four PAASAGE components: The *Cloudiator* interface, the *Metric Collector*, the *SRL Adapter*, and the *User Interface*. Cloudiator receives the deployment plan from the adapter and enacts the deployment of the application on the selected providers. The montiored metric values from the running application are collected by the Metric Collector responsible for distributing the metric values to the subscribers in the upperware and to the Scalability Rule Language (SRL) Adapter. This adapter is responsible for scaling the running application within a single provider; essentially by requesting the Cloudiator to start copies of running virtual machines. Such platform level scaling can only happen within the multiplicity intervals set by the reasoner in the current *configuration*. If the upper bound of the interval is reached and further scaling is necessary, the current *configuration* is no longer valid, and an exception is triggered that propagates back to the upperware Adapter that may have pending a more suitable *configuration* pre-computed by the reasoner. If that is not the case the reasoner is requested to produce a better *configuration* for the current execution context.

2.2 Components

CAMEL Textual Editor

The CAMEL Textual Editor allows to specify multiple aspects of cross-cloud applications, such as provisioning, deployment, service level, monitoring, scalability, providers, organisations, users, roles, security, and execution.
The CAMEL Textual Editor is based on the Eclipse Modeling Framework (EMF)[13] along with Object Constraint Language (OCL), Xtext[14].
EMF is a modelling framework that facilitates defining DSLs. It provides the Ecore reflexive metamodel, which is an Ecore model that conforms to itself (*i.e.*, it is reflexive). The CAMEL metamodel is an Ecore model that conforms to the Ecore metamodel.
OCL is a declarative language for specifying expressions on metamodels that are evaluated on models conforming to these metamodels. Eclipse OCL[15] is a tool-supported implementation of OCL that integrates with EMF. The CAMEL metamodel is annotated with OCL expressions for capturing (part of) the semantics of cross-cloud applications and guaranteeing the consistency, correctness, and integrity of CAMEL models at both design-time and run-time.
Xtext is a language development framework that is based on- and integrates with EMF. It facilitates the implementation of Eclipse-based IDEs providing features, such as syntax highlighting, code completion, code formatting, static analysis, and serialisation. The concrete syntax of CAMEL is a textual syntax defined as an Xtext grammar. The textual syntax was preferred over the graphical syntax by the use case partners in PaaSage.
Thanks to the combination of EMF, Eclipse OCL, and Xtext, the CAMEL Textual Editor allows users not only to specify CAMEL models but also to syntactically and semantically validate them.
The installation tutorial for the CAMEL Textual Editor along with the documentation of the language are available at: http://camel-dsl.org/documentation/

Input parameters

No input parameter is consumed by the component.

Output parameters

No output parameter is produced by the component. CAMEL models can be saved in textual syntax (.camel) or serialised to XMI (.xmi).

External dependencies

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

Known limitations

No limitations are known for the component.

[13] https://www.eclipse.org/modeling/emf/
[14] https://eclipse.org/Xtext/
[15] http://wiki.eclipse.org/OCL

Social network

The PaaSage Social Network (SN) is a social platform targeted at the creation of a community of users and developers. It enables users to exchange their experience and knowledge with respect to the PaaSage platform. Through the PaaSage SN, users can connect to other SN users and view their contributions, join groups, deploy applications, navigate through historical knowledge of previously executed application models, create application models by also utilizing useful modeling recommendations provided by the SN as well as comment and rate such applications models.

The registration section is kept simple by enabling the user to provide only the necessary details for the sign up process (display name, email address, username and password). However, the easiest and quickest way for a new user of the SN to log-in is to make use of their Facebook or Twitter account. In that way an account is created in the SN taking information from the Facebook or Twitter account respectively.

Regarding the application models shared in the SN, the user can: i) search for a specific model, ii) apply filters such as the deployment cloud, the geography and the tags of a model and iii) visit the application model page of a model and have access to more details about it.

The social network can be used as a UI for the PaaSage platform and for the deployment of an application. The first step is the user to fill in their PaaSage platform credentials. These credentials will allow the social network to connect with the PaaSage platform using a RESTfull API. The next step is to upload the provider model to the PaaSage platform. The last thing for the application deployment is the model of the application itself. The deployment actually consists of two phases, the reasoning and the actual deployment. In the model page of an application there is a button for each phase.

SN is available for users (`socialnetwork.paasage.eu`) to create accounts and exploit the functionality exhibited by the SN.

Input parameters

The user of the SN has to fill in a register form in order to create a profile. Moreover, application models must be uploaded to the SN to be shared with other users. Finally, the users have to specify the PaaSage platform credentials in order to use it for application deployment and the cloud credential credentials in order the necessary virtual machines to be created.

Output parameters

No output parameter is produced by the component. However, the uploaded application CAMEL model is persisted into the Metadata Database.

External dependencies

Library	*Description*	*License*	*Availability*
Elgg	social networking engine	GPLv2	`https://elgg.org/`

Known limitations

This component can be fully used.

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 the Hibernate Query Language (HQL), which is a higher-level language providing an SQL-like syntax but operating over meta-model elements and not tables. 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. Please note that this way of interaction is solely possible via the Luna version of the Eclipse Environment, which maps to CDO version 3.4.0.

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[16]. Please note that the CDOClient is compatible to interact with the CDOServer, unless the CDO version supported in either component is modified. This is in contrast to the case of the graphical based interaction where the user is responsible for ensuring the use of the right CDO version.

To be continued...

[16] https://tuleap.ow2.org/plugins/git/paasage/cdo_client?p=cdo_client.git&a=blob_plain&h=1d0abdaaea94d30df995ecc4eb1f028009aee745&f=documents/CDOClientDocumentation.pdf&noheader=1

...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, where the values to be provided are hsqldb and mysql, respectively.
- 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. The respective values to be provided are db and hibernate for the two kinds of stored supported, respectively.
- port - The number of port to which the server listens for incoming connections by clients.
- logging - Indicates whether logging should be set on or off.
- security - Indicates whether secured access to the CDOServer and its underlying repository is enabled (value of “on”) or not (value of “off”). Please see more details in [6] D4.1.2 deliverable of PaaSage.

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. In case the component goes down (in very limited situations), it can be easily restarted without losing the actual content of the underlying model repository.

CDO Client

This is a client component for the CDO Server enabling the storage and updating of models and the running of queries over the contents of underlying CDO model repository. As diverse requirements have been raised by the PaaSage component developers with respect to the different kind of model management functionality that needed to be implemented for facilitating their interaction with the CDO model repository, the CDO Client exposes methods which support the following tasks:

- EPackage registration mapping to the domain code to be manipulated
- textual (restricted to CAMEL models) or XMI model storage into a CDOResource with a particular name
- model export in both textual or xmi forms from a CDOResource to the file system. If the model to be exported references other models, then there are options to also export these models even in a recursive manner
- CDO repository export into a zip file with models in either textual or xmi form for back-up purposes
- CDO repository (in the form of a zip file) import (see previous item for the content of this zip file)
- xmi to textual model transformation
- query evaluation over the CDO model repository. Different query dialects are supported (OCL, SQL, HQL) depending on the type of store used to realise the model repository (see documentation of CDO Server).
- model loading from a path in the file system or from a specific URL
- model validation according to Ecore semantics and the OCL constraints posed
- deletion of models/objects
- transaction/view opening
- transaction/view closing
- CDOSession closing - this is the last step in the exploitation of the CDOClient as the session with the CDOServer is closed. The session is opened with the instantiation of the CDOClient.

A detailed documentation of this component is available in the PaaSage repository[17].
Please note that in case the CDOServer is configured with security on, then the CDOClient can be initiated with a respective userName and password to be used for authentication. If authentication is successful, then the respective session keeps the user credentials until the very end of the corresponding CDOClient object lifetime. This means that during the lifetime of the CDOClient instance/object, the user or program will be granted access to those resources on which it has the respective rights. There is no error to provide credential information even if CDOServer is not configured for security but this information is actually ignored.

To be continued...

[17] https://tuleap.ow2.org/plugins/git/paasage/cdo_client?p=cdo_client.git&a=blob_plain&h=1d0abdaaea94d30df995ecc4eb1f028009aee745&f=documents/CDOClientDocumentation.pdf&noheader=1

...CDO Client

This component can be configured in many ways mainly for properly interacting with a CDOServer. The properties that can be configured are the following:

- host – the IP of the host on which the CDO Server runs
- port – the port on which the CDO Server listens
- repository – The name of the repository to be created (default is "repo1")
- logging – Indicates whether logging should be set on or off (default is off).

Input parameters

user Name	The user name in case the security feature of the CDO Server is on.
password	The password in case the security feature of the CDO Server is on. This and the previous parameter represent the user credentials for authentication and can be used in the instantiation of the CDO Client.
property File Path	The path in the file system where the .properties configuration file for the CDO Client can be found. This parameter can be used in the constructor of the CDO Client.

Output parameters

We are talking about a component that provides multiple functionalities in the form of different methods. In this respect, the output depends on the respective method called. This was also the case for the input of this component, where we have chosen to explain the parameters for its construction.

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`
Log4j	Apache logging library	Apache Software Licence 2.0	`http://logging.apache.org/log4j/2.x/`
commons-logging	Apache logging library	Apache Software Licence 2.0	`https://commons.apache.org/proper/commons-logging/`
javatuples	Tuple management	Apache Software Licence 2.0	`http://www.javatuples.org/`

Known limitations

Textual syntax is only supported for CAMEL models. It is possible to use a textual syntax for another DSL but this would require some modifications in the code and in the maven dependencies file.

Importer

This component can be exploited for importing various types of models (basic models as well as provider and use case models) into a CDO Repository by also respecting the model storage guidelines specified in the context of WP4 (see D4.1.2 Deliverable [6]. In particular, this component generates the appropriate CDO resource folder structure and imports the models into resources which are stored at particular points inside this structure. Moreover, some of the CAMEL models provided might also be split into two or more parts.
More details about the main functionality exposed by this component are given below:

- import provider models from the file system into the CDO Repository
- import use case models according to the model storage guidelines by splitting the organisation information into a seperate CDO resource with respect to the rest of the information involved in the respective CAMEL model
- import geo location models into the CDO repository which are derived from the FAO geopolitical ontology[18]
- import a basic metric model from the file system into the CDO Repository. This model can be exploited for expressing Service Level Objectives (SLOs) and optimisation requirements for multi-cloud applications.
- import a basic security model which is generated from an excel file corresponding to the Cloud Controls Matrix[19] of Cloud Security Alliance. This model can be exploited for expressing high-level security requirements and capabilities in the form of security controls for multi-cloud applications and cloud providers, respectively.

A detailed documentation of this component is available in the PaaSage repository[20].

There is no configuration for this code but as it depends on the CDOClient code, the latter code should be properly configured. More details about this can be found in the documentation of this latter component in this deliverable as well as in the README file of this component in the respective git repository[21].

Input parameters

inCDO	This boolean parameter is used in the constructor of the Importer in order to denote whether the models are to be imported in the CDO Repository. If this is not the case, then only the models generated from non-XMI files (i.e., security and location models) are actually stored in the file system.
hibernate	This boolean parameter is used in the constructor of the Importer to denote the actual realisation of the CDO Repository. A value of true denotes that a HibernateStore is exploited, while a value of false denotes that a DBStore is exploited. This parameter is required in order to enable to formulate and perform the appropriate queries in the dialect that is actually being supported from the underlying CDO model repository.
resourcePath	The path in the file system where the XMI file or a directory of XMI files for the models to be loaded, processed and stored in the CDO Repository should be situated.

To be continued...

[18]http://www.fao.org/countryprofiles/geoinfo/en/
[19]https://cloudsecurityalliance.org/group/cloud-controls-matrix/
[20]https://tuleap.ow2.org/plugins/git/paasage/cdo_client?p=cdo_client.git&a=blob_plain&h=1d0abdaaea94d30df995ecc4eb1f028009aee745&f=documents/CDOClientDocumentation.pdf&noheader=1
[21]https://tuleap.ow2.org/plugins/git/paasage/cdo_client?p=cdo_client.git&a=blob_plain&h=45a9c2d3daa88a7269d3007f91fe4ec1907faa26&f=README&noheader=1

...Importer

Output parameters

All main methods of this component return a boolean value which indicates whether the storage of the respective model loaded/processed or generated has been successful.

External dependencies

Library	*Description*	*License*	*Availability*
Jena Core	Jena Core Library for RDF Management	Apache Software Licence 2.0	https://jena.apache.org/
Apache POI	Java API for Microsoft Documents	Apache Software Licence 2.0	https://poi.apache.org/

Known limitations

Currently tested on the Scalarm use case model. As now use case models are separated from the organisation model of the provider of the use case, the respective code needs to be updated (simplified). The update should also consider the import of user-specific provider models.

Knowledge Base (KB)

The Knowledge Base is the means via which added-value knowledge out of the CDO (model) repository can be derived. The derivation relies on rules which are incorporated in the Knowledge Base and which, for the current moment, enable the derivation of best deployment facts for applications and their components. Such derivation relies mainly on the past execution history of the current application/task or of other applications/tasks that are similar to the current one.
The rules have been designed carefully according to a specific domain model based on a particular mode of interaction with the CDO repository. In particular, the rules rely on a statically provided CDOSession object from which the respective CDO repository information can be obtained via queries. Rule facts rely on the domain model which have been carefully designed so as to reduce the amount of communication needed with the CDO repository by storing handles or CDOIDs to the actual objects stored in the CDO repository. This enables browsing the facts generated and only when such facts need to be processed, an attempt to actually retrieve the CDO object of interest is performed.
The current functionality of this Knowledge Base component is related to the management of knowledge bases and respective sessions operating over them. For an explanation of the lifecycle of knowledge bases and sessions as well as for inspecting examples of interaction with the Knowledge Base, the reader should consult the (full) component documentation at: `https://tuleap.ow2.org/plugins/git/paasage/knowledge_base?p=knowledge_base.git&a=blob_plain&h=84e22851678ae1bfd9a4292701f0cfd6a25a8387&f=documentation.doc&noheader=1`.
The following kinds of functionality are currently supported:

- KnowledgeBase creation
- loading of rules in an existing KnowledgeBase
- StatefulKnowledgeSession creation for an existing KnowledgeBase
- firing of all rules of a KnowledgeBase in the context of a specific StatefulKnowledgeSession
- retrieval of current content produced within a StatefulKnowledgeSession
- domain object storage in the context of a StatefulKnowledgeSession
- query evaluation over the current content of a StatefulKnowledgeSession
- StatefulKnowledgeSession deletion
- KnowledgeBase deletion (leading also to the deletion of all of its StatefulKnowledgeSessions)

Two main clients are offered by this component which can be exploited for the knowledge base management. The StandaloneClient is a thick client directly interacting with an internal Knowledge Base engine. As such, all needed computations (e.g., firing of rules) occur at the client side. The RestClient is a Restful client which exploits the jersey java library to invoke methods exposed by a RestService which encapsulates the facilities of a Knowledge Base engine. Such a client is thus thin as the heavy work is performed at the server side while the client just steers the calling of the server methods in order to properly manage its knowledge bases. Both clients supplied offer the same set of methods which differ only in their signature due to the different technologies employed. The base KB engine exploited is the Drools Engine[22].

This component does not need configuration on its own. However, as it exploits the CDOClient for interacting with the CDO Repository, the latter component needs to be properly configured. The way to perform this is described in the documentation of the CDOClient component in this deliverable.

To be continued...

[22] `www.drools.org`

...Knowledge Base

Input parameters

The constructors for the two clients do not take any input parameter. In the sequel, we mainly focus on the public methods offered by the StandaloneClient for simplification reasons (more complicated parameters used in the other client) as well as by relying on the fact that the signatures between the two clients are quite similar.

kbName	The name of the KB. This parameter is used in all the public methods provided by this component due to the way knowledge base management is performed.
sessionName	The name of the session which is used for interacting with a KB. Similarly to the previous case, this parameter is used in almost all public methods of this component. Such methods are strictly related to the management of the session and not the knowledge base itself.
query	The content of the query to be performed over a specific session on a certain KB.
parameters	The parameters to the query to be posed.
objects	The objects to be added to a particular session.
dirPath	The path to the file system from which rules can be loaded on a certain knowledge base.
globalName	The name of a global object to be passed to a session and thus be exploited during the rule firing.
object	The global object to be exploited in the firing of rules for a specific session (in our case to interact with the CDO Repository).

Output parameters

Almost all main methods of the Standalone Client of this component return a boolean parameter indicating a success of the method call. Only one parameter returns an array of the objects that are stored in a particular session.

External dependencies

Library	*Description*	*License*	*Availability*
Drools	The Drools KB Engine	MIT Licence	`http://www.drools.org/`
JAXB API	Java Architecture for XML Binding API	CDDL v1.1 & GPL v2	`https://jaxb.java.net/`
Slf4j	Simple Logging Facade for Java	MIT Licence	`http://www.slf4j.org/`
JUnit	JUnit Testing Framework	Eclipse Public Licence v1.0	`http://junit.org/junit4/`
Jersey	RESTful Services Framework for Java	CDDL v1.1 & GPL v2	`https://jersey.java.net/`

Known limitations

Rule Processor

The Rule Processor component processes the output of the CP Generator in order to align the generated models—*CP Model* and *PaaSage Application Model*—with user requirements expressed in the original CAMEL model. The result are modified models, if required, that satisfy these user requirements. User requirements, which can be directly defined in the CAMEL model, are currently limited to Cloud provider preferences. Cloud provider candidates pre-selected by the CP Generator are checked against the Cloud providers a user intends to deploy their application to. Users can either directly express their favoured Cloud providers, or alternatively, include in the CAMEL model an indication to use either public or private Cloud providers. Example: Candidates for Cloud providers are Amazon and GWDG, but the user requires the deployment to be limited to Amazon. The Rule Processor then checks all constraints of the constraint programming model, and removes redundant constraints and variables associated with GWDG. While performing the check against the user requirements, it can result in internal components being no longer assigned to a Cloud provider candidate. It arises when user-imposed requirements could not be fulfilled. The Rule Processor then returns with an appropriate warning. Otherwise, the output of this component is—if required an updated—*CP Model* as well as *PaaSage Application Model*.

Another check that the Rule Processor performs is related to service level objectives (SLO). Users can express, for example, that the number of instances for an internal component should be greater than 2, but no larger than 10. The Rule Processor evaluates the constraints associated with such a SLO, and—if required—adapts the domain range. Users are informed about these updates. If the SLO cannot be satisfied, the Rule Processor returns an error message. SLOs are checked automatically by the Rule Processor if they are included in a CAMEL model.

The Rule Processor supports two modes of operation: as a stand-alone component called via the command line, and as a daemon service that subscribes and publishes onto ZeroMQ message queues. In the latter case, the Rule Processor expects a properties file named `wp3_profiler.properties` that sets the relevant queues. If the file is not present, default values are used.

In summary, the Rule Processor acts as a component that can be applied by users to ensure that their requirements are correctly expressed by the constraint programming model. If requirements can not be fulfilled, users are in need to adapt their CAMEL model based on information presented by the Rule Processor.

Input parameters

-m <ModelId>	A string that represents an identifier in CDO Server of a CAMEL model related to the application being deployed.
-c <CDOId>	A string that represents an identifier in CDO Server of the models generated and stored by the CP Generator, namely *PaaSage Application Model* and *CP Model*.
-o <OutputFile>	A string that represents an absolute file system path. If the parameter is not given, it defaults to `rp_output`. The file contains either the original CDO identifier (cf. <CDOId>) or a new CDO identifier when the constraint programming model was modified; new identifiers end with the suffix *v2*.
-d	If the parameter is given, the Rule Processor starts as a service and subscribes and publishes to ZeroMQ message queues defined in the properties file `wp3_profiler.properties`. If not present, default values are used. The Rule Processor then subscribes to the queue *startSolving* on `tcp://localhost:5544`, and publishes to the queue *RPSolutionAvailable* on `tcp://*:5545`.

Output parameters

<OutputFile>	The file <OutputFile>, as provided via the `-o` parameter, contains either the original CDO identifier (cf. <CDOId>) or a new CDO identifier when the constraint programming model was modified. Per default, new identifiers end with the suffix *v2*.

To be continued...

...Rule processor

ZeroMQ messages	If the Rule Processor is started as a service, it publishes to the queue *RPSolutionAvailable* a multi-part message having the following structure: <ModelId>, <CDOId> (v2 if modified), <CDOId> (previous identifier). If an error occurs, the second parameter is replaced with the prefix `RP_ERROR` followed by an error message. The third parameter still contains the original <CDOId>.

External dependencies

Library	*Description*	*License*	*Availability*
commons-cli 1.3.1	An Apache Commons library that offers an API for parsing command line options. The library is used to process and validate given input parameters.	Apache License v2.0	`https://commons.apache.org/proper/commons-cli/`
commons-jexl 2.1.1	An Apache Commons library that implements an expression language to allow scripting features, amongst others, in Java. The library is used to dynamically check mathematical formulas underlying service level objectives given in a CAMEL model.	Apache License v2.0	`http://commons.apache.org/proper/commons-jexl/`
Log4j 1.2.17	An Apache library to provide a standardized way of logging information at runtime. The component is used to print progress, modifications, and results while the Rule Processor is validating a given constraint programming model based on a CAMEL model.	Apache License v2.0	`http://logging.apache.org/log4j/1.2/`
JUnit 4.12	A framework to write unit tests for Java. The framework is, in particularly, used for testing the linear algebra functions used in conjunction with the JEXL library.	EPL v1.0	`http://junit.org/junit4/`
jeromq 0.3.5	The library provides a pure Java implementation of the ZeroMQ protocol.	LGPL v3.0	`https://github.com/zeromq/jeromq/tree/v0.3.5`

Known limitations

The component currently is limited to user requirements based on Cloud providers and service level objectives; requirements that arose during the project. The list of user-imposed requirements could be extended in the future.

Meta Solver	
The MetaSolver Component acts as a gateway to the different solvers which provide the reasoning capabilities in the PaaSage platform. It takes the refined *CP Model* and *PaaSage Application Model* output by the Rule Processor and prepares the *CP Model* for input into the solvers. For a first time deployment, the Metasolver verifies and creates, if necessary, default values for all metric variables in the *CP Model*. For reconfiguring an existing PaaSage application, it takes the run time metrics pushed by the Metrics Collector and maps the values to the correct metric variables in the *CP Model*. The Metasolver supports two modes of operation: as a stand-alone component called directly via the command line, or as a daemon service that communicates via ZeroMQ messages to other PaaSage components. A configuration file (`wp3_metasolver.properties`) is used to configure the application. The properties file should be located under the *PAASAGE_CONFIG_DIR* defined for the PaaSage Platform. If the file is not present, default values are used. The Metasolver subscribes by default to the Rule Processor via the queue `computeSolution` on `tcp://localhost:5556`; to the Metrics Collector on `tcp://localhost:5552`; to the Adapter via the queue `newCamelDeploymentAvailable` on `tcp://localhost:5550`; to the MilpSolver via the queue `MILPSolutionAvailable` on `tcp://localhost:5540`; to the CPSolver via the queue `CPSolutionAvailable` on `tcp://localhost:5541` and it publishes by default to all solvers via the queue `startSolving` on `tcp://localhost:5547` and to the Solver-to-Deployment component via the queue `newSolutionAvailable` on `tcp://localhost:5544`. ZeroMQ messages from the Rule Processor should contain in this order: a string representation of an identifier in CDO Server to the CAMEL model related to the application being deployed, a string representing an identifier in CDO Server to the *CP Model* to be solved and a string representation to the CDO directory identifier of the modified *CP Model*. Messages from the Adapter and solvers should contain in this order: a string representation of an identifier in CDO Server to the CAMEL model related to the application being deployed and a string representing an identifier in CDO Server to the *CP Model* to be solved. Messages from the Metrics Collector should contains a string representation of the new value for the Metric identified in the message queue. The metasolver publishes multi-part messages to the solvers which contain, in this order, a string representation of an identifier in CDO Server to the CAMEL model related to the application being deployed, a string representing an identifier in CDO Server to the *CP Model* to be solved and a time stamp to the last *Solution* in the *CP Model*. Messages published to the Solvers-to-Deployment component differ slightly in that the time stamp is replaced by a string representation to the CDO directory identifier of the modified *CP Model*. If an exception occurs during processing, Metasolver publishes an error message to the subscribers which would be intercepted by the PaaSage platform for feedback to the application designer.	
Input parameters	
-daemon	This parameter is only applicable to the *develop* branch of the Metasolver which supports ZeroMQ. When this parameter is specified, the Metasolver starts as a service. It subscribes and publishes to ZeroMQ message queues defined in the properties file `wp3_metasolver.properties`. If the file is absence, default values are used.
Camel Model Id	If started in command line mode, the first argument is a string representing an identifier in CDO Server to the CAMEL model related to the application being deployed. This argument is mandatory.
CP Model Id	If started in command line mode, the second argument is a string representing an identifier in CDO Server to the *CP Model* to be solved. This argument is mandatory.
Target Solver	If started in command line mode, the third argument is an optional integer parameter specifying the target solver or solvers to call. The options are: 0 denotes that call all available solvers, 1 denotes call MILPSolver, and 2 denotes call the CPSolver, and so forth.
Local Path To Metrics File	If started in command line mode, the fourth argument is an optional parameter specifying the path to file containing key value pairs of metric variables and values to be added to the *CP Model* before submission to the solver/s.

To be continued...

...Meta solver

Output parameters			
None	It is expected that the Solvers will generate new output and write directly to the CDO server. An error message will be logged if the Metasolver encounters processing error.		
External dependencies			
Library	*Description*	*License*	*Availability*
commons io 2.4	A library of utilities to assist with developing IO functionality.	Apache License 2.0	`http://commons.apache.org/proper/commons-io/`
eclipsesource minimal-json	A fast and minimal JSON parser.	MIT Licence	`https://github.com/ralfstx/minimal-json/releases/tag/0.9.1`
javatuples 1.2	A simple JAVA library to provide a set of java classes for working with tuples.	Apache License 2.0	`http://www.javatuples.org/`
jeromq 0.3.5	The library provides a pure Java implementation of the ZeroMQ protocol.	LGPL v3.0	`https://github.com/zeromq/jeromq/tree/v0.3.5`
JUnit 4.11	A framework to write unit tests for Java.	EPL v1.0	`http://junit.org/junit4/`
Log4j 1.2.17	An Apache library to provide a standardized way of logging information at runtime.	Apache License v2.0	`http://logging.apache.org/log4j/1.2/`
LOGBack 1.1.2	A logging framework.	Dual EPL v1.0 and LGPL 2.1	`http://logback.qos.ch/index.html`
SLF4J 1.7.10	A JAVA logging API.	MIT Licence	`http://www.slf4j.org/`
zeromq 3.1.0	Java binding for ZeroMQ.	LGPL v3.0	`https://github.com/zeromq/jzmq`
Known limitations			
The MetaSolver is limited to the use of the two solvers (MILP and CP) for this prototype. It also expects the Metrics Collector to provide values for `all` metrics identified in the CAMEL model.			

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.			

Constraint Programming solver (CPSolver)

The Constraint Programming (CP) solver is yet another solver which can be exploited for performing reasoning over the space of possible deployment configurations for an application. The main difference with other solvers is that it can include non-linear constraints and utility functions while it is also able to handle both integer-based and real variables. In the context of the PaaSage project, the functionality of the CP solver is similar to the one of the other solvers. This means that the procedure it follows to solve a CP model is the following:

- The CP solver receives the CDO repository path of the CP model, loads it into main memory and transforms it internally to the appropriate Java structures required by the solver for representing the actual constraint problem
- The constraint problem is solved
- The solution is written back to the CP model in CDO

Moreover, the latest developments of this solver led to its daemonisation in order to enable a queue-based mechanism of interacting with it. The main component interacting with the CP Solver is the Meta-Solver which requests the solving of a CP model in one queue and receives the result in another one. 0MQ[23] has been used for the realisation of the message queue mechanism, equivalently to all other components of the platform that have been daemonised.
The latest solver developments also include the capability of the solver to address the existence of multiple goals/optimisation requirements. In this case, the solver adopts the Simple Additive Weighting technique in order to transform all the goals to a single one to be maximised. The developments also involve [7] the possibility to interact with the Knowledge Base (KB) (see D4.1.2 deliverable [6]) in order to fix some parts of the constraint problem to be solved by exploiting best deployments for components and applications, derived as facts from the KB. This saves the solving time as well as takes into account the previous experience in deploying and executing the current or similar applications to the current one.
Out of the context of the PaaSage platform main flow, the CP solver has been updated [8] to exhibit many other interesting features which include:

- conjuctive selection of both SaaS and IaaS services for a multi-cloud application
- selection of software services for application tasks can rely on both externally offered SaaS as well as internal software components deployed as services
- capability to include functions which explicate the way performance on lower levels of abstraction (i.e., IaaS services) can influence the performance on higher levels of abstraction (i.e., SaaS services)
- capability to handle quality constraints on different levels which include security constraints in the form of security controls and Service Level Objectives (SLOs)

Concerning its implementation, the CP solver has been realised via the Choco constraint programming Java library[24] in conjunction with the Ibex solver[25] for the addressing of real expressions and variables.

To be continued...

[23] `zeromq.org`
[24] `www.choco-solver.org`
[25] `ibex-lib.org`

...Constraint Programming solver

Input parameters			
CP Model	The path to the CDO model repository from which the CP Model can be retrieved as well as updated		
CAMEL Model	The path to the CDO model repository from which the CAMEL Model can be retrieved. This parameter is actually not required by the CP Solver but is exploited mainly in its daemonised usage to distinguish the respective application for which the deployment has been reasoned as an extra knowledge that is propagated back to the Meta-Solver along with the deployment solution.		
timestamp	A timestamp value in the format of a long which can be used for signifying to the CP solver which metric variable values to be used for solving the CP model. These metric variable values are thus obtained from a previous solution of the CP model.		
Output parameters			
Solving Outcome	A boolean value indicating whether the CP model was feasible and a solution was produced for it or infeasible. The next parameters characterise mainly the daemonised version of the CP solver.		
CP Model	The path to the CDO repository where the updated (with the new solution discovered) CP model can be retrieved.		
CAMEL Model	The path to the CDO repository where the CAMEL model of the corresponding application can be found.		
External dependencies			
Library	*Description*	*License*	*Availability*
Choco	Choco Constraint Programming Java Library	BSD	http://www.choco-solver.org
Ibex	Ibex Constraint Programming solver for real variables	LGPLv3	http://www.ibex-lib.org
ZeroMQ	Messaging Middleware	LGPLv3	http://www.zeromq.org
Log4j	Apache Logging library	Apache 2.0	http://logging.apache.org/log4j/2.x/
Known limitations			
None.			

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 D3.1.2 [4]. 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 aggregated metric values that may change over time, e.g. the average number of users of the application per day. 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 for the current execution context as soon as a solution better than the current best configuration is found. The solver will receive updated metric values from the Metric Collector, 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 CP-model output from the Profiler 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 CP-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 application 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 D3.1.2 [4].	
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 the CP-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.

To be continued...

...Learning Automata based solver

External dependencies			
Library	*Description*	*License*	*Availability*
NLopt	Open-source library for non-linear optimisation, providing a common interface for a number of different free optimisation routines available on-line 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`
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 was released as part of the prototype and the interface has been tested to ensure that the solver is correctly built for a given 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			
Solution Model Id	A string that represents the identifier in CDO Server of a Camel Model related to the solution being deployed.		
Output Path	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 CAMEL deployable model.			

Adapter

The purpose of the `Adapter` is to transform the currently running application configuration into a target configuration in an efficient and safe way. The component has the following responsibilities: (1) producing and validating reconfiguration plans, (2) applying the plans to the running system, and (3) maintaining an up-to-date representation of the current system state. The component typically operates as follows: (1) it obtains a target deployment model from the Reasoner, (2) it produces a reconfiguration plan using the `Plan Generator`, (3) it validates that the plan is acceptable, (4) it applies the plan to the running system using the executionware, (5) it monitors the running system to detect changes, such as executionware-triggered scaling, and (6) it ensures that changes are reflected in the CAMEL-based system representation. The current implementation supports parallel plan execution, full causal connection between the running system and the CAMEL-based representation, and plan validation based on historical reconfiguration information. The component interacts with upperware using zeromq and with executionware using 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*
jeromq	Pure Java implementation of libzmq	LGPL 3.0	`https://github.com/zeromq/jeromq`
log4j 1.2.17	Logging library for Java	Apache License 2.0	`http://logging.apache.org/log4j/1.2/`
JGraphT	JGraphT libraries	EPL 1.0 and LGPL 2.1	`http://jgrapht.org/`
JSON.simple	A simple Java toolkit for JSON	Apache 2.0	`https://code.google.com/archive/p/json-simple/`
Halbuilder Standard 4.0.1	Serializer/deserializer factory supporting HAL+XML and HAL+JSON	Apache	`http://www.gotohal.net`
Gson 2.2.4	Java library that can be used to convert Java Objects into their JSON representation	Apache 2.0	`http://code.google.com/p/google-gson/`
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

None.

SRL Adapter			
The SRL Adapter is a sub component of the Adapter. Similar to the adaptation manager, the SRL Adapter is responsible to transform the current running configuration to a new target configuration. It's scope is however limited to monitoring data and aggregation functions. For this purpose, the SRL Adapter extracts the monitoring description provided by the Scalability Rule Language inside the CAMEL language, and provides the information to the ExecutionWare Frontend. In subsequent runs, the deviation between plan (model) and actual state (ExecutionWare Frontend) is calculated, an the actual state is adapted.			
Input parameters			
ZeroMQ	The SRL adapter receives a zeromq message from the adapter as soon as the adapter run has finished. The zeromq message contains the current execution context and basic properties of the current model.		
MDDB	The SRL adapter has a connection to the MDDB from which it retrieves the Scalability Rule description.		
Output parameters			
REST-Interface	Uses the REST Interface of the ExecutionWare Frontend to enact monitoring and aggregation needs.		
External dependencies			
Library	*Description*	*License*	*Availability*
jeromq	zeromq Library for Java	Mozilla Public License 2.0	`https://github.com/zeromq/jeromq`
Known limitations			
None			

Colosseum - ExecutionWare Frontend			
The Cloudiator [26] component Colosseum acts as Executionware Frontend in PaaSage. It 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 Sword. It orchestrates the allocation of virtual machines that is enacting using the Sword component of Cloudiator, and the deployment of the application that is handled by Lance.			
Input parameters			
REST-Interface	The REST-Interface offers the capabilities of the ExecutionWare to the Adaption Manager of the Upperware.		
Output parameters			
Java	The Executionware Frontend communicates with Sword using direct JAVA-calls.		
REST-Interface	The web-based interface provides the adaptation manager of the Upperware with control over the Executionware.		
RMI	To orchestrate the deployment of the application, Colosseum communicates with the lifecycle-agents (Lance) deployed on the managed virtual machines.		
External dependencies			
Library	*Description*	*License*	*Availability*
Hibernate Entityman-ager	Database	LGPL 2.1	http://www.hibernate.org
MariaDB Java Client	Database	LGPL 2.1	https://mariadb.com
Java Ham-crest	Testing	BSD	www.hamcrest.org
Google Guice	Dependency Injection	Apache License 2.0	https://github.com/google/guice
Google Guava	Helper Library	Apache License 2.0	https://github.com/google/guava
Apache Commons	Helper	Apache License 2.0	http://commons.apache.org/
type-parser	Helper	MIT	https://github.com/drapostolos/type-parser
reflections	Helper	WTFPL	https://github.com/ronmamo/reflections
JGraphT	Graph Library	LGPL	http://jgrapht.org/
Known limitations			
None			

[26] https://cloudiator.github.io

ExecutionWare UI			
The ExecutionWare UI provides a graphical interface for the REST interface provided by the ExecutionWare component Colosseum. It allows the users of PaaSage to see the current deployment status of the application. In addition it allows manual interaction with the ExecutionWare allowing to trigger e.g. manual scaling operation. Furthermore, it allows the user to access the monitoring data collected by the monitoring infrastructure in a graphical way.			
Input parameters			
REST-calls	The UI communicates with the Colosseum component by fetching the JSON data of the REST Api using Ajax calls.		
Output parameters			
Web	The UI provides its information via Web pages viewable in a browser.		
External dependencies			
Library	*Description*	*License*	*Availability*
Apache 2	Web Server	APL 2.0	https://httpd.apache.org/
Angular.js	Javascript Framework	MIT License	https://angularjs.org/
PHP	Programming Language	PHP License v3.01	https://secure.php.net/
Known limitations			
None			

Sword - Abstraction Layer of Execution Engine			
The component Sword of the Cloudiator framework[27] acts as abstraction layer for the Execution Engine. It harmonises the different provider specific APIs of the cloud providers, thus enabling the Colosseum component to allocate virtual machines from different cloud providers. Currently supported APIs are: Openstack Nova, Flexiant FCO, Amazon EC2 and Google Compute Cloud.			
Input parameters			
Java	Sword receives virtual machine allocation request directly via Java calls from the Colosseum component.		
Output parameters			
Cloud API	Sword calls the different cloud provider APIs via an abstraction layer supporting several cloud middlewares.		
External dependencies			
Library	*Description*	*License*	*Availability*
jclouds	Cloud Abstraction	Apache License 2.0	`http://jclouds.apache.org/`
Overthere	Remote Shell Connection (SSH/WinRM)	GPL 2.0 (FLOSS Exception)	`https://github.com/xebialabs/overthere`
Google Guice	Dependency Injection	Apache License 2.0	`https://github.com/google/guice`
Google Guava	Helper Library	Apache License 2.0	`https://github.com/google/guava`
Known limitations			
None			

[27] `https://cloudiator.github.io`

Lance - Lifecycle Agent of Execution Engine			
The component Lance of the Cloudiator framework[28] acts as lifecycle-agent of the Execution Engine. It resides on all virtual machines managed by the ExecutionWare and is responsible for deploying the application onto the virtual machine. It supports two deployment modes: Docker and Plain.			
Input parameters			
Java RMI	Lance receives deployment requests via a Java RMI interface. This interface is called by Colosseum acting as ExecutionWare Frontend.		
Output parameters			
OS calls	Lance calls the underlying operating system to install and start the component it is deploying.		
External dependencies			
Library	*Description*	*License*	*Availability*
etcd (optional)	Key-Value Store	Apache License 2.0	`https://github.com/coreos/etcd`
etcd4j	Client for etcd	Apache License 2.0	`https://github.com/jurmous/etcd4j`
Docker (optional)	Container	Apache License 2.0	`https://www.docker.com/`
Known limitations			
None			

[28] `https://cloudiator.github.io`

Monitoring and Aggregation Infrastructure			
The monitoring infrastructure is responsible for monitoring the metrics defined in the application model to aggregate collected raw metrics and to communicate the results 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 (Visor[29]) 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. **Aggregation** The aggregation framework Axe[30] is responsible for aggregation the collected raw metrics. **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.		
REST	The sensor offer a rest interface allowing Axe and Colosseum to configure monitoring needs.		
RMI	The aggregation framework receives its aggregation tasks from Colosseum via a RMI-based interface.		
Output parameters			
Monitoring Data	The collector provides the MDDB with the monitoring data stored in the TSDB. In addition it provides the monitoring data to the Solvers using the zeromq message queue.		
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`
sigar	Monitoring Library	Apache License 2.0	`https://github.com/hyperic/sigar`
zeromq	Message Queue	LGPL	`http://zeromq.org/`
Known limitations			
None			

[29] `https://cloudiator.github.io`
[30] `https://cloudiator.github.io`

Metrics Collector

This component is mainly used during the monitoring of multi-cloud applications. It can actually be exploited in two ways: (a) as a code which can perform aggregations over raw measurements that are stored in a Time Series DataBase (TSDB); (b) as a code which is mainly used for storing measurements in the CDO Repository as well as reporting these measurements to interested subscribed components via the use of the ZeroMQ messaging middleware. This component is currently exploited via the second way as metric aggregation is performed by the Cloudiator framework.
Concerning metric aggregation, the component is able to produce aggregations for metrics within one cloud or across clouds. In the first case, it should be deployed on the same user VM where the local TSDB is deployed and run. In the second case, it should be deployed in the domain of the Executionware module. For the actual aggregation, two main TSDBs are currently supported: (a) KairosDB[31] and (b) InfluxDB[32]. The component applies a multi-threaded architecture to perform the aggregation where threads are created to manipulate the execution context of the application as well as the measurement of a specific composite metric pertaining to this context. As such, the first type of threads is responsible for the management of the aggregation while the second type for performing this aggregation. Local KairosDBs are exploited for local aggregation of measurements while the CDORepository for the cross-cloud aggregation. To support the aggregation, quite well-known statistics operations (e.g., mean, median, percentile, etc.) were either directly exploited or realised from scratch (in case of KairosDB and cross-cloud aggregation via CDO).
Measurement storage and publishing relies on the facilities offered by the CDOClient and the ZeroMQ middleware. Concerning the publishing, the current rationale is that when a measurement is to be stored in CDO, it is checked whether it maps to a metric which is directly associated to a Service Level Objective (SLO), optimisation requirement or nonfunctional event of a scalability rule. If this is the case, then the measurement is published and the respective subscribers, such as the Meta-Solver, can retrieve it.
More details about this component can be found in Deliverable D5.1.2 [5].

Metrics Collector is mainly configured via a file named as "eu.paasage.executionware.metric-collector.properties". This file contains the following properties that need to be set:

- db – it can take the values of “influx” or “kairos” mapping to the two TSDBs currently supported, i.e., InfluxDB and KairosDB, respectively.
- host – the name/IP of the host where the TSDB resides
- port – the port number on which the TSDB listens
- mode – the operation mode of the Metrics Collector which can take the values of either “local” or “global”. Local mode indicates that the Metrics Collector should be deployed and run in a user VM to perform local aggregations. Global model indicates that the Metrics Collector should be run in the domain of the Executionware module to aggregate cross-cloud measurements as well as to perform the measurements storage in CDO and their reporting.
- runServer – a boolean value is expected here to denote whether a publication server should also be executed in order to support the publishing of measurements. By default the value is false. This parameter should only be set on in the case of a global MetricCollector. Please note that in the most recent development of this component, the publishing can be configured also via the respective class (MetricStorage and its two actual implementations) which performs the storage and publishing of measurements.

To be continued...

[31] https://kairosdb.github.io/
[32] https://www.influxdata.com/

- runClient – a boolean value denoting whether a subscription client should also be executed in order to receive requests for measuring collections of metrics on behalf of a certain application instance. By default the value is false. This parameter should only be set on in the case of a global MetricCollector.
- directCall – a boolean value denoting whether the public methods of the Metrics Collector are to be called in a direct or indirect manner (e.g., via a command file). By default this property takes the value of false.
- dynamic – a boolean value is expected to denote whether the public methods are dynamically called by changing the content of the command file. Obviously, this property should only be set on if the value of the directCall configuration property is false. By default, this property takes the value of false.

Please note that as this component has to store measurements in CDO, it exploits a CDOClient component. As such, the latter component needs to be also properly configured. The way to perform this is described in the documentation of the CDOClient component in this deliverable.

Input parameters

property File Path	The path to the file system where the configuration file for this component is situated.
metric IDs	A list of CDOIDs for the metric instances that need to be measured. Depending on the method actually called, the respective semantics is different. If the metric instances need to be measured, then *readMetrics* has to be called. If the metric instances need to be deleted, then *deleteMetrics* has to be called. Please note that the update of metrics can be performed via a deletion and a subsequent insertion of the metric instances.
exec Context Id	The CDOID of the Execution Context for which the measurements need to be produced. Please remember that the execution context represents a certain multi-cloud application execution episode for which we need to store measurements as well as cater for triggering scalability rules, if needed.

Output parameters

All main methods of this component do not return any output parameter value.

External dependencies

Library	*Description*	*License*	*Availability*
InfluxDB	The InfluxDB TSDB	MIT Licence	`https://www.influxdata.com/`
KairosDB	The KairosDB TSDB	Apache Software Licence 2.0	`https://kairosdb.github.io/`
Docker-java	Java Docker API Client	Apache Software Licence 2.0	`https://github.com/docker-java/docker-java`
ZeroMQ	Messaging Middleware	LGPLv3	`http://www.zeromq.org`
Log4j	Apache Logging library	Apache 2.0	`http://logging.apache.org/log4j/2.x/`

Known limitations

3 Licence and governance

3.1 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[33], LGPL3.0[34] and Eclipse[35].

2. The license should protect the investment we have done in PAASAGE and should therefore be "weak copyleft"[36], i.e. if someone improves the PAASAGE platform code, these improvements should be released back as open source for others to use.

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 a majority of initiatives hosted by Apache Foundation. The Apache license is also used by other organisations such as Appscale[37] or Red Hat[38]. 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 desire for the common PAASAGE license was that it should be well know in the developer communities to facilitate easy adoption of the PAASAGE code base. The selection was therefore confined to the the list of recommended licenses that is published by Open Source Initiative[39] 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) [40] 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.[41] The MPL 2.0 license can be compatible with MPL 1.1 and GNU licenses, which is acknowledged[42] by the Free Software Foundation (FSF).

Common Public License (CPL) or Eclipse Public License (EPL) The EPL is an evolution of the CPL[43]. It is a file based weak copyleft license that is associated to the Eclipse projects. The EPL is incompatible with the widely used GPL.

[33] http://opensource.org/licenses/Apache-2.0
[34] http://opensource.org/licenses/LGPL-3.0
[35] http://opensource.org/licenses/EPL-1.0
[36] http://en.wikipedia.org/wiki/Copyleft
[37] www.appscale.com
[38] www.redhat.com
[39] http://opensource.org/licenses
[40] http://opensource.org/licenses/MPL-2.0
[41] https://www.mozilla.org/MPL/2.0/FAQ.html
[42] https://www.gnu.org/licenses/license-list.html#MPL-2.0
[43] http://www.ibm.com/developerworks/library/os-cpl.html

Common Development and Distribution License (CDDL) [44] 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 by the PAASAGE consortium members as the common licence for the PAASAGE software.

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[45] in the root directory of the source code. Each source file should contain the following text as part of the file header:

```
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/.
```

3.2 Best practice on governance of Open Source Software projects

Governance in software projects can be defined as *“the complex process that is responsible for the control of project scope, progress, and continuous commitment of developers”* [9]. In particular, the scope and purpose of the “governance model” for an open source project can be defined as follows[46]:

> *“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.* [10] and O’Mahony [11].

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 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 have 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.

[44] http://opensource.org/licenses/CDDL-1.0
[45] This file can be copied from https://www.mozilla.org/MPL/2.0/index.txt
[46] http://oss-watch.ac.uk/resources/governancemodels

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 [10]. 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.

As aforesaid, PaaSage as a research project has clear goals and structured plans, but on the other side aims to deliver an open source code base. Hence, it needs to ensure optimal source code quality in its code base, so as to enable a governance model that considers both perspectives. As part of Task 1.7 (Evaluation Framework) of PaaSage, a final list of specific software metrics will be defined for evaluating the efficiency and reliability software quality characteristics. Currently, the software metrics of availability and response time but also metrics related to code quality [12] are considered for evaluating the PaaSage platform's efficiency and reliability; key characteristics defined also in the ISO/IEC 25010:2011 software quality model. In terms of reliability Kotaiah *et al.* state: "The root causes of poor reliability are found in a combination of non-compliance with good architectural and coding practices" [13].

Although the standard defines software product quality characteristics, ISO has not yet provided concrete models and measurement tools. Hence, EU research projects are studied and reviewed in order to identify a concrete model and tools for assessing open source quality and the support capacity of the developers community. QualiPSo research project defined the OpenSource Maturity Model (OMM), an assessment model for evaluating Free/Libre Open Source Software (FLOSS) and in specific the FLOSS development process [14]. It defines a set of rules and guidelines describing how to conduct the assessment process. The key project objective is to build trust in FLOSS software, promoting the adoption of the OMM model by FLOSS communities as a basis for developing products efficiently and making their products trustworthy for customers [14]. QUALOSS EU research project defined a methodology and tools for improving the productivity and the quality of open software products, in a similar context to QualiPSo OMM. The QUALOSS quality model further supports though assessment of the support capacity of the open software community in terms of development and evolution of the open source software product [15]. Both quality models provide appropriate rules and guidelines for accomplishing source code quality evaluation in PaaSage. At the present stage, QUALOSS model is deemed more suitable for the task at hand since it addresses a key aspect of the "hybrid" governance model that needs to be adopted in PaaSage since it assesses also the support capacity of the developers community. Nevertheless, the study of quality models is an ongoing task in PaaSage that currently focuses also on the review of quality models via comparative studies, e.g. [16, 17].

O'Mahony has identified five core principles that participants to open source projects value as essential to good community management [11]. 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.

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 model described below in Section 3.3 will be in effect during the first two years after the end of the funded PaaSage project, *i.e.* until October 2018, and confirms 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 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 6. Some of these components exists as interfaces to 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 Currently, each component in the PaaSage Open Source Project has a named developer assigned and this developer is the project leader for that component, and represents the component's developer team when deciding on the future for the PaaSage platform.

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, 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 September 2016 external developers will be accepted as part of the PaaSage development team by the 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.

3.3 PaaSage Governance

The governance of the PaaSage open source project extends the best practices identified by explicitly naming the people filling the various roles for the first two years after the end of the project, i.e. until September 2018 unless they are replaced according to the procedures described below.

3.3.1 Component project leaders

The role of the component project leader is

- To allocate bugs reported for the component to one of the component developers
- To decide on proposed contributions from users of the component that are not yet developers
- To give *commit rights* to qualified developers
- To decide on features and further development directions for the component together with the developers who hold commit rights.

The list of component leaders is given in Table 1.

3.3.2 Board of Architects

The *Board of Architects* will be responsible for the overall management of the PaaSage code base, and its evolution with new features and components. The duty of the board is

- To decide on the inclusion of new components;
- To decide on the removal of some components;
- To approve replacements of component project leaders based on recommendations and requests from the component project team;
- To decide on the development tools to use;
- To promote the PaaSage platform to users and other projects.

Table 1: PaaSage components and project leaders

Component	Sub-repository	Project leader
CAMEL editor	camel	Alessandro Rossini
	camel_cdo_storage	Kyriakos Kritikos
Social network	social_network	Manos Papoutsakis
	paasage_frontend	Etienne Charlier
	xmi_to_camel_tool	Christian Perez
	paasage_rest_tester	Etienne Charlier
CDO Server	cdo_server	Kyriakos Kritikos
	importer	Kyriakos Kritikos
	knowledge_base	Kyriakos Kritikos
CP Generator	cp_generator	Lionel Seinturier
Rule Processor	rule_processor	Dennis Hoppe
Meta Solver	meta_solver	Shirley Crompton
MILP Solver	milp_solver	Kamil Figiela
CP Solver	cp_solver	Kyriakos Kritikos
LA Solver	la_based_reasoner	Geir Horn
	la_converter	Lionel Seinturier
Solver to Deployment	solver_to_deployment	Christian Perez
Adapter	adapter	Nikos Parlavantzas
Plan Generator	plan_generator	Shirley Crompton
Cloudiator interface	executionware_backend	Daniel Baur
Metric Collector	metrics_collector	Kyriakos Kritikos
SLR adapter	srl-adapter	Frank Griesinger
Execution ware user interface	executionware_ui	Daniel Baur
Common components	cdo_client	Kyriakos Kritikos
	upperware_metamodel	Lionel Seinturier
	pom	Etienne Charlier
PaaSage deployment	deployment_scripts	Etienne Charlier
	paasage_one_click_install	Etienne Charlier
Tools	user_cloud_provider _model_generator	Kyriakos Kritikos
	identity_management	Etienne Charlier

The initial Board of Architects consists of the PaaSage team members listed in Table 2 with addition of the above mentioned project component leaders. The board will strive to achieve consensus for its decisions. In the unlikely event that consensus cannot be reached, the board will vote. Any board member can at any time ask that a discussion is concluded by a vote. A quorum of 2/3 of the components must take part in the vote for it to be valid. Each person has cumulatively a number of votes corresponding to the roles the person has in the board according to Table 2.

The board will always meet in on-line meetings as needed to fulfil its duties as described above. There will also be quarterly on-line meetings of the board to assess status of the platform, and propose proactive actions. Furthermore, each component project leader may at any time ask the board to meet in order to resolve issues related to a particular component.

Table 2: PaaSage Board of Architects

Role	Votes	Name
Chair	4	Geir Horn
Co-chair	3	Christian Perez
	3	Jörg Domaschka
Social Network Moderator	1	Kostas Magoutis
Integration Manager	2	Stéphane Mouton
Component project leader	1	One per component as named in Table 1. A person representing multiple components has multiple votes.

4 Availability and installation

4.1 Availability of PaaSage Platform and source code

Both source code and ready-tu-use packaging of the PaaSage Platform must be broadly available. OW2[47] is an open source community providing services for open source projects such as source code repository to manage and control source code versions, and issues tracker. OW2 defines itself[48] as aiming to

"a) promote the development of open-source middleware, generic business applications, cloud computing platforms and b) foster a vibrant community and business ecosystem."

PaaSage as open source project is member of OW2. The entire source code of the PaaSage Platform is hosted in OW2 open source repository and available for download. In addition, the PaaSage Platform is automatically built and assembled from source code using tools and infrastructure provided by OW2. Details of the build and integration process can be found in Deliverable D6.2.2 [18].

OW2 also hosts a web page dedicated to PaaSage[49]. The page gathers information on the project and links to documentation[50], source code repository through Tuleap version control system[51] and downloadable PaaSage Platform installation script.

4.2 PaaSage Platform installation

PaaSage Platform installation and configuration has been primarily documented in Deliverable D8.3.1 [19]. In order to foster adoption of the PaaSage Platform, the team of WP9 - Training and Dissemination - has adapted content of D8.3.1 to produce training materials. The goal is to ease the PaaSage platform installation process by guiding users steps by steps. The resulting installation instructions are available from the PaaSage web site, both from the front page[52] and from the "Training materials" section of the site[53]. Training material and how it has been produced are described in Deliverable D9.3.2 [19].

5 Conclusion

This brief guide documents the components developed for the PaaSage platform, the governance of the open source project, and gives instructions on how to install and run the software, which is the actual deliverable.

[47] https://www.ow2.org
[48] https://www.ow2.org/bin/view/About/OW2_Consortium
[49] https://projects.ow2.org/bin/view/paasage/
[50] http://www.paasage.eu/training-materials/paasage-in-a-nutshell
[51] https://tuleap.ow2.org/plugins/git/?group_id=107
[52] http://www.paasage.eu, "Download the PaaSage platform"
[53] http://www.paasage.eu/training-materials/installing-and-configuring

References

[1] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer, “What is DevOps? a systematic mapping study on definitions and practices,” in *Proceedings of the Scientific Workshop Proceedings of XP2016*, XP ’16 Workshops, (Conference location: Edinburgh, Scotland, UK), p. 12:1–12:11, ACM, May 2016.

[2] A. Rossini, K. Kritikos, N. Nikolov, J. Domaschka, F. Griesinger, D. Seybold, and D. Romero, “D2.1.3 – CAMEL Documentation ,” paasage project deliverable, PaaSage project consortium, October 2015.

[3] K. Kritikos, M. Korozi, B. Kryza, T. Kirkham, A. Leonidis, K. Magoutis, P. Massonet, S. Ntoa, A. Papaioannou, C. Papoulas, C. Sheridan, and C. Zeginis, “D4.1.1 – Prototype Metadata Database and Social Network,” paasage project deliverable, PaaSage project consortium, March 2014.

[4] C. Perez, S. Crompton, K. Figiela, G. Horn, F. Griesinger, D. Hoppe, T. Kirkham, K. Kritikos, M. Malawski, N. Parlavantzas, L. Pouilloux, D. Romero, C. Sheridan, P. Silva, and A. Sinha, “D3.1.2 – Product Upperware ,” paasage project deliverable, PaaSage project consortium, October 2015.

[5] D. Hoppe, K. Kritikos, C. Sheridan, E. Yaqub, J. Domaschka, D. Baur, F. Griesinger, D. Seybold, B. Balis, D. Król, M. Malawski, and A. Zarioh, “D5.1.2 – Product Executionware,” paasage project deliverable, PaaSage project consortium, September 2016.

[6] T. Kirkham, K. Kritikos, B. Kryza, K. Magoutis, P. Massonet, C. Papoulas, M. Korozi, A. Leonidis, S. Ntoa, C. Sheridan, A. Innes, and D. A. Imrie, “D4.1.2 – Product Database and Social Network System,” paasage project deliverable, PaaSage project consortium, September 2015.

[7] K. Kritikos, K. Magoutis, and D. Plexousakis, “Towards Knowledge-Based Assisted IaaS Selection,” in *CloudCom*, (Luxembourg), IEEE, 2016.

[8] K. Kritikos and D. Plexousakis, “Multi-cloud Application Design through Cloud Service Composition,” in *CLOUD*, (New York, NY, USA), pp. 686–693, IEEE, 2015.

[9] Patrick S. Renz, *Project Governance - Implementing Corporate Governance and Business Ethics in Non-profit Organizations*. Contributions to Economics, Physica Verlag Heidelberg, 2007.

[10] 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, 2008.

[11] 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, 2007.

[12] Diomidis Spinellis, *Code Quality: The Open Source Perspective*. Effective Software Development, Addison Wesley, 2006.

[13] Bonthu Kotaiah , Dr. R.A. Khan, “A survey on software reliability assessment by using different machine learning techniques,” *International Journal of Scientific & Engineering Research*, vol. 3, no. 6, pp. 1–7, 2012.

[14] E. Petrinja, R. Nambakam, and A. Sillitti, “Introducing the opensource maturity model,” in *Emerging Trends in Free/Libre/Open Source Software Research and Development, 2009. FLOSS ’09. ICSE Workshop on*, pp. 37–41, May 2009.

[15] M. Soto and M. Ciolkowski, “The qualoss open source assessment model measuring the performance of open source communities,” in *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, pp. 498–501, Oct 2009.

[16] E. Petrinja, A. Sillitti, and G. Succi, "Comparing OpenBRR, QSOS, and OMM assessment models," in *Open Source Software: New Horizons* (P. Ågerfalk, C. Boldyreff, J. González-Barahona, G. R. Madey, and J. Noll, eds.), vol. 319 of *IFIP Advances in Information and Communication Technology*, pp. 224–238, Springer Berlin Heidelberg, 2010.

[17] A. Adewumi, S. Misra, and N. Omoregbe, "A review of models for evaluating quality in open source software," *IERI Procedia*, vol. 4, no. 0, pp. 88 – 92, 2013. 2013 International Conference on Electronic Engineering and Computer Science (EECS 2013).

[18] E. Charlier and S. Mouton, "D6.2.2 – Final integration, user scripts and end-to-end tests ," paasage project deliverable, PaaSage project consortium, June 2016.

[19] E. Charlier and S. Mouton, "D8.3.1 – Exploitable Prototype System Product," paasage project deliverable, PaaSage project consortium, September 2016.