



PaaSage

Model Based Cloud Platform Upperware

Deliverable D1.6.1

Initial Architecture Design

Version: 0

D1.6.1

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

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

Project website address: <http://www.paasage.eu>

Project	
Grant Agreement number	317715
Project acronym:	PaaSage
Project title:	Model Based Cloud Platform Upperware
Funding Scheme:	Integrated Project
Date of latest version of Annex I against which the assessment will be made:	29 th August 2012
Document	
Period covered:	M1-M12
Deliverable number:	D1.6.1
Deliverable title	Initial Architecture Design
Contractual Date of Delivery:	30 September 2013 (M12)
Actual Date of Delivery:	11 th November 2013
Editor (s):	Tom Kirkham, Keith Jeffery
Author (s):	Tom Kirkham, Keith Jeffery
Reviewer (s):	Pierre Guisset, Philippe Massonet
Participant(s):	Keith Jeffery, Geir Horn, Lutz Schubert, Philippe Massonet, Kostas Magoutis, Brian Matthews, Tom Kirkham, Christian Perez, Alessandro Rossini,
Work package no.:	1
Work package title:	Technical Foundation
Work package leader:	STFC
Distribution:	Project team; EC PO
Version/Revision:	1.0
Draft/Final:	Final
Total number of pages (including cover):	84

DISCLAIMER

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

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

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

The European Union is established in accordance with the Treaty on European Union (Maastricht). There are currently 28 Member States of the Union. It is based on the European Communities and the member states cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice and the Court of Auditors. (<http://europa.eu>)



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

CONTENTS

Contents

Contents	4
Table of Figures	7
Executive summary	8
I. Introduction	10
A. Intended Audience	10
B. Document Structure	10
C. Main Actors	10
D. PaaSage Architecture Overview	12
E. PaaSage's model-based methodology	14
II. Cloud Problemscope	16
A. Rationale	16
1. Current State of the Art Capabilities	16
2. PaaSage Beyond the State of the Art	18
B. Requirements	19
1. Functional Requirements	20
2. Major Non-Functional Requirements	20
3. Summary	21
III. Paasage Lifecycle & Storyboard	22
A. Overviews	22
1. Lifecycle Overview	22
2. Storyboard Overview	22
B. IDE	23
1. IDE Design Storyboard	23
2. IDE Functionality	24
C. Configuration	25
1. PaaSage Configuration Storyboard	25
2. Configuration Functionality	25
D. Deployment	27
1. PaaSage Deployment Phase	27
2. Deployment Storyboard	27
E. Execution	30
1. Execution Storyboard	31
2. Execution Functionality	31

IV.	Component descriptions.....	36
A.	IDE	36
1.	Cloud Modelling.....	36
B.	Profiler.....	38
1.	CP Generator	38
1.	Rule Processor	39
C.	Reasoner	39
1.	Solvers	40
2.	Meta solver	40
3.	CP Solvers	40
4.	Learning Automata (LA) based allocator.....	41
5.	Utility Function Generator.....	42
6.	Solution Evaluator	42
7.	Simulator Wrapper	42
8.	Constraint Logic Programming	43
9.	Solver to Deployer.....	44
D.	Adapter	44
1.	Plan Generator	45
2.	Adaptation Manager	45
3.	Application Controller	45
E.	Metadata Database	45
1.	Metadata database layer.....	45
2.	Analytics Layer.....	50
3.	Social network infrastructure.....	51
4.	Trust and Identity Management.....	53
F.	Executionware.....	54
1.	Component Instance	55
2.	Component Wrapper/Message Interceptor.....	55
3.	Deployer	56
4.	Enforcement Engine	56
5.	Monitor(s)	57
6.	Interpreter	58
V.	Open Issues and Future Work.....	60
VI.	Conclusion	61
VII.	ANNEX 1 REQUIREMENTS TABLE.....	62
VIII.	ANNEX 2 GLOSSARY OF TERMS	69

A.	Cloud Related Concepts	69
B.	PaaSage Concepts	74
IX.	ANNEX3: The CLOUD OFFERINGS SURVEYED	77
X.	ANNEX3: The CLOUD OFFERINGS SURVEYED	78
XI.	Bibliography	81

TABLE OF FIGURES

Figure 1 PaaSage Actor Interaction	11
Figure 2 Main PaaSage Architectural Stack	13
Figure 3 Application lifecycle overview.....	14
Figure 4 Main PaaSage Components and Life Cycle Direction	22
Figure 5 Storyboard Design Phase.....	23
Figure 6 Storyboard Configuration Phase.....	25
Figure 7 Storyboard Deployment Phase	28
Figure 8 Storyboard during the Execution Phase	31
Figure 9 The global adaptation loop	34
Figure 10 CloudML	37
Figure 11 Architecture of the Profiler.....	38
Figure 12: Components that make up the Reasoner	40
Figure 13 Architecture of the Adaptor	44
Figure 19: Metadata database architecture	46
Figure 20: Metadata database schema	47
Figure 21: Integration of PaaSage metadata databases.....	50
Figure 22: PaaSage knowledge base and reasoning engine.....	51
Figure 23: The architecture of the Social Network infrastructure	52
Figure 24 Identity Management in PaaSage	53
Figure 25 Initial Architecture of the Executionware and its interfaces	55
Figure 26: Multi-Cloud monitoring and adaptation of Service-based Applications.....	57

EXECUTIVE SUMMARY

This document outlines the initial PaaSage architecture at the end of the first year of the project. The deliverable describes the key components that make up the PaaSage platform. It also includes an overview of the project requirements and use cases, using a Storyboard approach to link the use case behaviour with the architectural components. Identification of and expected advancement from the state of the art in the field is also part of this document. Finally we outline steps for year 2.

The architecture will deliver and support the following novel features in PaaSage:

- Advanced modelling language for Clouds, utilising models to characterise users, applications, data and platforms as the common thread through the PaaSage environment;
- Live Cloud model adaptation to ensure application execution in-line with service level agreements (SLA) and key performance indicators (KPI) criteria;
- Model-based support for the porting of legacy applications into the Cloud;
- Cross Platform application execution utilising enhanced PaaSage models;
- Optimised Cross-Cloud model based deployment of applications.
- Support for the development of complex deployments and the migration of local systems to Clouds in a model-based standardised way.

Support for PaaSage features requires the following architectural measures:

- Participation in the design and standardisation of an open, powerful, and expressive modelling language together with the MODAClouds project for Cloud-independent modelling of enterprise systems;
- Provision of an intelligent Integrated Development Environment (IDE) supporting the modelling language and supporting the developer in the task of optimising the application using knowledge from experts and monitoring;
- Creation of mappers and generators that allow a Cloud application modelled with PaaSage to be deployed in a distributed environment, interacting with multiple Cloud providers as required for Cross-Cloud deployments;
- Definition and implementation of metadata relevant for Cloud deployment of applications, alongside mechanisms to acquire the metadata and critical performance indicators from various classes of users, from running applications and to reuse the historical metadata available on the services in the application design and deployment.

The deliverable represents the first steps in the PaaSage architecture toward a blueprint of a platform that will be developed so that it can be used in the following ways:

1. Within a commercial or non-commercial organisation to improve the way applications utilise internal and external Cloud platforms;
2. Within an open systems development community to improve knowledge of how various applications perform on various (combinations of) Cloud platforms;
3. As an individual development environment for individual application developers who develop (for sale or other use) applications that need to be deployable across differing Cloud platforms;

And by the following classes of users:

1. Organisational or government policymakers;
2. Organisational chief executives;
3. Organisational IT directors;

4. Systems administrators (including database administrators);
5. Application developers or modifiers;
6. Business application owners;

This deliverable charts our progress at the end of the first year of the project. The architectural designs and component descriptions are the basis on which the initial software implementation will be built during year 2 of the project. It is expected during this process new requirements and functionality will emerge that will go into an internal updated version of this document at month 24 of the project. The second contracted deliverable (D1.6.2: Final Architecture) is due at Month 48.

I. INTRODUCTION

A. *Intended Audience*

The deliverable is a public document designed for readers with some Cloud computing experience but little knowledge of PaaSage. For the external reader the document aims to set out the key elements of the PaaSage high level architecture design and the motivations behind it. For the more technical reader the integration and more detailed description of the platform architectural components are contained in the document.

Finally for internal project partners, other than to crystallise thinking, the deliverable - to quote from the DoW- will also “be used to stimulate discussion among the partners and the advisory board to ensure we have the best design possible within the timescale and resources”.

B. *Document Structure*

This deliverable has the following structure. First, we introduce the PaaSage problemscope summarising PaaSage against current state of the art work within the domain of Cloud computing. Effort is made in this section to highlight preferred technology which PaaSage will either adopt or build upon and planned work beyond the state of the art will be outlined. The requirements section (drawn largely from D6.1.1 produced at M6) is part of the problemscope and documents a summary of the main PaaSage business and technical requirements. The requirements are largely gathered from the use cases and are split into a functional and non-functional classification. The requirements feed directly into the following section which outlines in detail the PaaSage lifecycle.

The lifecycle section gives a high level view on the PaaSage platform and how it is applied during a typical Cloud application lifecycle. Specific focus is given to describing how the PaaSage platform is applied in the use cases expressed as storyboards to aid reader understanding.

The final main block of content is a more detailed analysis of the PaaSage platform drilling into the component level of the architecture. This section explains each component and how they interact with each other. At this stage the design is in its early phase as the project is following a bottom up approach. The Future Work / Open Issues section follows the component descriptions highlighting work in year 2 of the project. Finally the deliverable ends with a Conclusion.

C. *Main Actors*

The main actors in PaaSage (in addition to the business application end-user and associated organisational management actors) can be split into between the application designer/developer, and Cloud Provider with PaaSage sitting in-between. The interaction of parties depends on the PaaSage platform deployment and application scenario. It is conceivable that during some application deployments PaaSage will interact with multiple Cloud Providers and vice versa as illustrated in Figure 1.

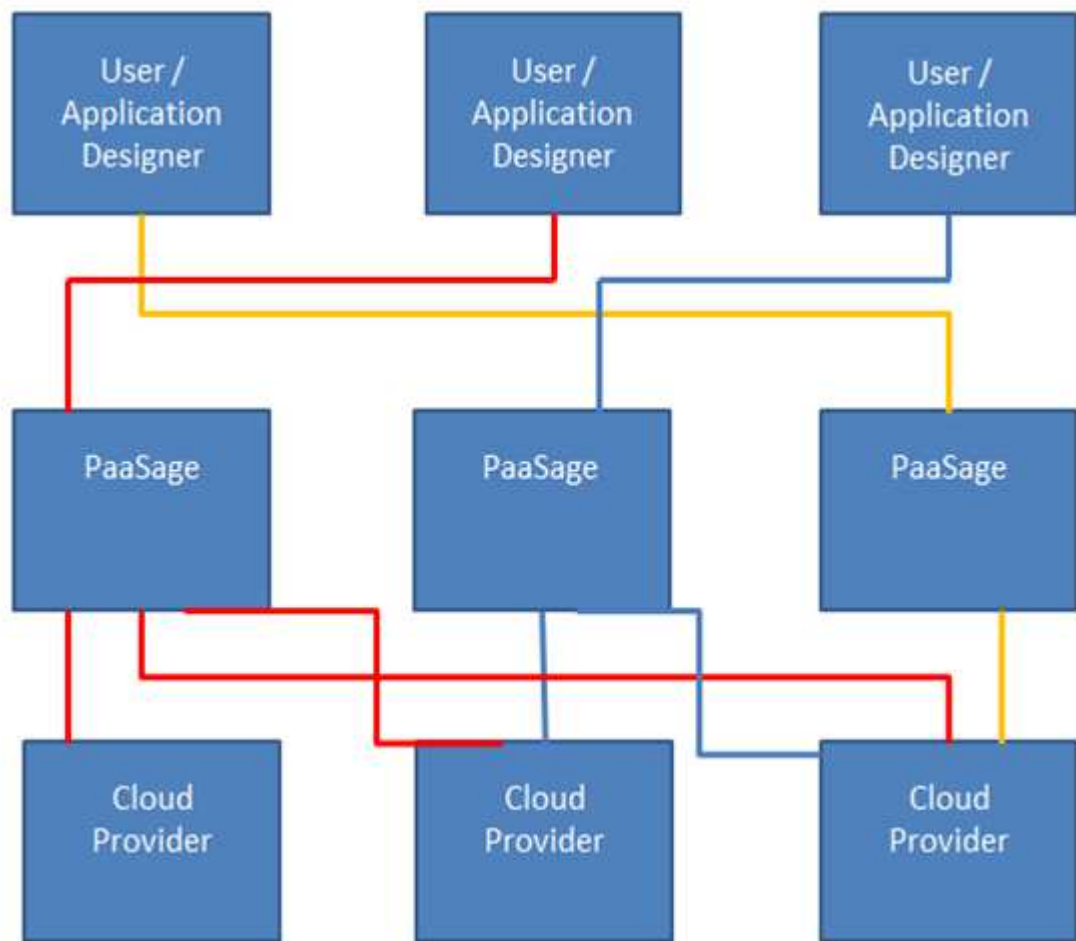


Figure 1 PaaS Actor Interaction

In this document when “user” is mentioned the document refers to users of PaaS which are application designers working for a toward a personal or corporate business goal via deploying to the Cloud using PaaS. These users engage directly with PaaS in an application development / design role via an interface such as an Integrated Development Environment (IDE). They are distinguished from business application users in the respect that they gain from PaaS’s use in their business activity. Types of business application users in the PaaS use cases include the Lufthansa flight scheduler and government business manager.

The platform acts as a broker between the user and the Cloud and can be deployed within an organisation or in an open community. For user / application designer we aim to realise through development and use of the PaaS platform a design once and deploy to all concept. Thus, users engaged with PaaS can have confidence that the configuration / business goals they model will be supported when their application is deployed across Cloud environments. This significant vision will make the Cloud more transparent, increase confidence in using Clouds and help business better predict / control resource usage / cost when deploying to the Cloud.

Deployments within an organisation such as Lufthansa present the PaaS platform as a shared organisational resource. The main remit of the platform is to aid the deployment of applications in the Cloud around the specific Lufthansa business model. These deployments are likely to yield commercially sensitive data especially the intelligence on Lufthansa history of use and Lufthansa expert knowledge in the metadata database and are therefore subject to organisational policies. Data which is less commercially sensitive can be shared outside of the platform with other PaaS implementations, the Lufthansa application benefits from shared data too.

The other type of deployment is in less commercially sensitive open community. Application scenarios here include ones where high levels of collaboration is needed on projects such as presented by the eScience domain. Here PaaSage is a shared resource, especially the intelligence in the metadata database on open history of use and expert knowledge. As mentioned above instances of organisational deployments will have access to and use the open community deployments.

Another possibility is for PaaSage to be provided by a third party supplier which provides both closed and open facilities. For the PaaSage platform we aim to create a new integrator / broker business model for Clouds. This model will encourage SMEs with domain specific knowledge to support model creation and integration with marketplaces of IPs.

Cloud Providers would not have to present any specialised interfaces to engage with PaaSage. However the platform will support provider specific interfaces for monitoring recording and sharing data on executions. It is expected that the PaaSage project will create new business innovation for all actors identified above. This will be driven by PaaSage technical and market innovation at the platform provider level. Through this innovation and methodologies to increase trust and confidence in Cloud adoption, it is expected that PaaSage will open up new markets to Cloud Providers. In return we expect the Cloud Providers to increasingly support and feed into the development of PaaSage standards to aid integration and release this business.

D. PaaSage Architecture Overview

At this year 1 stage, in order to document the architecture, this deliverable presents a high level view on the integration and design of the emerging different components / services in the project. Together these elements form the PaaSage Cloud deployment and management platform. More detailed explanation of individual services / components will be found in the more specific work package deliverables D2.1.2, D3.1.1, D4.1.1, D5.1.1 due M18.

The overall PaaSage design is summarised into 3 main component groupings as described in the Description of Work, the Integrated Development Environment (IDE), Upperware and Executionware:

The IDE is the main user-visible front end for the platform. The IDE extends the popular open source development platform Eclipse and supports the chosen Cloud Application Modelling Execution Language (CAMEL) including CloudML [1] [2]. The IDE as a PaaSage layer has the role of ensuring that model-based integration of the various functional components in the project is possible within a variety of application scenarios.

The Upperware is integrated with the IDE and presents via the IDE a collection of tools and components to capture the needs when developing and porting models at design-time. The Reasoner and Adapter make up the main components at this layer in order to support the user of the IDE with PaaSage model-based knowledge and to provide the Executionware at run-time with support from the running application and execution platform.

The Executionware provides platform-specific mapping and technical integration of PaaSage to the Application Programming Interfaces (APIs) of the execution infrastructure of the Cloud provider. This link also provides capabilities for extracting extended information about the behaviour of a variety of platform providers and possible reconfiguration to maintain service level objectives to support application behaviour. The Deployer and Adapter work together to ensure the optimal deployments from the Upperware are both maintained and monitored.

Each one of these main PaaSage elements integrates with the same service and component metadata database. This store contains information about past executions and also performance of different

Cloud providers. It is the main knowledge store in PaaSage and provides knowledge from outside the platform via social networks and other authenticated third party actors.

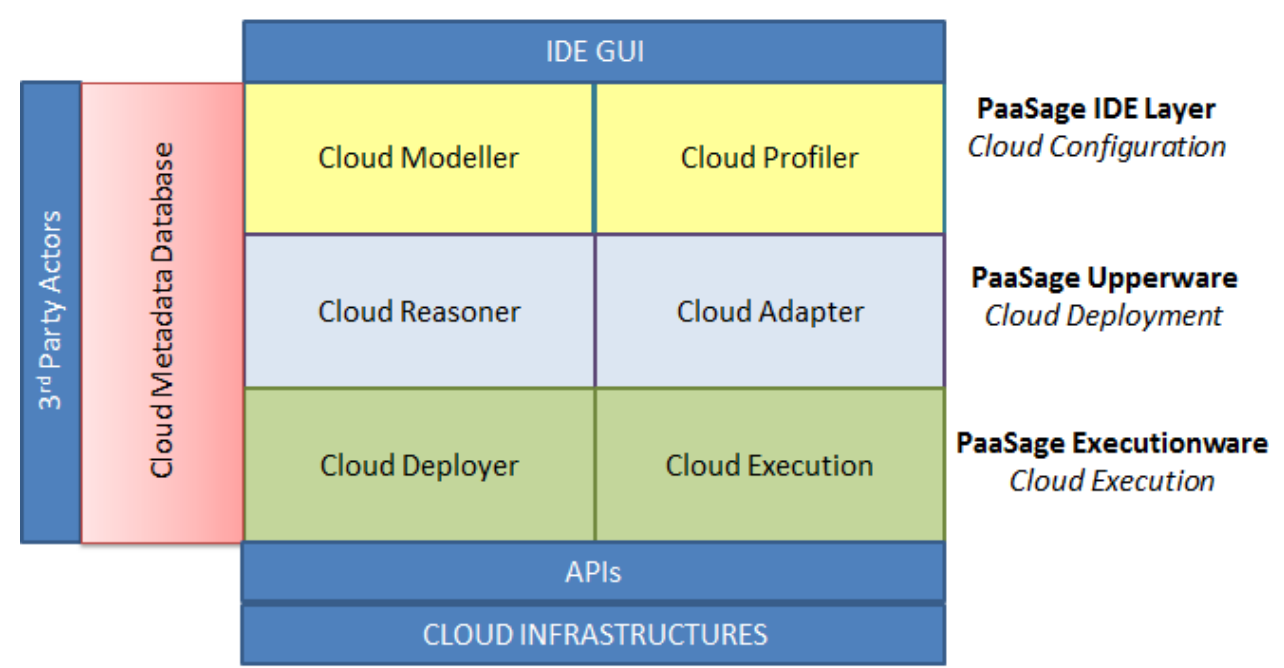


Figure 2 Main PaaSage Architectural Stack

The main PaaSage stack of components is illustrated in

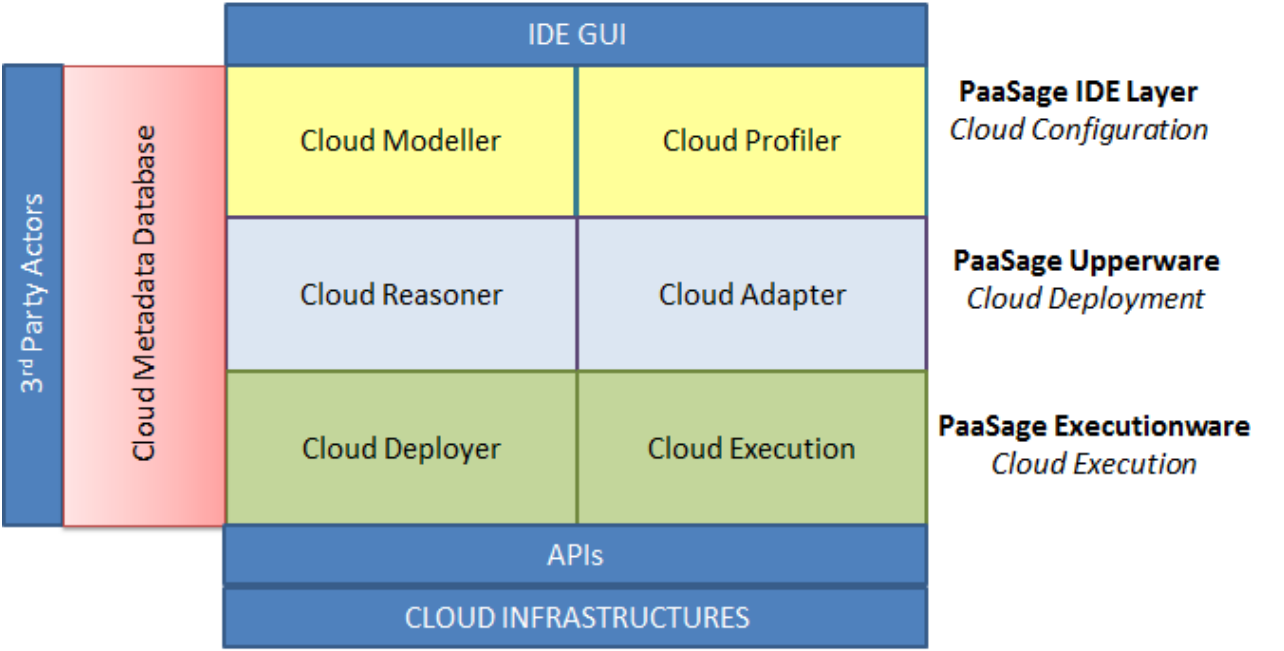


Figure 2, the architecture aims to integrate these components in order to deliver the PaaSage platform’s main functions. However, it could be possible to adapt individual components to deliver

explicit functionality in other specific Cloud environments i.e. functions in the Profiler and Reasoner could be adapted for use with other Cloud optimisation environments.

E. PaaSage's model-based methodology

PaaSage's model-based methodology is based upon the key Cloud lifecycle phases of configuration, deployment and execution. These phases are based on the Waterfall Model of Software Development with the following mappings Configuration phase (Requirements, Design) Deployment phase (Implementation) Execution phase (Verification, Maintenance) [3].

Configuration is concerned with modelling the deployment of applications, profiling platforms and infrastructures, and specifying Quality of Service (QoS) requirements and data management policies. Deployment is concerned with matching the Deployment Models of applications with the profiles of platforms and infrastructures based on negotiated SLAs and policies, and selecting one or more suitable Deployment Models. Execution is concerned with the management of the run-time execution of applications and monitoring / recording of KPIs based on SLAs and policies.

It should be stressed that although a waterfall model is used in the phases through which an application passes in PaaSage, the actual software development in the PaaSage project to provide the PaaSage platform is done using a spiral, agile approach.

In order to facilitate the integration across the components responsible for each lifecycle phase, PaaSage adopts a series of interlinked models.

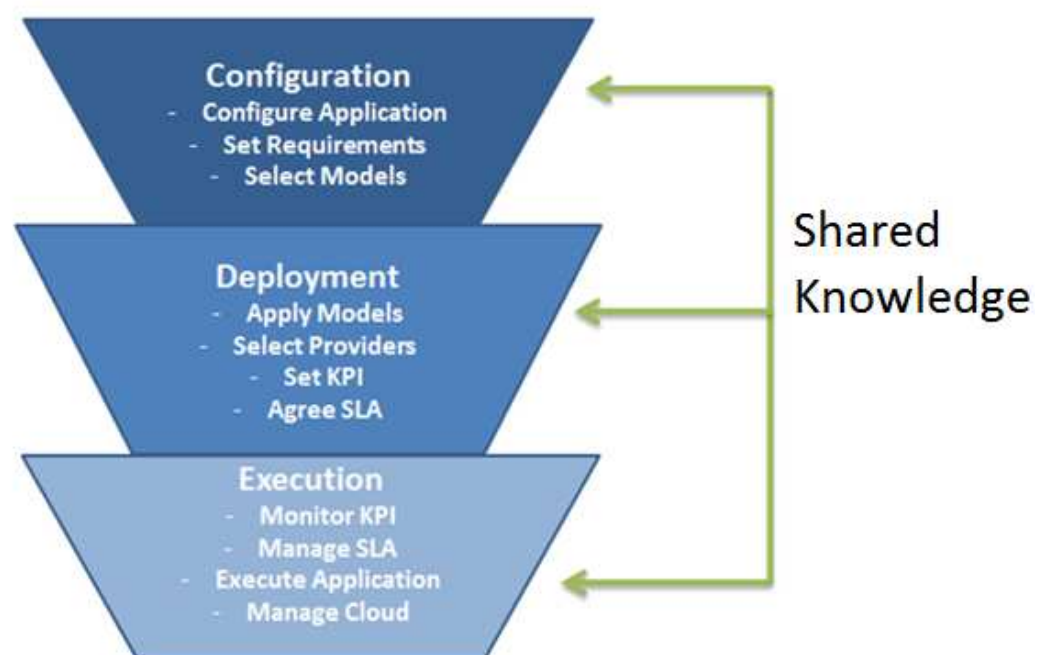


Figure 3 Application lifecycle overview.

Models in PaaSage are initialised as empty templates, populated with characterising and deployment rules that are extracted and replaced with deployment characteristics. Models are initially formed from user input in the IDE and contain platform, data and policy specific information. We envisage three types of model in PaaSage:

Configuration Model: The Profiler populates an empty Configuration Model template with requirements set via the IDE modelling component. The model template choice ensures that an adequate requirement set is gathered for specific scenarios. The initial model could be created using a standard such as UML, but the requirements in the model are translated by the component into terms that the Reasoner can understand.

Deployment Model: The Reasoner consumes the requirements packaged up in the Configuration Model passed from the Profiler. Using these requirements and data from sources such as the Metadata Database the model is transformed into a mapping of deployment characteristics which are a result of the reasoning.

Execution Model: The Adaptation Engine checks the Deployment Model against the current state of the Cloud Providers. The Execution model is populated with the suitable infrastructure and endpoints of services (i.e. monitoring) to support the applications execution.

I. CAMEL and CloudML

The models used to capture the lifecycle are expressed in a set of DSLs (Domain Specific Languages) that together form CAMEL. One of these is CloudML. CloudML is being developed within the MODAClouds [4] project. As part of the work on MODAClouds we expect CloudML to be linked to lower level messaging standards that we intend to implement across the PaaSage platform and between Cloud providers.

The models and data related to their execution in the Cloud are stored in the Metadata Database. This allows reuse of the models and the ability for component such as the Reasoner to look at the performance of previous models when composing new ones. This knowledge is also shareable between PaaSage platforms (subject to security and privacy). Reuse of the PaaSage models outside of the local PaaSage implementation will be a knowledge foundation that related tools and services within the open social network domain will be built upon. The open social network will be the source of the 3rd party actors shown in the stack in Figure 1 and drive the sharing and building of PaaSage knowledge and models.

II. CAMEL

The MODACloud development of CloudML is expected to follow the topological approach of the standard. Further detail in the description of application requirements and deployment characteristics as they relate to the Cloud infrastructure is the focus of this work. However, a key element of the PaaSage project is also the specification of rules and constraints to match the topological description of the Cloud.

In order to support the requirements for a PaaSage Cloud Modelling Language we intend to re-use existing DSLs and – if necessary – develop them further. In addition we shall interface these languages so they pass the required information through the phases of PaaSage. At this stage of the architectural development these candidate DSLs include SALOON (to characterise the offered CLOUD platforms), WS-Agreement (to manage SLAs), OWL (to characterise components) and a rule-based language (currently the subject of on-going assessment) to manage constraints.

II. CLOUD PROBLEMSCOPE

A. *Rationale*

Moving to the Cloud is difficult; generally little or no expertise exists in the form of tools and platforms to help the developer restructure his/her application toward the Cloud. Users from the business community struggle to visualise the implications in terms of measurable threats and benefits from application movement to the Cloud.

Thus, there has been only a slow take-up of Cloud technology for real business applications, although Clouds have been used for shared email environments, shared storage systems and similar purposes. Certainly many organisations have experimented using Cloud platforms (private or public) for systems development and one-off applications but major barriers exist in terms of the inability for applications to reconfigure dynamically across private and Public Clouds and maintain pre-Cloud SLA/QoS parameters.

The PaaSage project aims to address these problems using a model-centric approach. By developing a model-based Cloud management platform, PaaSage aims to deliver a platform-supported approach that provides greater flexibility and support for user / business requirements when managing applications across the whole Cloud lifecycle and deployment architectures.

1. Current State of the Art Capabilities

1. Other Research Projects

A systematic survey (using a standard agreed template) of existing CLOUD offerings was conducted in the first six months of the project in parallel with the work in WP6 on requirements leading to D6.1.1. While some offerings required just a cursory consideration, others were considered in detail. The list of considered EU offerings is in ANNEX III (Section IX) and key aspects illustrating the state of the art using a selection of these are summarised below.

Research in providing Infrastructure as a Service is a focus of several projects. Particular focus of work in this domain is in the support of innovation in infrastructure provision and monitoring toward greater resource use in the Cloud. A good example of such an approach can be seen in the OPIMIS project [5]. The OPTIMIS Toolkit comprises a set of tools to be used by Service Providers (SPs), Infrastructure Providers (IPs), Software Developers (SDs), and end users.

In terms of platform effort has been made in developing PaaS provision using more standardised approaches. A good example here can be seen in the effort to merge Service Oriented Architectures with Clouds. The Cloud4SOA [6] project is focused on integrating SOA principles of modularity and web services with the provision of PAAS. Other innovations of provision of platform are in the development of federated PaaS in projects such as Contrail [7].

Service development in Clouds as a focus of work can be seen in the data management community. Projects in this domain have tended to focus on the improved presentation and categorisation of data in Clouds to aid integration with Cloud services. A good example of such work can be seen in the cloudTM project [8]. Here the project is focused on creating a data centric middleware in order to aid better identification of data and its requirements to aid better efficiency and fault tolerance in the Cloud.

Linked to the PaaSage project is effort from the MODAClouds project in development of the CloudML standard. In addition other modelling projects are focusing on the use of models to support specific challenges such as the migration of legacy systems to the Cloud. In the Artist project models are used to describe and wrap legacy systems to aid migration [9]. Other projects are looking to existing

standards to aid the model based management of Clouds, such as the Mosaic Cloud project that has embraced ontologies as central to their modelling solution [10].

II. Commercial Offerings

Microsoft's Windows Azure [11] offers not only PaaS but also services for IaaS (e.g. VMs, virtual network, storage), and SaaS (e.g. media, active directory and web hosting). Each service can be used separately or combined to create an application. Users can manage the instantiation of a service through a simple configuration in a web portal, directly in a development environment (e.g. Visual Studio or Eclipse), REST API, and/or a command line tool. In terms of SLA, Windows Azure provides a guarantee of at least 99.9% availability of the time on their services [12]. Finally, Windows Azure provides several tools for collecting, monitoring and diagnostic information, such as:

<i>Management Portal</i>	<i>Displays the status of a hosted service, including each instance of the service.</i>
<i>Service Management REST API</i>	<i>Provides an API that can be used to programmatically retrieve the same status information displayed in the Management Portal.</i>
<i>Diagnostics</i>	<i>Provides the user with the ability to aggregate performance counters and logs from the hosted application instances, as well as any custom log files, tracing, and instrumentation output that the application produces. Also, it provides a mechanism for scheduling periodic copies of diagnostic information to the storage account.</i>
<i>Storage Analytics</i>	<i>Provides logging and metrics data for Windows Azure Storage.</i>
<i>SQL Database Dynamic Management Views</i>	<i>Provides information useful in diagnosing performance problems when using SQL Database.</i>

Google App Engine (GAE) [13] is a PaaS for developing and hosting web applications in Google-managed data centres. Thus, users need only to upload their applications without the need for maintaining any servers. GAE supports applications that run in one of several run-time environments, such as the Go environment, the Java environment, the PHP environment, and the Python environment. An application may be running in one or more GAE instances. The GAE instances are not real VMs but application sandboxes. They are similar to VMs, where both have a set amount of RAM allocated to them. However, GAE instances don't have the overhead of running operating systems and/or other applications. Thus the GAE instances have more usable memory than the VMs. Moreover, each GAE instance includes a security layer to ensure that instances cannot inadvertently affect each other. GAE also guarantees a SLA of at least 99.95% of the time in any calendar month [14]. With regards to monitoring of instances, the GAE Dashboard in the Admin Console has six graphs that provide users with a quick visual reference of system usage. The information displayed in these graphs gives the user a snapshot of resource consumption per second over a period of up to 30 days.

CloudBees [15] is a PaaS specialized in Java applications. The developers have the possibility to implement their applications with any JVM-based language, such as Java, Scala, and JRuby and to use a variety of run-times, such as JBoss, Tomcat, and the Play Framework. The PaaS enables the creation and removal of applications, databases and users. The applications can also be started, stopped and replicated. CloudBees exposes a REST API enabling the execution of these actions. The

monitoring of applications is done through the New Relic Monitoring service [New Relic, Inc], a performance management tool.

Cloud Foundry [16] is an open-source PaaS Cloud software as well as a hosted service offered by VMware. Many other companies offer PaaS services using the Cloud Foundry platform (e.g., AppFog and ActiveState). The PaaS supports multiple programming languages such as: Ruby, Python, PHP, NodeJS, Erlang and JVM-based languages like Groovy and Java. It also supports multiple run-times and frameworks (e.g., Spring, Rails and Sinatra) and application services (e.g., MySQL, MongoDB and RabbitMQ). Applications, users and databases can be added or removed. Applications can also be started, stopped, updated and replicated. Such operations are supported via a REST API, which also enables the retrieval of statistics related to uptime, disk use, CPU and memory usage. Additional information related to java applications can be retrieved by using Spring Insight [Spring], a byte-code instrumentation-monitoring tool.

Heroku [17], a Cloud application platform, supports JVM-based languages such as Java, Scala, Clojure and other programming languages as: Python, Ruby and Node.js. The REST API provided by the platform enables developers to create, remove and update users, applications and databases. Applications can also be started and stopped. Processes related to applications can be replicated for scaling purposes. The New Relic Monitoring service is used to monitor resources such as CPU, memory, network and processes.

Jelastic [18] is Cloud PaaS solution, which runs any Java or PHP application on the Cloud. Users select a software stack that includes application servers (e.g., Tomcat, GlassFish, Jetty) and SQL or NoSQL databases (e.g., MariaDB, PostgreSQL, MySQL, MongoDB, CouchDB). The platform provides an intuitive GUI enabling the creation of applications and databases. The GUI also provides functionality to start and stop applications, configure the load balancer and modify the number of application servers. It is possible to retrieve statistics about CPU, memory, disk, and network utilisation for load balancer, web server and database instances.

2. PaaSage Beyond the State of the Art

PaaS applications today have approached the task of application support through the creation of interfaces capable of supporting multiple programming languages backed up with management GUIs. This approach is clear in the current state of the art and in the approaches from Jelastic, Heroku, Cloud Foundry, Cloud Bees and even within Google and Azure described above. This approach assumes that the application developer user has a good knowledge of his / her application and how it should work / consume resources in the Cloud. In reality this is not necessarily the case as many end users particularly business wishing to port applications to the Cloud have little knowledge of what Cloud resources they require to support their applications.

PaaSage intends to take a step back from providing a purely technical interface to the PaaSage application developer user and encourage the user to model his / her requirements before technical integration takes place. User defined application models expressed in CAMEL will allow a richer expression of application developer user and business application end-user requirements translating down to how the Cloud is managed in terms of resource usage. Advancing research in MODAClouds and projects such as Mosaic [19] inform work in PaaSage which will look to group domain specific standards in models and deploy them at all stages of application use in the Cloud. The models are used from configuration through to execution in the Cloud; they will ensure that the user is presented with a consistent model of application activity based on his/her original requirements during design and deployment.

The management of the model driven application in the Cloud is novel and unique as it breaks from existing work in the automated management of Cloud applications. Common approaches via the use of an application developer user focused GUI linked to execution and policy are improved upon in

PaaSage to provide a common link in the GUI to the deployed model provided by the user. This model-driven approach will give a more holistic view of the application deployment. For example, a model-driven view can express more detail than a standalone policy or set of rules as models can contain information on the relationship between different monitored elements and incorporate rules.

The PaaSage core components are designed to make knowledge-based Profiling, Reasoning and Adaptation decisions on the deployment that will improve the performance of the Application deployed in the Cloud in step with user requirements. This work will build on techniques developed in projects such as cloudTM for data classification and. PaaSage will present knowledge-based live management of services and data in a holistic way. This holistic view will improve on project work in this area such as done by OPTIMIS where management decisions are made in isolation from the deployment. For example, in OPTIMIS the deployment optimisation is done in isolation from run-time optimisation. A common resulting problem in such systems is in the transfer of resources and subsequent time taken to transfer images during Cloud transformations [20].

In PaaSage we aim to reduce this by linking deployment and operation via the model-based approach. This will allow knowledge based decisions around the deployment in terms of its potential impact on operation of the Cloud. Such knowledge can influence the deployment to enhance execution via factors such as reducing the need for the application on the Cloud to transform (i.e. Cloud Burst). In the case where a transformation occurs the platform plans to reduce resource consumption through the knowledge-based deployment to nodes closer to typical transformation targets. For example, PaaSage will have knowledge of previous deployments of application types and the track record on their resource consumption and activity. By using this data and the knowledge of typical transformation that these applications require PaaSage can make optimal deployments to ensure faster transfer time of images via simple means such as network placement.

In summary, PaaSage provides model-based application support to the Cloud. The platform provides an intuitive way of adapting user requirements when managing applications in the Cloud. The model-driven approach enables a finer grained description of the deployment constraints allowing the platform greater flexibility to manage automatically the application during changes in the Cloud Infrastructure.

B. Requirements

D6.1.1 provides a description of 3 major use cases and the associated user requirements. The case studies described the application requirements and identified high level requirements on the PaaSage workflow. In addition a generalisation of the major user requirements was done (internal documentation) to provide the core requirements relevant to a wide range of user / application requirements. In D6.1.1 an initial assignment of requirements to PaaSage components was made. These are classified into functional (related to the business of the application) and non-functional (how the application is run) requirements. Further categorization and prioritization of these use cases (e.g., using the MoSCoW method¹ will be performed as part of an iterative process that includes further concretization and development of the PaaSage platform.

The table in Annex 1 provides an overview over the main requirements as identified in the context of D6.1.1 and considered in the following sections. Within the Annex the requirements are numbered R1, R2 etc.

¹ http://en.wikipedia.org/wiki/MoSCoW_Method

1. Functional Requirements

The main requirements identified in D6.1.1 and listed in ANNEX 1 specify that the PaaSage platform must help to optimise Cross-Cloud deployments with respect to deployment objectives and constraints. To achieve optimal deployment PaaSage must support best placement on the available Cloud Infrastructure of the application to be deployed (R1-7). Using models related to the application in PaaSage and its requirements, relevant data is gathered using the PaaSage platforms links to Cloud Infrastructure in terms of knowledge of previous activity and live monitoring. This functionality is needed to enable the setup within Cloud and Cross-Cloud deployments.

In terms of the application it is assumed that the Configuration Model provided by the analyst may not be complete, and that the PaaSage analysis will complete it to provide a Deployment Model. The final Deployment Model will be transformed into an Execution model that must describe the application as well as the target Cloud infrastructure. Deployment must be defined sufficiently to optimise the Cross-Cloud deployment. This includes defining the deployment units (R8-15) and the communication channels (R-16,17,18,19). It also includes defining the required dependability (R20-23, 25) and scalability (R-26-31) of the application. This is discussed in the next section on non-functional requirements.

Data is largely covered by the Deployment Model that must also provide information on how data may be optimally deployed in the Cloud. This includes describing the data volume (R-32) involved, the different ways that the data may be partitioned, the data flows and workflows involved (R-33).

Infrastructure requirements are also contained in the Configuration Model (R34-41). This will enable the Reasoner to match suitable Cloud providers automatically. The required infrastructure should specify the type of Cloud that is required (IaaS, PaaS or SaaS) as well as the Deployment Model (public, private, hybrid Cloud) and monitoring information available.

The PaaSage platform aims to help human analysts make optimised Cross-Cloud deployments (R 42-55). To this end the platform will assist the human analyst in finding near optimal Cross-Cloud deployments (R43). For example, PaaSage will help to find optimised Cross-Cloud deployments that make trade-offs between cost and performance goals. This requires having knowledge about each Cloud provider's typical performance and cost metrics in different scenarios (R42), possibly following their reputation (R45) and understanding their cost models (R55).

The PaaSage platform should support the entire lifecycle of the deployment (R 56-69). This means that PaaSage must provide suggestions for the initial Cross-Cloud deployment, monitor the execution of the deployed application by suggesting adaptations (R64) and reconfiguring (R66). When the application deployment finishes the Cloud lifecycle is finished and a report should be produced (R69).

2. Non-Functional Requirements

The case studies identified various dependability requirements in D6.1.1. This mainly included requirements on availability, security, integrity and privacy.

Availability (R-24, 25) is one of the main criteria for comparing and selecting Cloud providers. Market analysts often refer to uptime/downtime (time that a running

computer is available) to compare high availability Cloud providers. The PaaSage platform will consider availability of Cloud providers as a key selection criterion. The Configuration Model may include requirements on the global availability of the application that is deployed on multiple Clouds. Selection of the right Cloud providers to meet the application criteria will take into account guaranteed availability data from providers and/or historical availability data gathered by the PaaSage platform. PaaSage will also consider composition of availability data for the global Cross-Cloud deployment.

The case studies have identified end to end security (R21) as an important requirement for their Cross-Cloud deployments. Furthermore end to end security must be maintained throughout the whole lifecycle of the Cross-Cloud deployment. Related security requirements include ensuring Cross-Cloud access control (R20) as well as end to end data integrity (R22-23). Cloud providers should also support privacy requirements and comply particularly with the 95/46/EC European data protection directive [21].

Cross-Cloud deployments that need to scale in order to benefit from the on demand nature of the Cloud should be able to specify application elasticity rules (R26-29). The case studies have identified several requirements related to scalability.

Performance is a key requirement that needs to be taken into account by the PaaSage platform when proposing alternative deployment scenarios (R7, 14, 60, 66, 69). It must be possible to refer to performance as an optimisation goal for the Cross-Cloud deployment. Data locality is also important when location of processing nodes to data sources can be optimised to improve performance and other factors such as security and privacy.

Cost is one of the key drivers for the adoption of Cloud computing. Several case studies have identified the need to be able to estimate the cost of a deployment. This requires having knowledge of the cost models from the different Cloud providers that will be supported by the PaaSage platform. It must be possible to refer to cost as an optimisation goal of the Cross-Cloud deployment. It must also be possible to find Cross-Cloud deployments that meet cost/performance objectives.

3. Summary

The PaaSage requirements at this year 1 phase have determined the architecture design. In the next 12 months they will be adapted to specify software specific characteristics. The realisation of the requirements can be seen in terms of function in the three main components described later on in this document (Profiler, Reasoner and Adapter). The Non Functional requirements have also influenced design in other components such as the Metadata Database and its initial security design.

III. PAASAGE LIFECYCLE & STORYBOARD

A. *Overviews*

1. Lifecycle Overview

This chapter aims to give a lifecycle summary of the architectural development in PaaSage. The approach taken is from the perspective of stakeholders at phases before engagement with PaaSage, during PaaSage deployment, execution and reconfiguration. This approach is further reflected with the inclusion of relevant storyboards for each use case. A summary of the main components with respect to the lifecycle direction can be seen in Figure 3.

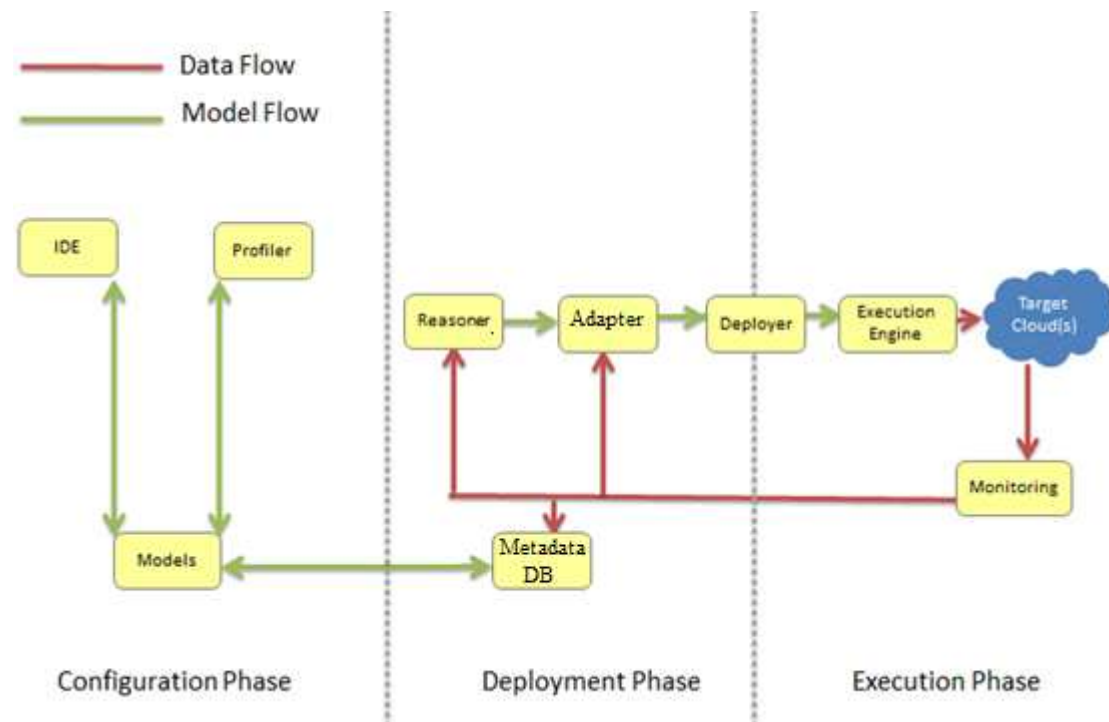


Figure 4 Main PaaSage Components and Life Cycle Direction

The lifecycle is broken down into three main phases. These are configuration, deployment and execution. The configuration phase is concerned largely with characterising as models the application, user, data and available Cloud infrastructure(s). This configuration is used in the deployment phase to select (Reasoner) target infrastructure(s) that satisfy criteria in the Configuration Models. Finally during the execution phase the Deployment Model is executed and can be rolled back (adapter) in case of redeployments due to execution errors or changes in infrastructure.

2. Storyboard Overview

The PaaSage work plan defines four main use cases. These use cases form the basis for our storyboards that are presented to help explain the main lifecycle phases supported by the PaaSage architecture.

eScience use case: Is concerned with the support of complex and large scale workflow based cloud (high performance) computing applications. PaaSage is expected to aid the application design and deployment process to the Cloud.

ERP (Enterprise Resource Planning) use case: The ERP use case is concerned with the delivery of the Cloud on multiple Client devices and the separation of local / remote processing in order to optimise the application. The application is expected to be highly mobile allow technicians to work when they are not connected to the internet.

eGovernment use case: The eGovernment use case presents the problem of how a hybrid Cloud can be both managed and constructed in PaaSage. The requirements from the use case include strict data processing rules alongside the ability for the Cloud to transform to meet demand connecting local services to processes running in the Cloud.

Airline Scheduling use case: At the heart of the airline scheduling use case is the problem of to transform a client-server application with a centralised database and fat client UI, into a cloud application that also supports mobile computing and multiple devices. In this scenario rapid saleability is needed while maintaining integrity of both the application and data. The use case is focused in the airline industry in the case when an incident occurs and planes / passengers have to be rapidly re-routed.

B. IDE

1. IDE Design Storyboard

In terms of the storyboard the configuration phase is pre-dated by a pre-PaaSage engagement design phase that starts with the individual users in our four use cases of eScience, ERP, eGovernment and Airline Scheduling. As Figure 4 illustrates our users have different demands.

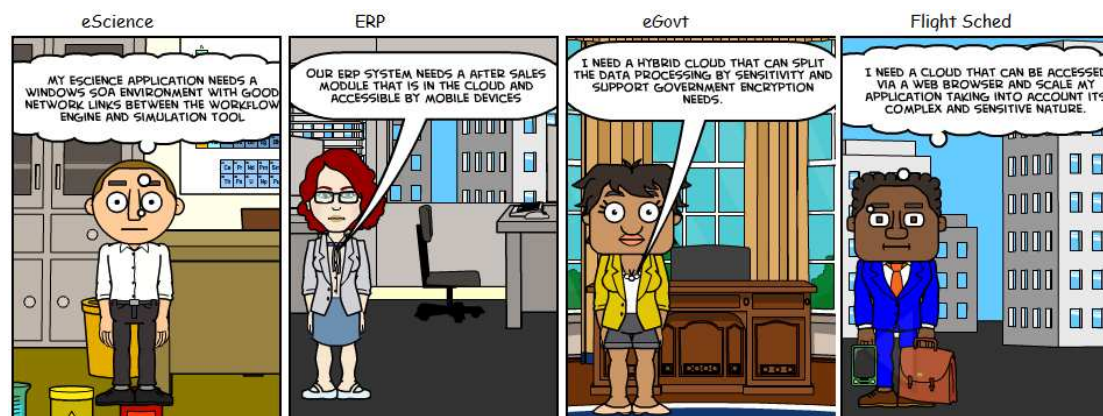


Figure 5 Storyboard Design Phase

The eScience user during application modelling sets requirements related to the platform that the application will run on and the levels of quality of service (QoS) needed to support successful execution.

The ERP use case at this phase could contain specific deployment characteristics that are reflected in the business process and policies of the organisation. For example the platform is important but also support for mobile devices.

The Public Sector Citizen Portal design could specify a hybrid Cloud model where services in Private Clouds can communicate with services in Public Clouds. Sensitive data will have to be stored in Private Clouds and Authentication plus digital signature services are used to secure the application and guarantee end-to-end security. The application must also be scalable in both public and Private Clouds and be portable between data centres

The Airline Scheduling design also includes the need to distribute data depending on its sensitivity. Of great importance is the ability for the application to scale quickly in order to react to demand.

2. IDE Functionality

I. Specification using PaaSage IDE

Stakeholder: multiple, broken down below

During this phase, all information needed to steer execution is specified. This involves aspects such as (1) the business goals, (2) security policies, (3) company policies & contractual constraints, (4) technical constraints. These requirements are used to start the PaaSage configuration phase in the next section. Depending on the type of company and application, this may also include end user conditions (customisation), though they may also emerge at run-time, leading to reconfiguration.

II. Configuration Phase Using the IDE

The configuration phase is the process by which the main stakeholders in the application specify their application execution requirements with associated user and data characteristics. For example, the constraints leading to the choice of the required service model (IaaS, PaaS, SaaS), the required Deployment Model (private, hybrid, public, partner), and also specify a list of cloud providers, e.g. Amazon, Azure and RackSpace especially if there are organisational policies on this. In parallel the characteristics of Cloud platforms / infrastructures are updated as a model. These requirements are captured either by using supported standards and imported into PaaSage or via the use of PaaSage tools via the IDE

During this phase the user / application designer must describe the application to be deployed. This description must state the optimisation goals and constraints of the deployment. An example of optimisation is to minimise cost and maximise performance while maintaining the data in a Private Cloud. The units of deployment and the communication links of the application to be deployed must be described. It must be possible to describe the elasticity rules that describe for each deployment unit how that unit scales up and down with respect to monitored variables such as response time or queue length. It must be possible to specify constraints on availability, performance, cost, security and privacy of the application. This Configuration Model is then used to transfer the requirements expressed as rules to the deployment phase in the lifecycle.

C. Configuration

1. PaaSage Configuration Storyboard

In the formation of the Configuration Model the main component used is the Profiler. The user takes a back seat and is able to monitor the platform's progress as illustrated in Figure 5.



Figure 6 Storyboard Configuration Phase

During the eScience configuration the dependencies in the workflow are checked by the Profiler to ensure that the application is suited for deployment on the Cloud. In particular focus is given to non-functional characteristics such as performance and security policy which have a strong influence on how the deployment is configured.

ERP configuration is dictated by which dependencies exist between the workflow components upon deployment. In addition to this client side applications capable of processing data off-line will be identified.

The eGovernment use case during configuration is driven by data security and the need to identify and separate potential data for public or private Cloud processing. This is complicated by the data processing rules also affecting location of service deployment either on Public or Private Clouds.

Airline Scheduling is again concerned with data dependencies during configuration. As the key function is to support rapid scalability the data and service dependencies have to be supported in the configuration to enable this.

2. Configuration Functionality

III. Specification of application outside PaaSage IDE

Stakeholder: Developer, mostly

The programmer develops his/her code in a normal fashion, yet basing on modular / service-oriented principles. He/She uses a standard tool (UML, BPEL etc.) to generate the software architecture and generate the code. The developer follows some guidelines using the PaaSage supporting documentation on how to develop applications that can be deployed on multiple Clouds (cross -Cloud deployments).

Once the code specification is complete the software architecture following UML standards can be imported into the PaaSage IDE, with clear linkage between UML and code objects; furthermore there should be a strict classification of software artefacts that will define the execution environment they require (Java applications demand for

a JVM; servlet applications require a Servlet container; more specific Servlets may require a dedicated Servlet container; links to databases may be generic (any SQL-capable database) or very specific (e.g., Oracle 12c).

If the application is a legacy application the process is slightly different. In this case the code for the application may not use common standards or be based on common service orientated / modular principles. In this case the application will be treated as a black box and UML will be used to describe dependencies needed for deployment and execution. Of course CAMEL will be used to describe how the black box may be optimised for Cloud deployment using PaaSage.

IV. Specification of business goals

Stakeholder: Business owner or CIO

The main commercial stakeholder specifies what kind of business goals he wants to pursue with the execution of the application. This will most likely not be on a technical level, but instead include considerations, such as “serving 1000 users without notable delay” and “costing less than 1000€/per day”. PaaSage tools will assist in specifying these constraints via methods such as rules.

Generic Cloud business knowledge may help in generating these rules, along the line of particular guidelines, such as “response times less than 1ms are not feasible”, “you should specify maximum number of users”, and “response time means interaction time with a GUI”. This knowledge could also be supported by the PaaSage Reasoner’s knowledge of previous executions. Such knowledge helps the commercial stakeholder in specifying all information needed and will assist the system in decoding it. This information can either be specified by the stakeholder him/herself or by any other external expert in the general knowledge base (see below).

V. Specification of application processing policies

Stakeholder: policy makers in the company

Policies are not necessarily strongly connected to the application in question, but may instead generally apply to the company, such as contractual arrangements or wider legal constraints. Accordingly, similar to the business goal transformation rules, these policies may be defined once and reused multiple times. Since it is to be expected that these goals are highly company specific they have to be either strongly associated with the company (and used for none other) or selected by the policy makers anew every time. Since these policies will most likely be confidential, they also have to be hosted in highly secure environments.

VI. Specification of Technical Constraints

Stakeholders: IT administrator, developer, similar

Here concrete constraints are put forward to describe how the application is hosted. These may derive from the software architecture as well as other policies and may be implicit knowledge by the software / infrastructure engineers, but they may also incorporate concrete technical constraints in the way the application is configured for this use case. Note that some technical constraints are directly given by the application (see step I).

For example, a technical configuration choice may be that, since the application is configured to use a file system instead of a database, a file system is needed in the hosting environment, even though the application did not necessarily declare that.

VII. Other

Stakeholders: external experts

As described in more detail in the context of the Reasoner, a set of “ground rules” must exist that define the essential expertise. This includes decomposition rules, interpretation rules, Cloud scaling rules etc. etc. They will partially be defined by the Cloud hosts (Cloud providers), but also by general business and technical experts all over the community (“network”). For example, the eScience application may have specialised data processing needs requiring certain levels (performance, latency) of network connectivity between processing nodes. In order to ensure this specific knowledge of node location and bandwidth is needed.

D. Deployment

1. PaaSage Deployment Phase

Regarding the distribution of application data it must be possible to optimise the deployment of the application data in the Cloud with respect to the specified optimisation goals and constraints. This implies specifying a data partitioning model that describes what partitioning is permitted by the application. Similarly, to specify a data consistency model that describes how much inconsistency the application can tolerate. It should also specify the data flow and workflow models for the application.

The deployment specification must also describe the required target Cloud infrastructure. It must be permitted to specify by name potential Cloud providers, e.g. by specifying that a given deployment unit may be deployed on Amazon, Rackspace or ElasticHosts. Specification of constraints on the location of the Cloud provider - for example to respect legal constraints on the location of data – is required. Similarly the specification of requirements on the security and privacy of the Cloud provider infrastructure is needed. Required resource types are specified independently from the specifics of each Cloud provider, such as requesting Cloud storage resources in the form of a file system or a database.

The PaaSage architecture will find potential Cloud providers by matching deployment requirements with a list of Cloud provider models. Cloud provider data will include its location, cost models, resource types, security/privacy model, and other important attributes such as availability or performance of resources.

2. Deployment Storyboard

During application deployment the main component in action is the Reasoner. For the end user the component maintains a link with their requirements passed in as part of the Configuration Model that was formed in the previous step.

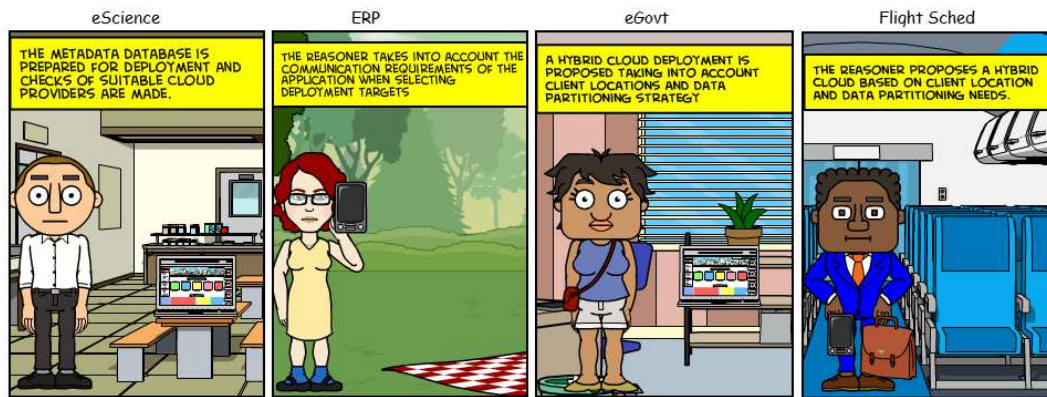


Figure 7 Storyboard Deployment Phase

In the eScience scenario at the deployment phase the metadata database is prepared to structure the deployment of the large scale application. Using data and knowledge in the metadata database, checks of Cloud Providers related to the workflow and access to data will be made.

For the ERP scenario at deployment the main concern is the communication links and data processing balance between the Cloud and the mobile clients. The deployment of nodes could be made to ensure specific effort is made with the synchronisation of online / offline mobile clients.

Central to the eGovernment deployment is management of how data is partitioned along with services in a Hybrid cloud. The Reasoner will ensure that QoS is respected in selected cloud infrastructures to the extent that the more essential data to the application is positioned on infrastructure with greater reliability and QoS than non-essential data / functionality.

Airline Scheduling at deployment has to ensure that the consistency of data is maintained as the office based application is rapidly (re-)distributed across nodes in the Cloud. Deployment Phase Functionality

1. Pre-selection of Constraints and Data Preparation for Reasoner ("Profiling")

Stakeholder / Component: Profiler

Although the Profiler belongs to the configuration phase it is worth noting that the constraints, rules, policies etc. given already constrain the deployment possibilities due to two reasons: (1) direct conflicts in the specification and (2) experience, along the line of what consequences typically arose / did not apply. Effectively, this means pruning the search tree for the Reasoner: whilst the Reasoner could principally perform all these operations itself, it would take considerably longer time (as there would be an exponential search tree explosion).

In effect, the Profiler thus generates (and maintains, see execution phase) a set of models describing all execution relevant information that the Reasoner has to optimise over. The Profiler thereby incorporates expertise from software & model analysis to interpret the data obtained and cross-references it against the rules and constraints given.

Concretely, each Deployment Model maintains the following set of specific requirements:

- Application requirements

Describes all information necessary to execute an application instance according to the intentions by the developer and host. This means that it includes the following information

- The individual software components of the application;
- The software architecture (work- and dataflow);
- The execution behaviour in the sense of when which component created which load on resources;
- The basic machine readable scaling rules according to execution expertise and software architecture (such as that scaling out helps to increase performance in module A if number of users are larger than X);
- The application specific constraints related to deployment in the case of PaaS (such as needs SQL database, needs license X, can only run on Azure);
- The general application constraints related to offering / selling the application (including maximum total cost, total latency, maximum number of users.);
 - Quality of service / deployment constraints;
 - The constraints and conditions of the individual application instance such as typical execution speed, typical load, TREC;
 - Module specific behaviour rules, such as under which load to scale out etc.
- Host requirements

Describes the specific conditions and constraints set by the Cloud provider. It also includes, next to the basic set up of the infrastructure and hosting capabilities, monitored information and their logical consequence for the specific Cloud provider:

 - The types (storage such as file system, devices and databases computation capabilities such as VMs) and amount of resources available as well as the types of the resource instances;
 - Monitoring Services, what access and data do they provide.
 - The quality requirements, such as the effective bandwidth and latency during execution, the typical resource load. This can be matched to application requirements to allow better searching;
 - The general rules and constraints, including the license and cost requirements;
 - Typical behavioural constraints and rules, such as how long it takes to perform a scale out, when scale out should be typically performed. etc.
- Data requirements

Describes the structure of the data being consumed / produced in the application in the widest sense. This may well be an inherent part of the application requirements.

 - Size;
 - Consumption / production pattern (data flow);

- “Type” (structured, unstructured);
 - Security/Privacy/Affinity Policy/Constraints;
- User requirements

All information related to a specific (class of) user(s), such as typical requirements, preferences and typical usage behaviour e.g. types of devices and mobility.

The requirements are used by the Profiler to create a set of constraints, rules and policies in a “Reasoner-readable” format that effectively span the minimal search tree, i.e. with all conflicts eradicated. Notably, conflict-resolution may require feedback from the user. These are presented to the Reasoner in the Configuration Model.

II. Optimisation and decomposition

Stakeholder / component: Reasoner

The Reasoner takes all rules / functions from the Configuration Model as generated by the preceding steps and tries to find a deployment configuration fulfilling the constraints and ideally optimising them. The Reasoner will generate a deployment configuration (graph) building up from the workflow / software model, which identifies all deployment boundaries and low level scaling rules that can be enacted by the execution engine.

The Reasoner does thereby NOT generate rules “out of the blue”. This means that an according set of rules and functions must be pre-generated. This includes next to the input by user or developer also “common ground rules”. The Reasoner will resolve unknown parameter values for these rules, and select the set of rules appropriate for the current configuration. Examples: "Scale out if more than X users by adding a new VM" will see a numerical value for X, and all rules applicable for Azure will be removed if the Amazon offering is chosen for the deployment.

The Reasoner creates the Deployment Model which contains a collection of possible deployment configurations (options), possibly ranked, and linked to real time monitored data and historical execution data from current and previous related deployments.

E. Execution

The PaaSage platform aims to optimise Cross-Cloud deployments with respect to deployment goals and constraints. The first PaaSage architecture will focus on optimising performance and cost of Cross-Cloud deployments. It will support the deployments of the four case studies and be general enough to be widely applicable supporting the deployment of multi-tier applications as well as workflows. It must also allow applications to scale up and down in the Cloud within the confines of constraints set in the PaaSage models using functions of the Cloud providers such as Elastic Hosts [22]. The optimisation of deployments will also learn from past deployments. This will be done by mining execution history in the metadata database and by running complex queries on the history of runs. The aim of the learning is to find which executions gave the best results as well as the underlying reasons for those results.

The PaaSage platform will also provide support for optimisation of data partitioning and replication. The aim is to find the optimal data partitioning and replication

deployment that meets the data consistency constraints. The optimisation will use the data partitioning, data flow, workflow and data consistency models from the deployment specification.

The PaaSage platform will – in addition - provide trusted, secure and privacy aware Cross-Cloud deployments. A Cross-Cloud monitoring system will support monitoring Cross-Cloud deployments. The PaaSage platform will be evaluated with a few selected Cloud providers such as Amazon, Azure or Rackspace.

1. Execution Storyboard

At execution time the PaaSage platform supports all the use cases by automatically monitoring of the engaged Cloud Infrastructures in line with user requirements passed in from the Deployment Model.

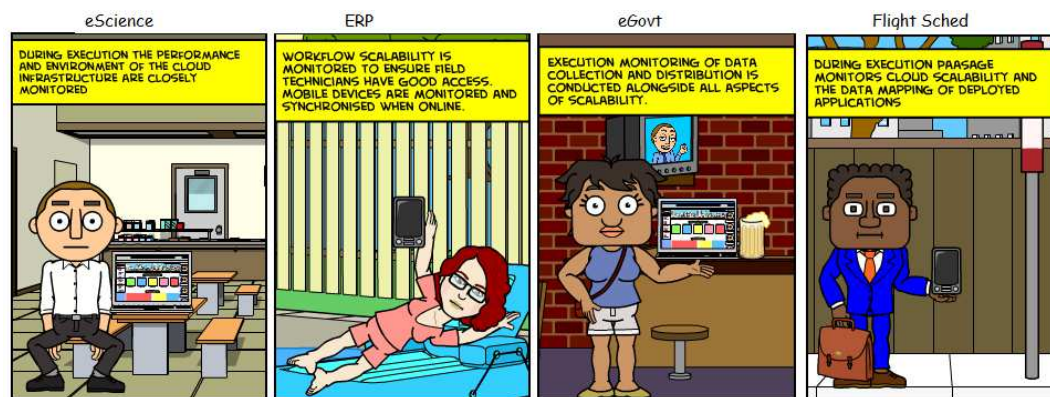


Figure 8 Storyboard during the Execution Phase

During the execution phase in the eScience application the performance and behaviour of the application and Cloud Infrastructure is monitored closely by the PaaSage platform. If a fault occurs the platform can create new instances of the workflow.

The ERP application is also monitored in a similar way and effort is made to ensure mobile devices are synchronised as they come on and off line. Possible adaptations in the case of large volumes of offline devices can be the creation of more services to increase availability for online technicians.

Application execution in the eGovernment scenario is focused on scalability to serve all municipalities and monitoring to ensure data integrity and security. Adaptation takes place to ensure the balance between public and private data processing is balance to ensure the scalability of the Cloud.

The Airline Scheduling use case during execution has a focus on the collection of distributed data and its processing to create composite views. Monitoring is of great importance to maintain the integrity of data and the application as demands are put on the Cloud scalability. Adaptation to maintain access to remote datasets and security of data is integral to the platforms management of the Cloud in this scenario.

2. Execution Functionality

1. Adaptation towards the host

Stakeholder: Adapter (Upperware and Executionware)

The role of the adapter is to transform the currently running configuration into the target configuration received from the Reasoner. In the case of a first time deployment, the currently running configuration is empty. The adapter is then responsible for generating the proper commands to the Deployer which is responsible to correctly enact this configuration on the chosen provider offerings. It also provides the Deployer with instructions about the parameters to monitor, and rules to adjust the running system within the boundaries of the target configuration. For instance, if the configuration says that up to 10 VMs can be used, then the execution engine can safely scale up to 10 VMs using whatever scalability rules that applies for the chosen provider.

Continuing the above example, the addition of another VM can be made by the execution engine every time another 100 users are using the system (this will be prompted by monitored data analysis by the execution engine). Yet, the adapter does not need to know about each new user entering the system; it only needs to know when the execution engine adds another VM to make sure that the number of VMs stays within the deployment boundary of fewer than 10 VMs. In other words, the adapter does not care about the specific adaptation process for a given Cloud environment (see below), but cares specifically about all configuration steps needed for the proper “orchestration” of the execution.

When it is detected that the current configuration is no longer valid, i.e., outside the constraints set by the Reasoner as implied by the monitored data, the adapter asks the Reasoner to produce a new target configuration, and subsequently adapts the running system to this target configuration.

The Adapter takes the Deployment Model and adds knowledge from sources such as previous executions from the metadata database and Cloud monitoring framework. The result is a set of individually deployable artefacts and a set of configuration scripts to start each artefact on the given provider to which it has been allocated.

II. Deployment

Stakeholder: Deployer (Executionware)

The actual deployment according to the specific host characteristics and requirements and the low-level execution environment (i.e. selection of the right monitoring engine and the right interpreter etc.) is performed by the Deployer. The Deployer is provider-specific and only deals with the components to be deployed in the respective designated environment – it has no view on the total system.

The Deployer produces the initial deployment of the individual components and their execution environment. Note that the according components / images may still be inactive until actually triggered

III. Execution

Stakeholder: Execution Engine & Interpreter

Execution is triggered with the first request from the business application end user with whatever external trigger is required. This trigger is external to PaaSage, but must be catered for in the sense that the destination must be reachable.

During execution, application requests triggered by the respective module are converted from the PaaSage API into operations specific to the respective environment

the component is hosted on. These operations can range from storage access to actual manipulation of instances.

Under best circumstances, the execution will simply follow the work- / dataflow and finalise its process. During execution monitored data about workflow / application execution is created.

IV. Monitoring

Stakeholder: Monitor (Executionware)

For getting information about the currently running VMs, PaaSage makes use of the monitoring framework offered by the Cloud providers. This enables gathering status information, such as network load, processor load. In order to execute rule-based actions, the PaaSage monitor can principally query any further data source, including other monitors and/or the metadata database. Monitoring may thus also supervise invocations performed on the component and actions taken by the execution engine. What is actually monitored and where / how the monitoring data is delivered is defined by the monitoring rules and their selection by the Reasoner.

The monitoring information is captured according to the specification (needs) of (a) the Profiler (stored in the metadata database and passed in encoded form to the Reasoner), (b) the execution engine, and (c) the adapter (stored in the metadata database)

V. Local Adaptation (reconfiguration)

Stakeholder: Execution Engine / Interpreter

Given certain conditions as registered by the monitor, such as that the network is overloaded, the execution engine can take adaptation actions in order to compensate for these conditions. The engine thereby follows no intelligence, other than the one explicitly provided by a set of behavioural rules provided with the deployment Configuration Model. These rules include actions such as when to scale out, when to scale up etc.

As a consequence of such actions, consistency needs to be maintained depending on the (lack of) support by the respective infrastructure.

The execution engine only takes actions within the respective environment, i.e. will not directly contribute to Cloud-bursting or Cross-Cloud deployment of a single component. Such adaptations necessitate a global reconfiguration of the deployment.

VI. Global Adaptation (reconfiguration)

Stakeholder: Adapter, Reasoner, metadata-database

Not all reconfiguration takes place only within the Cloud environment local to the component. We can identify the following situations (among others) where more global adaptation is required by the PaaSage platform:

- The local resources become insufficient, meaning either that:
 - More additional resources are needed (bursting);
 - A different host is needed (relocation);
- Multiple connected components need to be adapted at the same time
(Note that this can potentially be achieved using local execution rules);

- The information gathered so far indicates that the system is seriously misbehaving (e.g. missing critical constraints) and the local adaptation does not seem to compensate it;
- The Reasoner has found a better deployment

Such conditions should be detected by the Adapter through complex event processing on monitoring data available in the monitoring infrastructure (the metadata database provides summaries and pointers to the raw data). When the Adapter detects that the running system is outside the current configuration once obtained from the Reasoner, it invokes the Reasoner to produce a new configuration. If this new configuration is deployable under the application invariants (checked by the simulator in the Reasoner and Adapter), a set of configuration scripts, one for each used platform, is generated and passed on to a platform-specific Deployer. The operation of the adapter is shown in Figure 8.

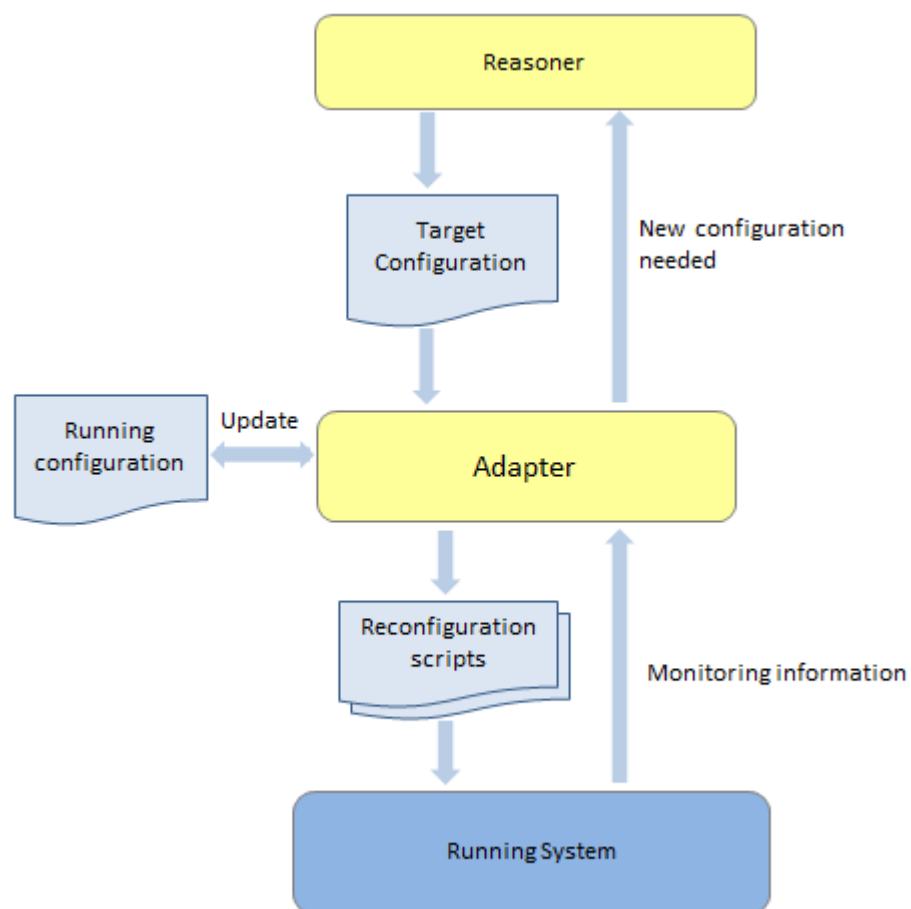


Figure 9 The global adaptation loop

Adaptation cannot just consist of a new deployment configuration without further details. Instead it must be an adaptation script (containing undeploy and redeploy instructions) in order to specify exactly how the adaptation takes place. This involves aspects such as:

- Graceful shutdown;
- Smooth transition;

- Which configuration change;
- Which new instance is required (for which Cloud);
- Which instances to be destroyed;

The Adapter produces an incremental deployment that starts with the current deployment and changes it. Once the new configuration (and the way of achieving it) is specified, the new deployment configuration is fed to the Deployer.

IV. COMPONENT DESCRIPTIONS

The previous section of the document has explained the main functionalities, in high level view, of the PaaSage project. We have looked at the main functionality and how it relates to our Use Case needs via the Storyboards. In this next section we shift the focus onto a more detailed view of the individual components that make up the PaaSage platform.

A. *IDE*

The IDE component is the start point at which the user / application designer engages with PaaSage. The IDE will be presented as an Eclipse plug-in and will support a variety of modelling standards such as UML for creation of applications for the PaaSage platform. The output from Eclipse will be translated by the Plug-in for integration with the interfaces which will be defined for the components described in more detail below.

1. **Cloud Modelling**

PaaSage embraces modelling efforts in the MODAClouds project [4] and we rely on that project for CloudML input and development into the PaaSage project. Initial work on the architecture indicates that the cloud modelling language required by PaaSage – at least for configuration and the earlier phase of deployment - is much richer than the CloudML being developed in MODAClouds.

CloudML allows the creation of novel models and meta-models for enterprise systems, focusing on architectural styles for platform independent solutions. At present it appears that it is useable within PaaSage only for the adapter and last phase of the Reasoner but not for the IDE, Profiler and much of the Reasoner. This is because CloudML lacks the ability to capture effectively information outside of the scope of the application's Cloud topology.

In order to improve CloudML we aim to develop greater support within the standard for the expression of constraint and rules associated with the Clouds architecture and data management actions. This will be achieved in PaaSage by the development of CAMEL which will include domain specific languages such as CloudML from MODAClouds.

The CAMEL standard will focus on delivering greater capture of constraints set in the application design process. It will support the expression of rules and integrate with other standards for rule based decision. As part of this integration we are investigating how Ontology can help support and develop the CAMEL standard in the setting and verification of rules based on system knowledge. CAMEL is likely to be formed from a group of DSLs including SALOON, WS-Agreement, OWL and a rule-based language for constraint expression and management.

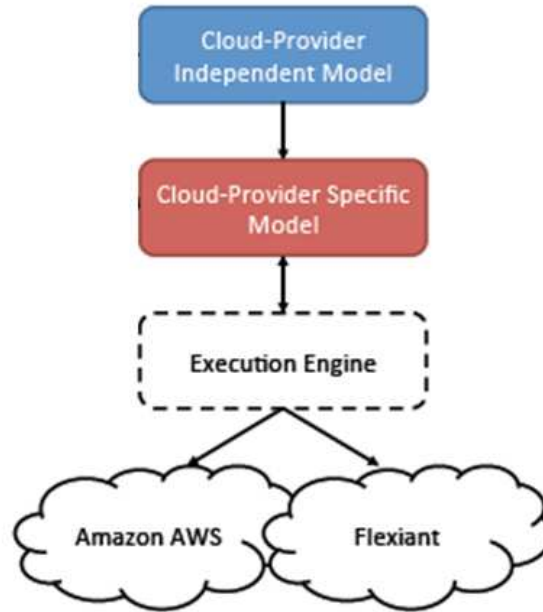


Figure 10 CloudML

At design-time, the Cloud application developers use a CAMEL to specify the provisioning and Deployment Models with additional input from system administrators and data administrators. These models encompass the topology of the nodes of the Cloud infrastructure, as well as the topology of the software artefacts deployed on these nodes.

CloudML considers provisioning and Deployment Models at two levels of abstraction, namely Cloud Provider-Independent Model (CPIM), and Cloud Provider-Specific Model (CPSM).

A CPIM represents a generic provisioning and Deployment Model that is independent of the Cloud provider. This model consists of two main kinds of elements, namely the node types and the artefacts types. A node type represents a generic virtual machine (e.g., a virtual machine running GNU/Linux). This element can be parameterised by provisioning requirements (e.g., 2 cores _ compute _ 4 cores, 2 GiB _ memory _ 4 GiB, storage _ 10 GiB, location = Europe).

An artefact type represents a generic component of the application (e.g., a Java servlet of an application for document collaboration, a Jetty container, and a MongoDB database). This element can be annotated with deployment commands (e.g., retrieve the Java servlet from <http://www.paasage.eu/>, configure it, and run it), deployment dependencies (e.g., the Jetty container and the MongoDB database have to be deployed before the Java servlet), and communication channels (e.g., a Java servlet communicates with another Java servlet through Hypertext Transfer Protocol Secure (HTTPS) on port 443).

The CPIM can be serialised using two formats, namely the JavaScript Object Notation (JSON) and the XML Metadata Interchange (XMI).

B. Profiler

The main objective of the Profiler is to look into the list of goals and preferences (which are set by various users), and come up with a list of potential candidate providers that satisfy the aforementioned inputs and other additional constraints like SLA and elasticity rules. An example of goals set by the organisation and defined by the business user is minimizing the response time and total cost, whereas a list of preferences could be running the user application on Amazon in Europe instead of in Asia / USA and deploy the database on the Private Cloud.

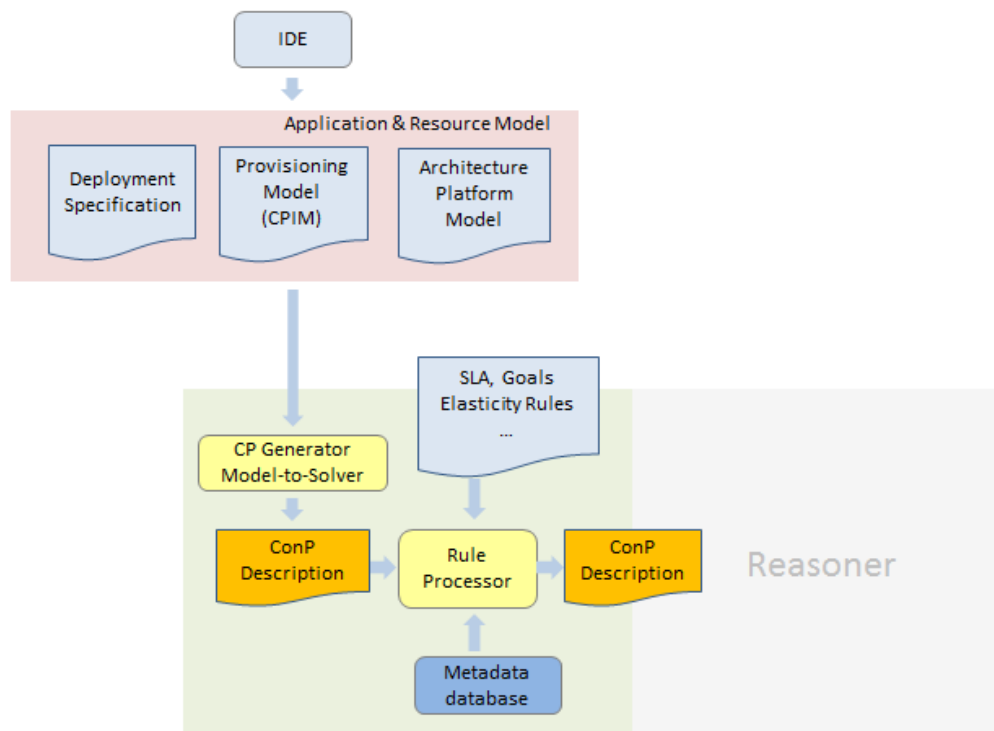


Figure 11 Architecture of the Profiler

As shown in Figure 11, the Profiler interacts with the IDE in getting a list of models to be processed by the *Constraint Programming (CP) Generator*. Then, the *CP Generator* is responsible for producing a *CP Description* that defines a list of input constraints for future deployments. Finally, the *Rule Processor* takes this description along with other inputs, such as SLA, elasticity rules, goals, and real-time information from the Metadata Database, to generate a list of possible and feasible deployments (defined in a new *CP Description*) that will be used by the *Reasoner*.

1. CP Generator

The CP Generator looks into several application and resource models that are defined in the IDE and produces a *CP Description* that contains a list of deployment variables, domains and constraints. It is also the responsibility of the *CP Generator* to prioritize the constraints and variables, and resolve any conflicting parameters from the models.

The CP Generator identifies variables, domains and constraints by analysing the input application and resource models, and the deployment specification. The CP Generator

produces a CP Description that lists variables and their domains constraints derived from the input models and deployment specification.

1. Rule Processor

The Rule Processor is responsible in generating a list of possible and feasible deployments (defined in a new *CP Description*) that satisfy all the given constraints and inputs. The Rule Processor works by processing of additional information related to the application to complete the CP Description. It also verifies the CP Description (e.g. remove redundant constraints, detect variables without domain, etc.)

The Rule Processor receives as an input the *CP Description* from the *CP Generator* that that defines a list of input constraints for the future deployment. These include Elasticity Rules, preferences, goals, and SLA, along with initial values of monitored resources (e.g. response time, memory usage, etc.)

The Rule Processor produces a CP Description that a list of possible and feasible deployments wrapped into the Deployment Model expressed in CloudML. Moreover, it contains resource parameters to be monitored (e.g., memory and disk usage).

C. Reasoner

In a nutshell the Reasoner receives application and context models (from the Profiler) in CAMEL format and outputs Deployment Models in CAMEL. This process relies on the Reasoner extracting requirements from the CAMEL and using the current state of Cloud Infrastructure and knowledge from the metadata data base to conduct reasoning. The component architecture can be seen in Figure 11.

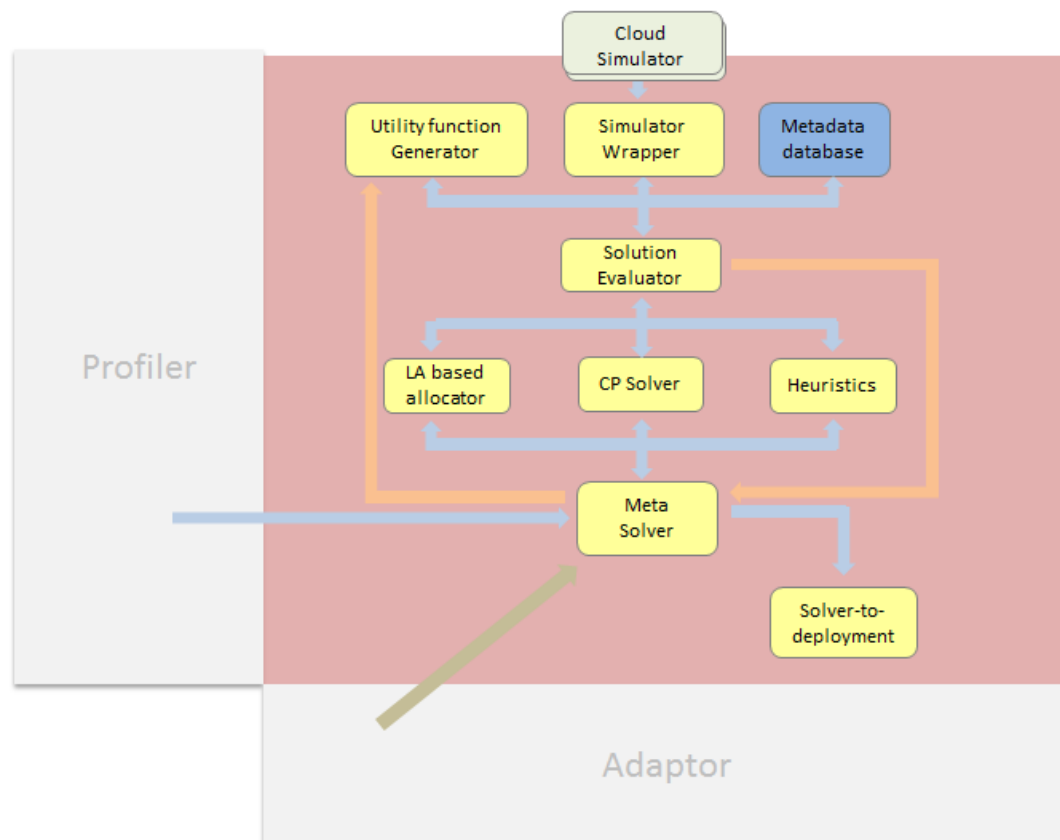


Figure 12: Components that make up the Reasoner

Central to the Reasoner is the concept of Solvers. The Solvers sit at the centre of the component and conduct the main functions in the Reasoner.

1. Solvers

The role of a solver is to assign a value to a variable from the variable's domain so that all constraints of the problem are satisfied. A set of values assigned to all variables of a problem is called a *configuration*, and a configuration that satisfies all the constraints is called a *feasible configuration*.

There will typically be many feasible configurations, and the number of feasible configurations grows exponentially with the number of variables of the problem. One would normally not be satisfied with *any* feasible configuration, but rather try to find the feasible configuration that is “best” according to some quality criteria, e.g. system perceived *utility*. It should be noted that the utility could be returned as measured “goodness” of the deployed system; it could come as a result of a simulated deployment; or from the evaluation of a functional expression. In other words, the term *utility function* will be understood indiscriminately of all these three ways its value can be obtained as an abstract mapping that takes as input a configuration and returns a value that describes the quality of the deployment according to the given configuration.

2. Meta solver

There are many different algorithms, or *solvers*, that can be used to assign values to the problem variables depending on the relation among the variables as being linear or non-linear, and the domains of the variables as intervals over the real numbers or as integers, including binary decision variables. The *meta solver* will select one or more solvers appropriate for the problem, and dispatch to these the problem or part of the problem. It will also receive feedback from the configuration constructed by the set of solvers chosen, and may use this to change the solvers used for building the next configuration in the subsequent iteration.

Finding the optimal configuration is normally only possible for certain restricted problems, and in general one will have to evaluate every possible feasible configuration in order to assess *a posteriori* the best configuration. This is impractical for all but the smallest problems. In reality one will therefore need to run the solvers for as many iterations as allowed by the time budget available for finding a configuration. It is a task of the meta solver to control the execution of the individual solvers, and stop or pause them when a solution must be returned.

It is anticipated that the search for an improved solution can continue in the background even after one has decided to go for deployment of a particular configuration. In this way one could have one or more better configuration ready should there be necessary to adapt globally the running configuration for some reason.

3. CP Solvers

Constraint programming (CP) simply refers to a set of variable domains and their associated variables whose relations are defined in terms of a set of constraints. It does not specify how and in which order these variables are assigned values, and what algorithms to use for finding these values. If the domains are intervals of real numbers,

and the constraints and the utility function are all *linear*, it is a *linear programming* problem. *Non-linear programming* problems do not require linearity [23].

There is a plethora of CP solvers available, both commercial ones and open source. However, they are generally not able to operate with stochastic variables, which are output as a result of the variance platforms / application performance measurements. They can therefore most likely only be deployed for the sub-problem consisting of real valued variable domains and deterministic variables. On the positive side, they are normally capable of finding *optimal* configurations for quite large problems in polynomial time. For deterministic variables that are discrete, special solvers from the domain of *combinatorial optimisation* must be applied [24].

4. Learning Automata (LA) based allocator

When the variables become stochastic, the problem gets worse. If one had statistical data with samples of utility function values for a large number of runs, one could use *statistical interference* to estimate and test hypotheses about deployment outcomes for each configuration [25], [26], [27]. One will normally not have the luxury of a huge database of previous deployments, and it is therefore necessary to resort to methods that are able to learn the better variable values as new observations of the utility function becomes available. Given that the variance of the mean value decreases with $1/\sqrt{N}$ for N observations, we will get better and better estimates for the mean characteristics as we get more observations. This leaves us with two options: We can defer making any decision until we have a large number of observations, or we can use methods that are able to learn better and better as new observations come along. If the domains of the variables are continuous, one could use *parameter identification* techniques to assign variable values [28]. However in the case of discrete variables selecting the right value for a variable becomes a *Markovian Decision Problem* [29], for which *reinforcement learning* algorithms [30] can be deployed.

A special sub-set of reinforcement learning algorithms called *Learning Automata (LA)* [31] will be used in PaaSage. LA are characterised by having a firm mathematical foundation allowing core properties like scalability and convergence to be rigorously analysed. Furthermore, when many values are assigned to many variables of the same problem, one automation can be given the task of assigning one value. One would thereby exploit the concept of an *automata game* [32] in order to converge to a feasible configuration, and a proposal for an LA based solver for PaaSage can be found in [33].

Heuristics (search algorithms) Given that the solver can be any algorithm that is able to assign values to the variables from their domains while respecting the constraints of the problem, one can deploy as a solver any method capable of doing this assignment in a stochastic environment, as stated by the *No Free Lunch* theorem [34]: “For all possible performance measures, no search algorithm is better than another when its performance is averaged over all possible discrete functions”.

There are many different search algorithms available in the literature, and they can broadly be classified in two groups: Those algorithms aiming at finding the globally best configuration [35], versus the algorithms starting with a rough first guess of a solution and then trying iteratively to refine the solution [36]. The latter class of stochastic local search algorithms will be preferred in PaaSage because they will at any time be able to return the best configuration found until that point.

5. Utility Function Generator

The utility function generator will use the information about goals and preferences distilled by the Profiler into the Constraint Programme Description. The role of the utility function is to provide a quick alternative to simulating the deployment, or to make the actual deployment, in order to have feedback on the “goodness” of a particular configuration. The different solvers are all, in one way or the other, iterative and for each iteration towards a feasible solution feedback on the usefulness of the current configuration is needed. A utility function is traditionally the way to assess a proposed deployment in self-adapting software systems [37].

Experiences [38] show that it is very hard for the system designer to formulate a good utility function, and one often has to resort to a weighted sum of the different measurable goals and preferences [39] as it is easier for a human operator to tune the preferences and priorities of the different goals, and thereby implicitly adjusting the weights of the utility function sum.

One will therefore necessarily need to try capturing the imprecise goals and preferences in the utility function, and the purpose of utility function generator sub activity in PaaSage is to investigate more sophisticated ways of doing this than just a weighted sum. Given that fuzzy reasoning [40] has proven useful in making decisions under uncertainty, fuzzy methods will be the point of departure for the investigations on a more representative utility function.

Bearing in mind that the main task of the utility function is to guide the search for solution, one has the added benefit in PaaSage that the same configuration can be subjected to an evaluation by the utility function as well as by the deployment simulator. In this way it is also possible to obtain feedback from the simulator on the quality of the utility function itself and adjust the utility function accordingly. Hence, PaaSage may in this way iteratively improve the utility function making it more and more trustworthy as a quick way to evaluate a candidate configuration.

6. Solution Evaluator

The Solution Evaluator module aims at offering a standardized function evaluation interface to all solvers. It forwards the function to evaluate to the Utility Function Generator, the Simulation Wrapper, or the Metadata Database (MDDb) depending of the function to evaluate. The parameters are of course different depending of the actual evaluator. For example, the Simulation Wrapper needs a lot of metadata to describe the cloud to be simulated. If fuzzy methods are unable to capture adequately the user’s goals and preferences, we will look at other methods like statistical regression to construct the utility function as a weighted combination of the problem’s variables based on past execution history. An invocation to the MDDb can be triggered for example to retrieve some historical data; hence, metadata to describe how to evaluate the historical data are needed (mean, average, duration of data to take into account, etc.).

7. Simulator Wrapper

Simulator wrapper is a way to hide the mechanism used to obtain a feedback on a particular configuration. The wrapper can either start a simulation, or it can evaluate the utility function. The feedback provided by the wrapper to the solver is supposed to be consistent in the sense that a better configuration receives a better feedback value.

Thus the module aims at wrapping a Cloud simulator such as SimGrid [41]. It converts application and resource descriptions of PaaSage into a Cloud simulator specific format. It will also convert the results of a simulation into the needed PaaSage model. The simulator may be able to interact with the Solver to test resource allocation decisions, *i.e.* mapping but also What-If questions.

The simulator generates traces that have to be translated as simulation feedbacks to the Solver. The traces contain the life-cycle of all the resources used and the cost per unit of time of running the application. The traces must log all the requests arrival. It must also contain the time to process a request at each tier and the Round Trip Time for each request.

The simulator needs a simulation request for a given application on the whole (or a part of the) platform composed of possibly multiple Clouds. Accordingly, it must be able to interact with the meta-data database for retrieving information about the platform such as the description of the resources, *i.e.* Physical Machines and their inter-connections, the different billing schemes, monitoring information about different resources, availability of instance types, virtual storage and network resources, etc. This interconnection will take the form of a translator between the platform model used by the meta-data database and the one used by the simulator.

Another interconnection will be between the application model and the simulator. A translator will transform the application model to the simulator one. Furthermore, the application model may be enriched with information contained in the metadata database. Indeed, the simulator needs to have access to these information to run accurate simulations based on real-world observation and developer provided models.

8. Constraint Logic Programming

An alternative way of determining suitable Deployment Models given an application model and several Cloud resource models is to follow a logic-based matchmaking and optimization process. In this approach, Cloud infrastructure descriptions are translated into logic-based knowledge in the form of predicate facts. Similarly, the application model (along with any other deployment requirements and goals) is expressed in the form of predicate or constraint goals. Then, matchmaking between application requirements and infrastructure offerings is performed based on a set of constraint satisfaction rules and optimization objectives leading to a set of ranked deployment configurations/solutions.

Rules can either be resource-related (low-level) or referring to application characteristics (high-level). For instance, a low-level rule could provision a virtual machine with low disk throughput to an application with low storage requirements. A high-level rule, on the other hand, could satisfy a constraint that two tasks be deployed geographically close to each other by deploying them on VMs offered by the same Cloud provider. Rules can also be used to transform high-level requirements to low-level ones to enable their direct matching with respective low-level (Cloud resource) capabilities, leading to more accurate matchmaking and optimization results.

Rules can be expressed by deployment experts or derived from learning processes based on deployment history. The deployment history can also be inspected and processed so as to produce new facts, *e.g.*, providing some performance insights from previous practical experience. Thus, an important characteristic of the rule base, as well as the fact base, is that they should both be dynamic, quickly adapting to any changes

implemented by infrastructure providers or new deployment-related knowledge that may be acquired.

The above described matchmaking and optimization process can be implemented using a constraint logic programming (CLP) approach, realized using Prolog and Constraint Handling Rules (CHR). The approach can simultaneously consider multiple optimization objectives, even under over-constrained requirements. It also has the ability to simultaneously support more complex requirements and preferences provided in the form of disjunctions of sets of constraints. In the general case, matchmaking can yield multiple deployment solutions, which can be ranked by exploiting the Analytic Hierarchy Process (AHP) to prioritize optimization criteria and normalize their values based on particular utility functions that can allow the slight violation of particular optimization objectives to cater for solution feasibility.

9. Solver to Deployer

This module translates the output of a solver into the Deployment Model representation. It also participates to lowering the dependencies of solver to the remaining of PaaSage. This module is strongly linked to the Model-to-Solver module.

D. Adapter

The adapter has two main responsibilities. First, it is responsible for transforming the currently running application configuration into the target configuration in an efficient and consistent way. Second, it is responsible for performing high-level application management, which involves monitoring and adapting components deployed on multiple cloud providers. The adapter is composed of three components: the plan generator, the adaptation manager and the application controller.

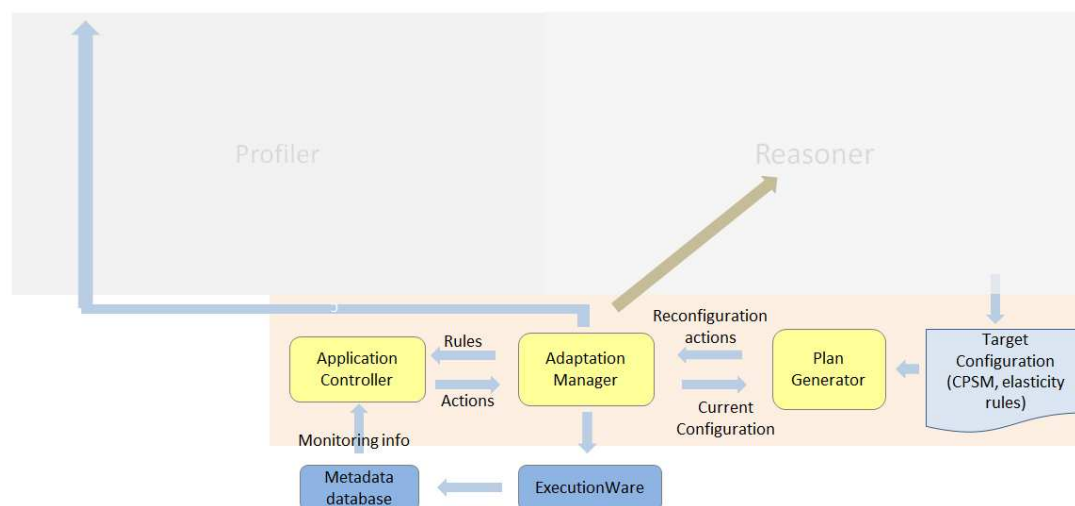


Figure 13 Architecture of the Adaptor

The Adapter receives information on the target application configuration of the Deployment model expressed in CAMEL. The Adaptor processes this model to produce a CAMEL Execution model that contains deployment descriptions, including software artefacts and rules.

1. Plan Generator

The plan generator compares the target configuration (Deployment Model) which it receives from the Reasoner with the running configuration and generates an efficient and correct reconfiguration plan, containing an ordered set of reconfiguration commands. This is expressed in CAMEL and associated domain specific languages and known as the Execution Model.

The Plan Generator sends the Execution Model to the Adaptation Manager for further checking. If any inconsistencies are present in this model it is sent back to the Plan Generator for reconfigurations taking into account the feedback. During its operation the Plan Generator uses knowledge / policy from the MDB in the construction of its models.

2. Adaptation Manager

The adaptation manager is responsible for driving the reconfiguration process across one to many Clouds. First, it validates the reconfiguration plan by estimating and comparing reconfiguration costs and benefits. If the plan is valid, the manager applies the plan by sending deployment descriptions to the Deployer and global rules to the application controller. The manager also minimises inconsistencies in the presence of reconfiguration failures. If the plan is not valid, the manager asks the Reasoner for a new target application configuration. After applying the plan, the manager updates the running configuration.

3. Application Controller

The application controller implements high-level management policies that need global knowledge or involve multiple cloud providers, such as policies involving cross-cloud migrations. The controller collects information on the application execution, evaluates global rules, and triggers reconfiguration commands.

E. Metadata Database

The metadata database (MDDB) follows the architecture depicted in **Error! Reference source not found.** The MDDB layer comprises the metadata model and the implementation of the distributed physical store (which includes federation capabilities); the Analytics layer, providing support for a variety of analytics over historical metadata; and interfaces to the Profiler, Reasoner, Executionware, and Social network infrastructure components. The MDDB is meant for long-term preservation of information. It is designed to associate mutations with a wall-clock timestamp and to trace the identity of the sources of mutations. It thus shares principles with archival systems, temporal databases, and provenance systems.

1. Metadata database layer

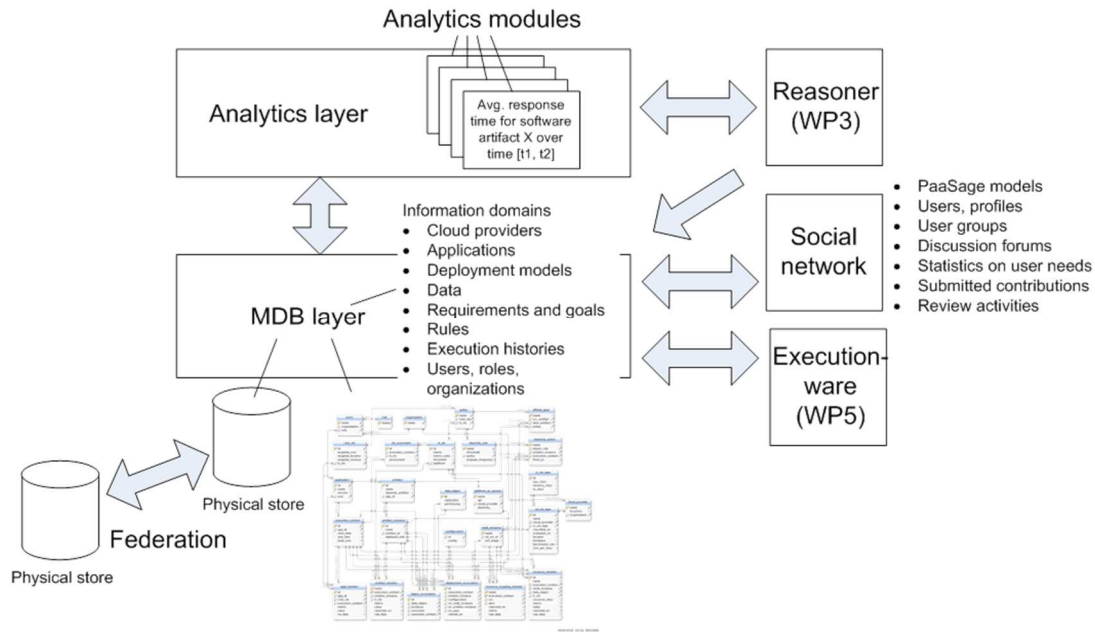


Figure 14: Metadata database architecture

The MDDB model (whose schema in standard E/R notation is shown in **Error! Reference source not found.**) describes the applications and their deployment adopting principles from specifications such as CloudML, PIM4Cloud, and TOSCA and extending them for the unique needs of PaaSage. In more detail, the meta-model is meant to capture

- The description of an application;
- Application requirements and goals;
- Runtime aspects of its execution histories such as monitoring information at different levels, invocations of rules and policies, and quality of service assessments.
- Rules and policies;
- Provisioned resources;
- Cloud provider characteristics;
- Users, roles, and organizations.
-

Next we provide a detailed presentation of the MDDB information organized along these thematic areas.

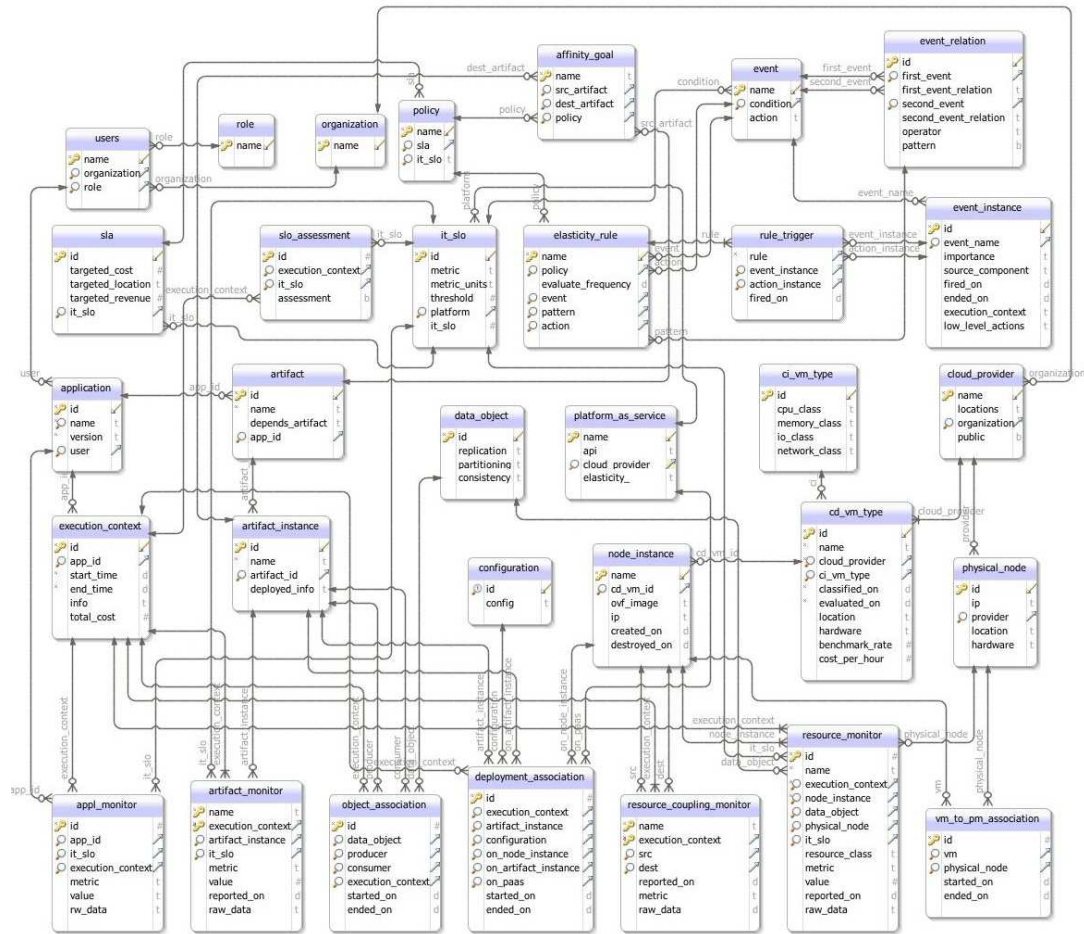


Figure 15: Metadata database schema

Application descriptions

The MDDB stores application descriptions expressed in CloudML. It additionally extends those descriptions to express lifecycle management concepts such as the evolution of the application and its deployments over time. A version of an application is rooted at an APPLICATION object and comprises software ARTIFACT and ARTIFACT INSTANCE objects, which correspond to generic and specific software component descriptions respectively. An ARTIFACT INSTANCE can be deployed either on another ARTIFACT INSTANCE or on a NODE INSTANCE representing a VM resource. The deployment relationship is a *temporal association* represented by a DEPLOYMENT ASSOCIATION object (with a start and end time). In addition to descriptions of software components, the data used by them and their characteristics (replication, partitioning, consistency) are expressed in DATA OBJECT classes. Data objects are connected to software artifacts via temporal OBJECT ASSOCIATION classes (a data object is typically connected to its producer and consumer components). Object associations model data flow within application descriptions.

Application requirements and goals

Application requirements and goals are expressed as service-level objectives (SLOs) or other types of constraints on the deployment and/or behavior of applications. In the MDDB schema, requirements and goals are represented by SLA (service-level agreement), IT SLO, and AFFINITY GOAL classes. SLA expresses non-IT (business level)

constraints such as targeted overall cost, location preferences/restrictions, etc. SLA expresses the fact that a top-level constraint implies an agreement to support the required constraints in addition to expressing an objective. IT SLO expresses requirements on an IT metric, such as throughput and response time. The IT SLO class describes the metric and its units, as well as the desired threshold. It is connected to the software artifact on whose operations the objective applies. An IT SLO may or may not be translated into a service level agreement during deployment (for example, the expressed objectives may be taken into account but no hard guarantees provided on them). AFFINITY GOAL expresses dependencies between artifacts, such as the requirement to place two software components physically or logically nearby or far apart (e.g., place components so that they fail independently –i.e., in different availability zones- and/or so that their communication path is optimized –i.e., within the same communication domain).

Runtime aspects and application execution histories

The application requirements are connected to monitoring information represented by APPLICATION MONITOR, ARTIFACT MONITOR, RESOURCE MONITOR, and RESOURCE COUPLING MONITOR. Each monitor relates to the metric specified in the corresponding service-level objective and to the type of object to be monitored (i.e., application, artifact, resource). It is important to note that the MDDB monitoring objects contain highly aggregated information rather than raw monitoring data; the latter is managed separately by a time-series database. All monitoring information related to a specific execution of an application is connected to an EXECUTION CONTEXT object (featuring a start and end time of the execution as well as other aggregated information such as cost of the run). The EXECUTION CONTEXT is also connected to one or more SLO ASSESSMENT objects (evaluations of the degree to which an SLO was achieved) and deployment information for the application indicating which artifact instances were deployed on which artifact or node instances and what was their configuration. Note that the validity intervals (i.e., time duration from start to end time) of an execution and a deployment association can differ—in other words, a particular deployment may participate in several executions.

Rules and policies

The MDDB meta-model expresses rules and policies, an example of which is the ELASTICITY RULE that dictates an adaptation action in response to a violation of an IT SLO to which the rule applies. Rules are associated with a specific event, which comprises a condition (e.g., a metric violating a set threshold) and an action. Event and action manifestations during execution are expressed as EVENT INSTANCE and RULE TRIGGER objects connected to the corresponding execution context of an application. More general rules relating to the occurrence of any type of event can cover general cases of application adaptation. Additionally, the MDDB supports the definition of event relations (represented by the EVENT RELATION class) constructed as expressions connecting events or relations to other events or other relations via logic operators. An event pattern is defined as an event relation that is responsible for triggering a rule. The MDDB is planned to adopt and interoperate with established standards in this space, such as the Esper event-condition-action (ECA) rule and event processing language.

Provisioned resources

Each NODE INSTANCE (a CloudML concept referring to a deployment container) is of a particular CD VM TYPE and CI VM TYPE, where CD stands for *Cloud dependent* and CI for

Cloud independent. A CD VM TYPE describes a real-world VM type offered by a Cloud provider (such as for example Amazon EC2 m1.small or a specifically configured Flexiant FCO VM). CI VM TYPES are the result of (periodic) classifications of Cloud-specific VM types into Cloud agnostic resource classes, performed by the MDDB runtime Classification is based on a systematic benchmark-driven methodology to produce a vector of performance metrics (CPU, memory, and I/O) that characterize each supported VM, followed by statistical clustering (using for example the k-means algorithm) to categorize VM into Cloud-agnostic class such as SMALL, MEDIUM, and LARGE.

Cloud provider characteristics

Cloud providers are described in CLOUD PROVIDER objects, including information such as data center locations and whether the Cloud provider is of private or public type. Organizational information about Cloud providers is modeled separately (as described below). Their offered higher-level programming platforms (such as Java 2 Enterprise Edition, etc.) are described in PLATFORM AS SERVICE objects; modelling of such platforms is expected to draw information from related projects in this space, such as Cloud4SOA. To model Private Clouds where PaaS can have visibility in the underlying physical infrastructure, the PHYSICAL NODE and VM-TO-PM ASSOCIATION classes describe characteristics of physical machines (e.g., CPU architecture, number of cores, etc.) and temporal mappings between physical machines and the VMs deployed on them over time.

Users, roles, organizations

The users, roles, and organizations associated with the rest of the modeled entities describe information on the users and other stakeholders of particular applications, the roles that they play, the organization to which they belong, and the organization Cloud providers correspond to. This information is expressed in the USERS, ORGANIZATION, and ROLES classes (designed along the lines of the ideas developed in the development of the CERIF data model²).

The physical MDDB store will be designed for scalability and high availability through the use of parallel database technologies and principles such as horizontal data partitioning across distributed server nodes. Extensive use of the Eclipse Modelling Framework (EMF) is an incentive to leverage Eclipse Connected Data Objects (CDO) technology for its support for disconnected operation and a variety of distribution mechanisms depending on the connectivity level between the distributed CDO stores (where, e.g., some might be close to the partner/project component locations to allow for fast interconnection and transferring of information). The size of the MDDB will depend on how it is deployed into or across organizations and what application domain it belongs too. For example the MDDB for a collaborative eScience set of active applications would contain more history and data than a single instance used less frequently with fewer users in a private organization.

An important concern to address is the integration of PaaS metadata databases originating from different installations of the PaaS system. We expect that cases where the metadata databases to be integrated do not conform to exactly the same DB schema (due to variations in the version of PaaS used in different installations), will be common. A solution that we intend to exploit in such cases is the use of Ontology

² <http://eurocris.org/Index.php?page=CERIFreleases&t=1>

as a common schema to bridge the gap between the two databases. In particular, we first define a common Ontology to cover the concepts and relationships involved in the databases to be integrated. Then, we map each database model/schema to that of the Ontology. In this way, any DB-specific schema discrepancies are resolved by the mapping and hidden to the PaaSage user. The user will need to know only the Ontology schema in order to pose (SPARQL assuming the ontology is encoded in RDF) queries to the system and thus any information that is differently represented in the databases will be presented to the user in a unique, uniform way. The mechanisms supporting this mapping guarantee that the relational data of the database can not only be transformed to semantic data but also updates on the relational data are propagated to the respective semantic knowledge base. The architecture of the envisioned model (with all the components involved, including the Analytics Manager) is visualized in **Error! Reference source not found.**

In terms of technology support, we propose the use of the standardized RDB2RDF language proposed by W3C, called R2RML (<http://www.w3.org/TR/r2rml/>), a powerful and expressive language already supported by several Semantic Knowledge Bases / Triple Stores (along with the required synchronization functionality), such as Virtuoso, D2RQ and Oracle's Spatial and Graph RDF Semantic Graph. The above process additionally covers the case where databases are heterogeneous with completely different schemas (such as for example when an external contributor collects data that is modeled differently). This can happen for instance, when a user of the Social Network desires to offer his/her data to the PaaSage community.

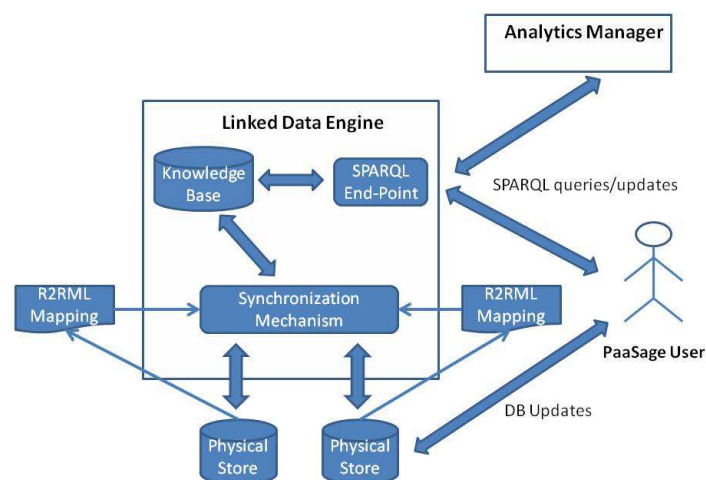


Figure 16: Integration of PaaSage metadata databases

2. Analytics Layer

The analytics layer will be responsible for performing various types of analytics (e.g., computing statistical measures over existing metrics) over historical metadata for the whole application or its components through exploiting the Analytics Manager component. Apart from exploiting the monitored data stored in the MDDB (in the form of the values obtained for some metrics), this component will also interface with the

Monitoring Engine of the Executionware in order to obtain additional information, such as raw measurement data as well as aggregated information.

The analytics layer will also comprise a Reasoning Engine that will be able to derive new knowledge by exploiting the content of MDDB via the execution of rules. The new knowledge will be stored in a structured way (complying with an Ontology schema) within a knowledge base (KB) and be continuously informed through the execution of the rules over the MDDB and the KB itself. Through the derivation of new knowledge, the PaaSage system will be able to: (a) perform simple queries over the KB, answerable in a shorter time compared to direct complex querying of the MDDB; and (b) exploit the knowledge derived in order to provide extended (e.g., always suggest trustful cloud providers) or added-value functionality (e.g., use rules to enable the automated matching of application components/artifacts to Cloud services). **Error! Reference source not found.** depicts the architecture of the MDDB with the KB accompanying one or more MDDB physical stores. A difference between **Error! Reference source not found.** and **Error! Reference source not found.** is that whereas in the former the Analytics Manager is shown to draw its input from the KB (in other words, the KB has already performed significant mining of the MDDB), in the latter it is shown to have the ability to explore directly the two MDDB physical stores. In general, both options should be provided for flexibility. Knowledge, whether generated by the Reasoning Engine or by the Analytics Manager, is stored in the Knowledge Base.

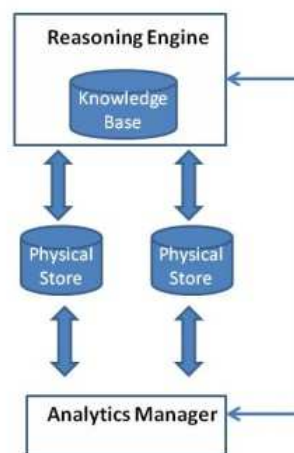


Figure 17: PaaSage knowledge base and reasoning engine

3. Social network infrastructure

The PaaSage social network aims to engage the open-source community (both users and developers) into the PaaSage model-based platform-independent code development model. The open source community will benefit by leveraging previously-captured historical knowledge (such as, which module / combination of modules achieves the desired results on which platform(s)), via cost / benefit feedback at development time, deployment suggestions, best practices, etc. The social networking platform will also

motive the open-source community to contribute knowledge from independent experience, complementing the information discovered by the PaaSage Upperware.

The social network will offer various features to its users, such as a *forum* through which users can communicate and exchange information and a *graphical user interface* through which various user tasks can be performed like connecting with other similar users, posing questions to the MDDB, and contributing knowledge and metadata from personal experience. To this end, the infrastructure supporting this social network and its goals should be able to store information, such as PaaSage models, user information in models, statistics on user needs and submitted contributions as well as support the proper functioning of the forum and the graphical user interface.

The architecture of the Social Network infrastructure is depicted in **Error! Reference source not found..** A standard user can: (a) contribute to the social network by describing his/her expertise and areas of interest (applications, Clouds, etc.) and providing his/her own metadata database contents, and (b) participate and learn by joining groups of like-minded users, participating in discussions and posing questions. A user should be allowed to specify a number of keywords of interest (e.g., “applications involving a JEE application server and a SQL database”, “anything over the Flexiant Cloud”, “anything using the Amazon Elastic Java Beans platform”) and receive notification when a contribution comes in that relates to any of them.

Other users can engage in discussions with a standard user. An expert user is enlisted to translate a standard user’s questions to database queries (possibly after a number of direct queries) or to validate his/her contributions to the knowledge base. An expert user will also be able to guide standard users through the content contribution process (there should be an auditing phase involved to ensure the validity of the data). A special type of user, the GitHub/devops user, is particularly targeted due to bringing together the well-established GitHub developer community with the Cloud deployment and service engineering communities. With increasing credit, a standard user can be elected an expert user and be allowed to join the ranks of super users.

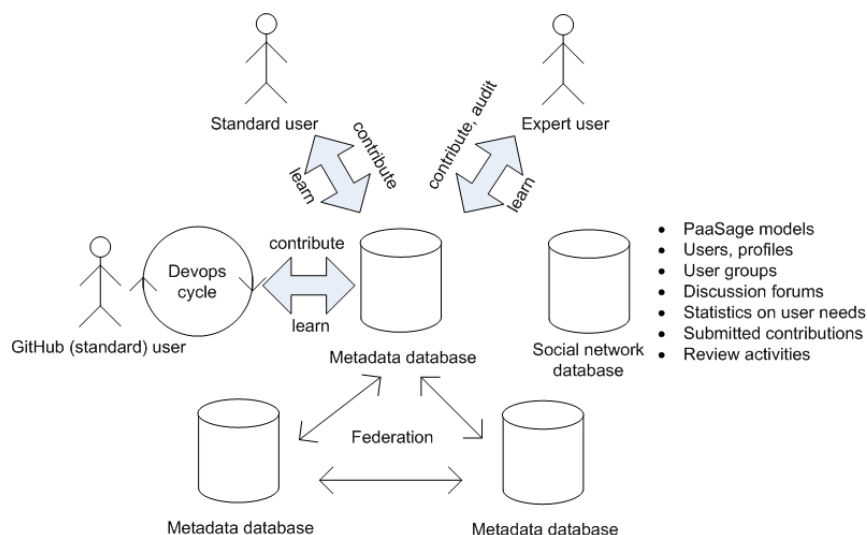


Figure 18: The architecture of the Social Network infrastructure

4. Trust and Identity Management

Central to the integration of the PaaSage metadata with third party data is the need to authenticate and authorise contributors. PaaSage support of an Identification, Authentication and Authorisation mechanism for contributors is linked to the establishment of an Identity Management mechanism. By using identity information we plan to associate data with specific contributors which will enable the establishment of identity rooted reputation and trust models around contributed data in PaaSage.

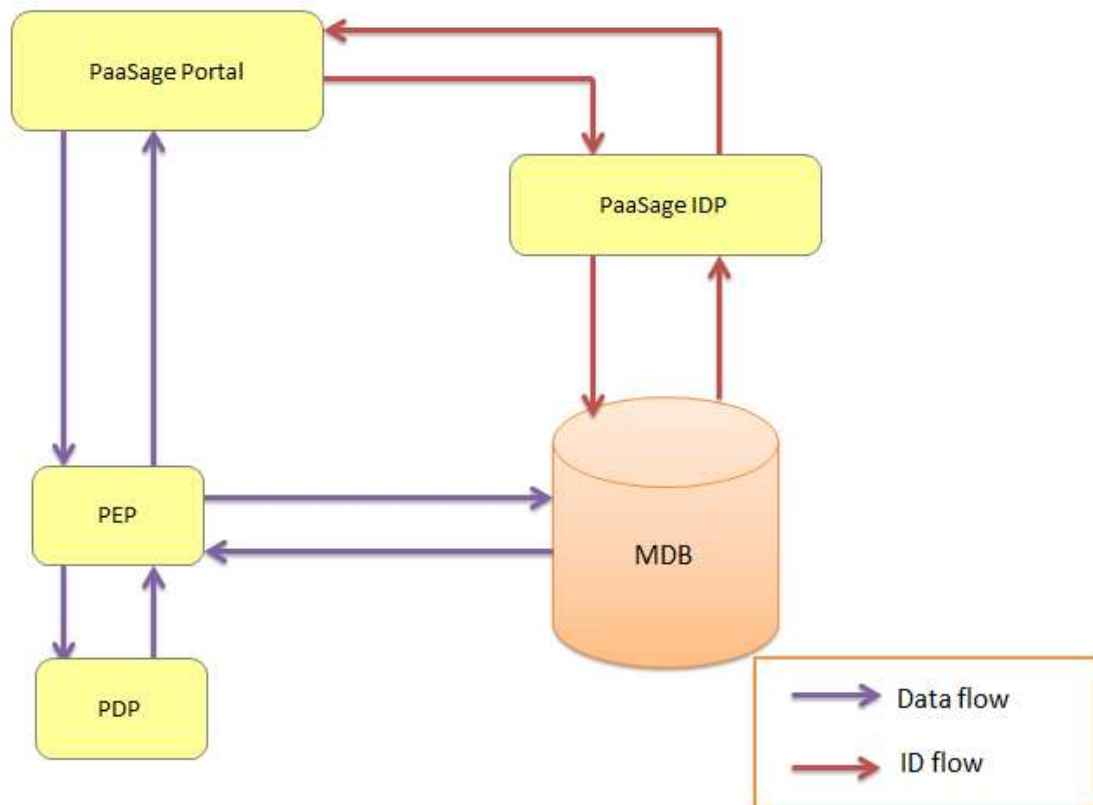


Figure 19 Identity Management in PaaSage

Figure 15 illustrates and initial design for the identity management in PaaSage. It is expected that users both platform users and third parties will authenticate through a PaaSage portal. This could be a web service interface for automated calls or a specific web front end for users.

Here the participants can login via a federated ID. For example users could use SAML2 tokens from other federated PaaSage platforms or present OpenID credentials for checking by the portal. Once authenticated by the portal the user is issued a PaaSage identity token for the session that they are authenticated for. This token will specify the user's privileges in PaaSage.

As data is sent for storage or retrieval from the MDDB checks on the identity token of the user is performed at the Policy Enforcement Point (PEP). The PEP checks policy associated with the data in the MDDB against privileges in the user token. The checking is performed by the Policy Decision Point (PDP) which then issues an accept or deny response to the PEP. Based on the response the action on the MDDB is either permitted or rejected.

The security policies in the framework are to be defined and could directly relate to the reputation / trust model of identities in the platform. Policy would be applied to restrict access to specific data for certain groups of users or ensure that specific users are prevented from adding types of data to the MDDB.

F. Executionware

The main purpose of the modules and artefacts provided by the Executionware are to enable the execution of the individual components (services) of the PaaSage application in a fashion that the overarching goals and constraints are met. The Executionware thereby forms the lowest level of support in the PaaSage system, meaning that it has no understanding of the whole application – both in terms of the application description, and the constraints / requirements. Instead, the Executionware concentrates primarily on the individual components and how they need to be adapted in order to meet *their part* of the requirements and boundary conditions.

The Executionware directly builds on functionality offered by the various Cloud platforms and by intermediate software layers such as middleware frameworks. In particular, the first prototype of the Executionware will base on the Cloudify³ and jClouds⁴ frameworks.

The Executionware gets low level deployment rules from the Upperware. These rules enable the Executionware to (a) (re)deploy the various application components across diverse cloud platforms and (b) to perform low-level adaptation operations depending on the current execution conditions. In order to perform such adaptation operations the Executionware relies on monitoring information gathered from the run-time system of the application components. It further may make use of events issued by other components when they perform their individual adaptations.

Summarising, the Executionware only gathers the specified information from (local) monitoring and assesses it against a set of given rules to perform an according operation. It is thereby the task of the Upperware to ensure that the application components (services) are chosen to be deployed in an environment that supports the necessary actions in terms of (1) communication, (2) adaptation operations, and (3) monitoring. The operations that the Executionware principally has to support and to realise relate to the primary concepts of Clouds. This means that the Executionware has to enact operations as listed in the following. The operations are triggered by sequences of events matching rules. Again, the respective rules must come from a higher-level instance, in particular, the Upperware:

- Moving (relocating) the VM;
- Creating new instances of a service (scale out);
- Replicating status / data;
- Destroying instances (scale in);
- Scaling an instance up and down (e.g. increasing size of the database);

³ <http://www.cloudifysource.org>

⁴ <http://jclouds.incubator.apache.org/>

The Executionware has to reside close to the component that it supervises in order to ensure that the necessary information is available and that the necessary actions can be performed. “Close” thereby meaning that it should at least reside within the same host environment (same Cloud infrastructure) and potentially even on the same resource. For instance, monitoring has to be co-located with the Executionware, as only the Executionware is aware of the actual mechanisms provided by the platform running a particular component instance.

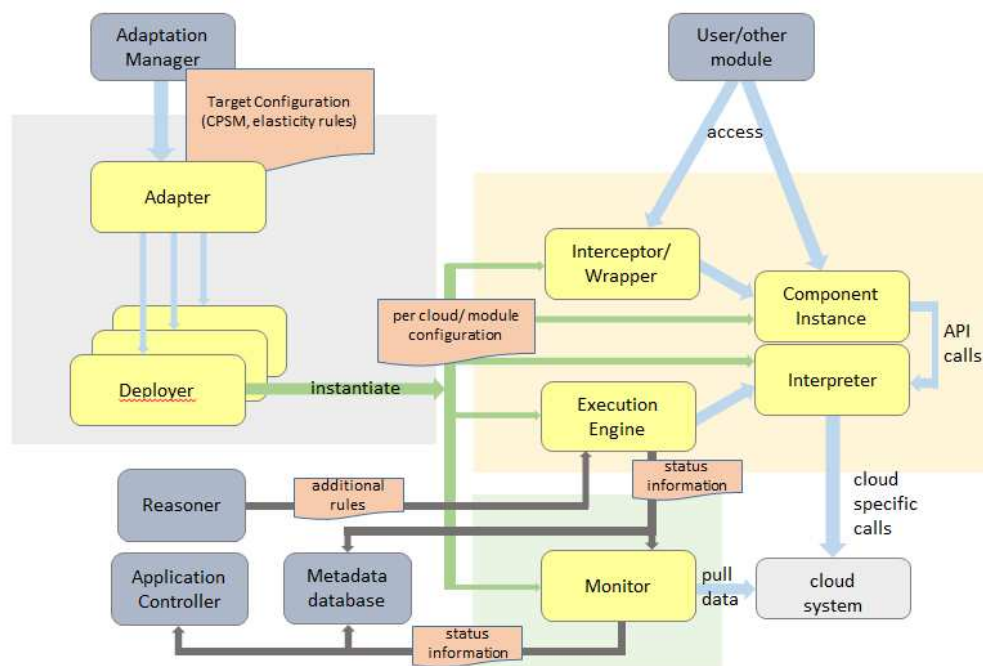


Figure 20 Initial Architecture of the Executionware and its interfaces

The overall architecture of the components of the Executionware is shown in Figure 20. We discuss them in the succeeding sections.

1. Component Instance

The component instance is the code part (application component/artefact/instance) that is treated as a single (black) box by the PaaS system. This is an individual part of an entire workflow application. Even though it may be split up or a composition itself, once deployed, it is considered a single instance.

As the component instance is treated as a black box, the interfaces it provides to users or other parts of the application can vary and are generally unknown to the Executionware.

2. Component Wrapper/Message Interceptor

When the interface of the Component Instance is known, the Component Wrapper exposes a virtual interface to the Component Instance, so that the invocations and messages calls reach the Component Wrapper before being relayed to the Component Instance. This way, the Executionware can get full control over the Component Instance

even when the environment does not allow such fine-grained control. Wrapping Component Instances also allows retrieving more fine-grained monitoring information. The Component Wrapper may perform any actions on the message (including measuring, routing, extending etc.) prior to relaying it. Even though the Component Wrapper will generally be deployed together with the Component Instance this is not absolutely necessary. In case only information about messages is required, the Component Wrapper may be realised as a message proxy.

The interface of the Wrapper is identical to the interface provided by the Component Instance. In addition, the Wrapper may contain a management interface to retrieve monitoring data and to configure its functionality dynamically. The Component Wrapper is by far the most sophisticated component in the Executionware. Its specification will be further detailed at later stages of the project.

3. Deployer

The Deployer executes the necessary steps to deploy the component instance(s) along with its/their execution environment and configure the rules according to the specification of the Deployment Model. The Deployer is specific for a dedicated cloud environment, i.e. there will be different implementation of a Deployer for each cloud environment as long as the differences cannot be abstracted by some cloud middleware such as cloudify/jgroups. The Deployer ensures that the correct number of component instances is deployed and further enables the monitoring of system parameters for these instances as requested by the deployment configuration. In addition to the component instances the Deployer further configures and deploys an Enforcement Engine that is responsible for micro-managing the set component instances it has deployed.

The Deployer receives from the Adapter deployment information for one specific application component targeting on specific cloud platform. Beside the component code, the deployment information further specifies the number of instances to start, security configurations, as well as routing configuration, if required. It also contains information about which data to monitor for all deployed component instances.

4. Enforcement Engine

The Enforcement Engine is the management entity of the Executionware. It captures the monitoring stream from all instances and matches it against the specification of the local scalability rules. When a rule matches, the Enforcement Engine delegates the action further to the Interpreter. The rules engine used in the Enforcement Engine is similar to a policy engine and effectively only evaluates a set of event-condition-action triples. The engine has no intelligence beyond the rules provided with deployment of the module instance/artefact and the execution components. It may contain a set of hard-coded rules that “always” apply, though – such as “general knowledge”. In this case, these rules should principally be capable of being over-written. Apart from processing the log stream itself, the Enforcement Engine may relay the monitoring information to the meta-data database and to the Adapter, if necessary. In that case, it will ensure a normalisation of the monitored data so that data from different cloud systems has the same format and scale when stored and processed outside the Executionware. If further evolution of PaaSage requires compression or pre-processing of monitoring data, the Enforcement Engine is the right place to add it.

The Enforcement Engine receives a set of scalability rules from the Deployer that contain a set of event-condition-action triples to be evaluated against the monitoring stream.

5. Monitor(s)

Monitors gather the relevant data directly at the component instances and relay the data further to the Enforcement Engine (and from there to the meta-data database). The monitoring data serves for taking decisions on the overall application deployment as required by Adapter and Reasoner. In general, the module will be a slim wrapper around the monitoring capabilities provided by the cloud platform and cloud infrastructure. Accordingly, every infrastructure may have its own implementation(s) of the monitor. In the remainder of this section we describe a distributed monitoring architecture for multi-tier applications deployed on multi-clouds.

The monitor does not receive any input. It outputs monitored data in a platform-specific format.

Figure 21 depicts a framework for multi-cloud monitoring and adaptation of service-based applications (see [5] for a more detailed exposition). The framework focuses on monitoring infrastructures that operate in a cross-layer manner.

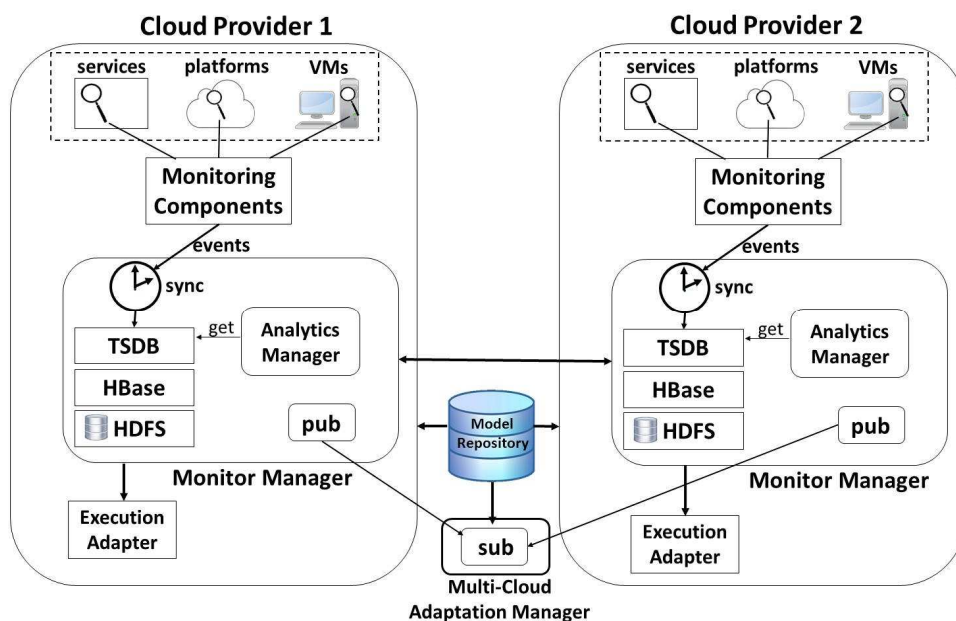


Figure 21: Multi-Cloud monitoring and adaptation of Service-based Applications

In a multi-cloud setting, service-based applications are deployed on various Clouds based on the capabilities of the respective Cloud platforms. By considering that various layers are involved in the deployment and execution of a cloud-based application, monitoring should be performed at all layers, i.e., the SaaS, PaaS, and IaaS. The main monitoring functionality is encapsulated by the Monitoring which retrieves monitoring

information, stores it a time-series database (TSDB), and reports events of interest (such as detected service-level violations) via a publish/subscribe mechanism to Adaptation Engine instances.

Per-Cloud, federated TSDBs are used to provide persistent event storage of time-stamped events. They additionally perform rollups (e.g., aggregated metrics such as average, max, min) for user-specified intervals. A variety of commercial and open source TSDBs can be used to handle time stamped events. In terms of possible technological realisations of the framework, a TSDB especially designed for distributed systems with high scalability requirements would be a suitable candidate among possible choices (the open source OpenTSDB⁶ a prominent candidate).

A publish/subscribe mechanism handles transferring raw monitored events and TSDB rollups to an Adaptation Engine. Different adaptation-engine instances may be deployed to distribute adaptation load across applications/Clouds, where each engine is interested only in relevant events and rollups. One possibility for communicating events and rollups between TSDB and an Adaptation Engine is to use a pub/sub event notification service. In terms of promising technologies in that front, Siena⁷ is one choice that is expressive enough to capture all appropriate event information via an extensible data model without sacrificing scalability and performance during event delivery.

Monitored events from within each Cloud are directed to a local TSDB instance, which can use distributed non-relational key-value store technology (such as Apache HBase⁸) to organize the event time-series. HDFS⁹, a distributed file system replicating data across all Cloud providers, handles time series storage. To achieve high performance during event collection, each Cloud's local replica is updated eagerly; remote replicas are updated in a relaxed (asynchronous) manner. Reads are performed from local copies when available. The monitor manager includes the synchronization and publishing mechanisms on top of TSDB.

6. Interpreter

The Interpreter is the interface to operations and behaviour modification on a per-component-per-cloud platform basis. Its task is performing the actions triggered by the small-scale scalability rules. Since the rule language may differ from the API of the infrastructure, this means that the respective action needs to be interpreted (translated) into a set of host-specific invocations. Generally, the Interpreter will be tightly integrated with the Enforcement Engine, but multiple Execution Engines may share a single Interpreter. The interpreter is triggered by the Enforcement Engine with the action that is to be executed and it in turn transforms it into a sequence of operations that can actually be executed by the hosting environment. Accordingly, like the Monitor and the Execution Engine, every platform/infrastructure may have to have its own implementation of the interpreter.

6 <http://opentsdb.net/>

7 <http://www.inf.usi.ch/carzaniga/siena>

8 <http://hbase.apache.org>

9 <http://hadoop.apache.org>

The Interpreter receives an action to execute such as 'scale up component X' together with all data and configuration information required to execute the action. This may include the component code, required monitoring information and wiring data.

V. OPEN ISSUES AND FUTURE WORK

This document is the first description of the PaaSage architecture developed during year 1 of the project. Little implementation and use case integration has taken place by this stage and is scheduled to properly start in year two of the project. This work will lead to more requirements and potential changes in component descriptions and numbers. The current document describes our direction of travel and records our initial work and collaboration.

Central to the architecture is the use of models. Critically, our models are reliant on innovation provided by the MODAClouds project in the form of CloudML. PaaSage is focused on both guiding and channelling this innovation into our application demands. In support of these models we are also identifying domain specific languages to be used alongside CloudML within the CAMEL. As the project progresses and it is expected that CloudML will be extended to support increasing functionality (especially in the early configuration and deployment lifecycle phases) as the project progresses.

As PaaSage is implemented effort will be made on the integration of the platform with sources of third party data. These will include social networks and other online communities. Part of this process will be the development of a PaaSage community to support the exchange of knowledge between PaaSage implementations. Such a community and data integration plan also depends on the development of a Trust fabric in the project based on factors such as reputation. Other work in this domain will involve engagement with existing work to share online data particularly in the Open Data community and research projects such as Engage [42].

Infrastructure monitoring particularly cross level between SaaS, PaaS, IaaS is a focus of work for the next year. This development is need in order for PaaSage to develop Executionware and knowledge based services. The behaviour of services and platforms has to be both monitored and captured and cross referenced. Thus, the creation of services in the platform to gather and classify this data is a significant piece of future work.

Finally the changes in components and models have the potential to change how we view the Cloud lifecycle with respect to PaaSage and its main components and architectural layers. The current architectural stack represents the direction of travel we currently have in terms of planning development and implementation. However, it is possible that the architecture could evolve to include extra functionality or reflect changes in focus on how PaaSage relates to the Cloud as the project develops.

VI. CONCLUSION

The architecture presented in this document is the result of the first year collaboration between the industrial and scientific / academic technical experts in the PaaSage project. The document provides the core base on which the platform will be extended and for those innovations we intend to deliver to both the Cloud Community and our industrial partners. It is expected that new features will arise through this joint work and experimentation with our use cases.

Within the next year we will expand our requirements in respect to the application of the PaaSage platform in our use cases. This work will also feed into the MODAClouds project and the development of CloudML to suit the functions required in the project.

Building on the current component design and integration in this document will be the major focus of the next year in the PaaSage project. As extra requirements emerge from use cases, and improved functionality in CloudML is presented, our PaaSage platform will evolve and interaction between the functions is set to increase in terms of detail and functionality.

The major effort in year 2 will be in the development of the PaaSage platform using spiral and agile development methods. A first release is expected in month 18 leading to an early prototype in month 21 of the project. Development of the PaaSage platform will continue throughout the project.

VII. ANNEX 1 REQUIREMENTS TABLE

Table 1: overview over the requirements identified in D1.6.1

Requirement Number and Short Description
R-1 NonFunctionalCriteriaAnalysed
The non-functional criteria must be analysed for each application component that is to be deployed in the cloud. Key non-functional requirements include availability, performance, scalability and security.
R-2 ParallelisationCodeAnalysis
The platform could provide code parallelisation based on an analysis of the code. Parallelised code could then be deployed on multicore machines or HPC clouds.
R-3 ApplicationDependenciesIdentified
The platform must analyse the application component dependencies using the software-architecture information. Knowledge of dependencies is required for reasoning about deployments.
R-4 ColocationOfVMDefined
The components that need to be co-located or distributed to different VMs should be known.
R-5 LegacyApplicationsDeployed
A framework for analysing and modelling legacy and cloud applications is required in order to understand their delivery models and find integration solutions
R-6 RequiredCloudTypeKnown
The required Deployment Model, i.e. public, private or hybrid cloud platform could be specified.
R-7 ProbabilityOfGoodFuturePerformanceKnown
PaaSage should provide alternative execution plans based on the probability of good future performance.
R-8 DeploymentLocationPreferencesSpecified
The preferences for location of deployment units should be specified
R-9 PreferredDataPlacementLocationDefined
The preferences for placement of data based on size and location can be defined by the customer.

R-10	ApropriateVMInstanceSizeTypeKnown
The appropriate VM instance size and type must be known.	
R-11	PhysicalEnvironmentsMappedToPlatform
Logical environments, e.g. test or development environment, should be mapped by the platform to physical Cloud Environments.	
R-12	RequiredComponentLatenciesKnown
The different latencies required by the application components should be known. Latencies are used to make decisions on deployments and placements of VM on physical machines.	
R-13	TransientWorflowsSupported
'Transient workflow' should be supported. This means that everything a user does is persistent and available on whatever client he/she works on. When a user moves e.g. from the desktop browser to a mobile client, he/she expects to see the same data after login to the same application.	
R-14	MinCostForMaxPerformanceOfWorkflow
Workflow deployments should be optimised: Minimize the leasing cost while maximizing the contribution to reducing the overall workflow execution.	
R-15	AccessFromMultipleDevicesSupported
Users must be able to access the cloud based application from different devices. Users have different roles (that can change over time), different knowledge about scheduling insights (e.g. expert schedulers vs. supporting staff) and also different environments where they work. A scheduler can e.g. work in his office using a full-fledged power client or he/she can be in a meeting and needs just read-only access to the data over a mobile device	
R-15	InteroperabilityBetweenApplicationDefined
It should be possible to improve interoperability between applications, at least for applications of the same application suite	
R-17	CrossCloudCommunicationSupported
Cloud deployments (Services) must be able to communicate seamlessly across different cloud-based applications .	
R-18	SeamlessMultipleCloudIntegrationSupported
Integration of cross cloud applications should be supported. It should be possible to support a process where a workflow and User Interface is run in a private cloud, but it reuses public data/Open Data-databases, and is integrated with locally installed archiving and accounting systems for a municipality.	
R-19	CloudServiceIntegratedWithCustomerApplications

It should be possible for existing applications with a large local installed base to integrate these with a cloud offering delivering standardized processes where the process is run in the cloud, but closely integrated with the business applications installed at each individual customer.	
R-20	CrossCloudAccessControlIntegrated
The platform must provide methods to ensure access control across datasources	
R-21	EndToEndSecurityGuaranteed
Security concerns must be covered at all time, moving from a private cloud e.g. into a public cloud (even for parts of the system) must be possible in a secure and reliable way.	
R-22	CrossCloudDataIntegrityAndAuthenticity
The integrity and authenticity of data should be guaranteed across clouds.	
R-23	EndToEndDataIntegrityGuaranteed
Data integrity must be guaranteed end to end.	
R-24	ServicesAvailableGlobally
Deployed services should be available globally with the same level of QoS	
R-25	HighAvailabilityOfServices
Access to external interfaces is a vital part for such deployments. Data can be exported and imported using a standard file format and data can be sent to other departments or to partners. High availability is important.	
R-26	OperationalIntegrityGuaranteed
Data must be constantly updated from multiple sources but operational integrity of the data must be maintained.	
R-27	ComponentScalabilityDefined
The components that need to scale must be specified, as well as by how much and when to ensure the e.g. availability and response time requirements	
R-28	OtherScalabilityDefined
Scalability other than elasticity must also be defined, e.g. defining how much memory could be allocated to an application.	
R-29	ElasticityDefined
It should be possible to specify the elasticity and scalability of applications across datacenters, and across business processes over the year	
R-30	ApplicationLoadDefined
The expected application load for an application must be defined	

R-31	SmallAndLargeCustomersServed
Deployed systems should be available every day to customers around the globe, ranging from small to large organisations and using different business models.	
R-32	DataVolumeSpecified
Expected data volumes must be specified	
R-33	CrossCloudDataFlowModelled
It should be possible to model processes and dataflow across cloud and local solutions	
R-34	TestEnvironmentStrictlySeparatedFromOperationalApplication
Test environments should be as close as possible to the real application, but still strictly separated; just another instance in the cloud.	
R-35	TestEnvironmentsEasilySetUp
It should be possible to easily setup different test environments for different test scenarios (e.g. RfC tests, exploration of new business scenarios, integration tests etc.)	
R-36	HybridCloudDeploymentSupported
Hybrid cloud models should be supported with some services running in private clouds and other services running in public/partner clouds	
R-37	GradualMigrationToCloudSupported
It should be possible to gradually migrate an application to the cloud: moving to the cloud will contain trial and error experiences where applications are gradually shifted from locally installed software to gradually more cloud based models	
R-38	MigrationFromTestEnvironmentSupported
Configuring and migrating an application to a new environment is a long and error prone process. These migration steps needs to be executed in a test environment beforehand.	
R-39	RequiredCloudForEachComponentKnown
The appropriate public, private or hybrid platform should be identified for each component.	
R-40	ServerDeployedInCustomerPrivateCloud
It should be possible to deploy database and and application server(s) in a private cloud or or in customer data centers.	
R-41	ServerDeployedInPrivateCloud
It should be possible to deploy servers in in a private cloud, and not only on external public/partner clouds.	

R-42	CloudProvidersKnown
A list of available cloud providers should be managed. The type of each cloud providers should be captured, e.g. Enterprise Software Bus as a service	
R-43	NearOptimalDeploymentCalculated
The deployments that are calculated do not have to be optimal, but should be near optimal.	
R-44	TargetDeploymentEnvMappedToCloudProviders
The required target environment should be mapped to the target cloud providers. This mapping needs to be managed across the deployment lifecycle.	
R-45	ReputationOfCloudProvidersTakenIntoAccount
The reputation of available cloud providers should be managed and it should be based on past performance.	
R-46	ImprovedDistributionOfLoad
The application load should be distributed across the cloud resources.	
R-47	CostTimeTradeoffsTakenIntoAccount
Different types of trade-offs should be taken into account. They should include Cost/time trade-offs and use of private/public clouds (private may be preferred if available – e.g. security may demand that handling certain data cannot be removed from the Private cloud).	
R-48	PriorityOfRequestTakenIntoAccount
The urgency and priority of cloud deployment requests should be taken into account.	
R-49	CloudEnabledDataManagement
It should be possible to deploy on database clouds. The database technology used in a cloud environment needs to be a different one. Topics like the CAP theorem, ACID vs. BEST, the shared-nothing approach etc. needs to be addressed in such application architecture, designed for the cloud.	
R-50	ExternalDataAccessible
It should be possible to easily access other IT-systems within a company and outside of the company from the cloud based application. Time zones should be taken into account when deploying an application in multiple clouds, especially when the application must be accessible globally from anywhere in the world.	
R-51	CloudServiceIntegratedWithCustomerApplications
It should be possible for existing applications with a large local installed base to integrate these with a cloud offering delivering standardized processes where the process is run in the cloud, but closely integrated with the business applications installed at each individual customer	

R-52 ElasticityDefined
Elasticity and scalability across datacenters, and across business processes over the year should be specified
R-53 HighAvailabilityOfServices
It should be possible to other high availability cloud based services. Access to external interfaces is a vital part for such deployments. Data can be exported and imported using a standard file format and data can be sent to other departments or to partners.
R-54 PayPerUseAccountingModel
A cross cloud pay-per-use model should be used. Several models should be investigated such as pay per use, pay as you save, pay one time access fee, or a mix of other models. The issue of aggregation of payment model must be addressed when several providers are involved.
R-55 CostFunctionKnown
The aggregate cost function of a cross cloud deployment should be known to the customers so that they can estimate costs based on different load scenario.
R-56 DeploymentSelected
The human analyst must be able to select a deployment from several deployment scenarios and to easily understand the tradeoffs between the different deployments.
R-57 FullPortabilityMaintained
Full portability of the cross cloud deployments must be guaranteed
R-58 MinCloudAdministrativeOverhead
Access to resources should be easy and minimize administrative overhead
R-59 AvailabilityOfComponentsMonitored
Availability of deployed components must be monitored.
R-60 CloudNetworkOptimisationsSupported
Communication performance depends on the network connection into the cloud. The new architecture should use cloud specific network optimizations
R-61 ResponseTimesMonitored
The response time of deployed components must be monitored
R-62 RelocationBasedOnUserExperience
It should be possible to relocate deployed services and data based on user experience

R-63	RelocationbasedonNetworkExperience
Relocation of services and data based on network experience	
R-64	AdaptationGuidedByPolicies
Adaptation, which is an automatic process, should be guided by policies; for instance should we halt and migrate some VMs to another cloud provider, or just continue running sub-optimally?	
R-65	CloudBurstingSupported
Cloud bursting should be possible from a private cloud should be supported.	
R-66	DeploymentsReconfigured
During execution, there is real-time checking whether the performance is as you expect (via updates to the MD-DB). Several options when SLA is violated, prioritise alternative resources. If performance drops below acceptable levels in the SLA – Halt maybe check point and reconfigure	
R-67	InstancesRestarted
VM instances should be restarted when faults occur	
R-68	IntegratedCrossCloudDeploymentManagement
Framework for «SOA/Cloud» management should keep control on Dependencies	
R-69	DeploymentReportFinalised
After execution has been completed a scenario close out report on overall performance will need to be lodged with the MD-DB	

VIII. ANNEX 2 GLOSSARY OF TERMS

A. *Cloud Related Concepts*

Advertising-based pricing model – A pricing model whereby services are offered to customers at low or no cost, with the service provider being compensated by advertisers whose ads are delivered to the consumer along with the service.

Amazon EC2 – Amazon’s Elastic Compute Cloud Web service, which provides resizable computing capacity in the cloud so developers can enjoy great scalability for building applications.

Amazon S3 – Amazon Simple Storage Services — Amazon’s cloud storage service.

Billing and service usage metering – You can be billed for resources as you use them. This pay-as-you-go model means usage is metered and you pay only for what you consume.

CDN – Content delivery network — A system consisting of multiple computers that contain copies of data, which are located in different places on the network so clients can access the copy closest to them.

Cloud – A metaphor for a global network, first used in reference to the telephone network and now commonly used to represent the Internet.

Cloud Application – a software application that is never installed on a local machine — it’s always accessed over the Internet. The “top” layer of the Cloud Pyramid where “applications” are run and interacted with via a web-browser. Cloud Applications are tightly controlled, leaving little room for modification. Examples include: Gmail or Salesforce.com.

Cloud Arcs – short for cloud architectures. Designs for software applications that can be accessed and used over the Internet. (Cloud-chitecture is just too hard to pronounce.)

Cloud as a service (CaaS) - a cloud computing service that has been opened up into a platform that others can build upon.

Cloud Bridge – running an application in such a way that its components are integrated within multiple cloud environments (which could be any combination of internal/private and external/public clouds).

Cloud Broker – An entity that creates and maintains relationships with multiple cloud service providers. It acts as a liaison between cloud services customers and cloud service providers, selecting the best provider for each customer and monitoring the services.

Cloudburst - what happens when your cloud has an outage or security breach and your data is unavailable. The term cloudburst is being use in two meanings, negative and positive:

Cloudburst (negative): The failure of a cloud computing environment due to the inability to handle a spike in demand.

Cloudburst (positive): The dynamic deployment of a software application that runs on internal organizational compute resources to a public cloud to address a spike in demand.

Cloudcenter – A datacenter in the “cloud” utilizing standards-based virtualized components as a datacenter-like infrastructure; example: a large company, such as Amazon, that rents its infrastructure.

Cloud client – computing device for cloud computing. Updated version of thin client.

Cloud Computing – A computing capability that provides an abstraction between the computing resource and its underlying technical architecture (e.g., servers, storage, networks), enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal

management effort or service provider interaction.” This definition states that clouds have five essential characteristics: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. Narrowly speaking, cloud computing is client-server computing that abstracts the details of the server away; one requests a service (resource), not a specific server (machine). **Cloud computing** enables Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Cloud computing means that infrastructure, applications, and business processes can be delivered to you *as a service*, over the Internet (or your own network).

Cloud Enabler – A general term that refers to organizations (typically vendors) who are not cloud providers per se, but make available technology, such as cloudware, that enables cloud computing. Vendor that provides technology or service that enables a client or other vendor to take advantage of cloud computing.

Cloud envy – used to describe a vendor who jumps on the cloud computing bandwagon by rebranding existing services.

Cloud governance and compliance – Governance defines who’s responsible for what and the policies and procedures that your people or groups need to follow. Cloud governance requires governing your own infrastructure as well as infrastructure that you don’t totally control. Cloud governance has two key components: understanding compliance and risk and business performance goals.

Cloud Hosting – A type of internet hosting where the client leases virtualized, dynamically scalable infrastructure on an as-needed basis. Users frequently have the choice of operating system and other infrastructure components. Typically cloud hosting is self-service, billed hourly or monthly, and controlled via a web interface or API.

Cloud Infrastructure – The “bottom” layer—or foundation—of the Cloud Pyramid is the delivery of computer infrastructure through paravirtualization. This includes servers, networks and other hardware appliances delivered as either Infrastructure Web Services or “cloudcenters”. Full control of the infrastructure is provided at this level. Examples include GoGrid or Amazon Web Services.

Cloud Manageability - You need a consistent view across both on-premises and cloud-based environments. This includes managing the assets provisioning as well as the quality of service (QOS) you’re receiving from your service provider.

Cloud OS - also known as platform-as-a-service (PaaS). Think Google Chrome.

Cloud Operating System – A computer operating system that is specially designed to run in a provider’s datacenter and be delivered to the user over the Internet or another network. Windows Azure is an example of a cloud operating system or “cloud layer” that runs on Windows Server 2008. The term is also sometimes used to refer to cloud-based client operating systems such as Google’s Chrome OS.

Cloud-Oriented Architecture (COA) – A term coined by Jeff Barr at Amazon Web Services to describe an architecture where applications act as services in the cloud and serve other applications in the cloud environment. An architecture for IT infrastructure and software applications that is optimized for use in cloud computing environments. The term is not yet in wide use, and as is the case for the term “cloud computing” itself, there is no common or generally accepted definition or specific description of a cloud-oriented architecture.

Cloud Platform – The “middle” layer of the Cloud Pyramid which provides a computing platform or framework (e.g., .NET, Ruby on Rails, or Python) as a service or stack. Control is limited to that of the platform or framework, but not at a lower level (server infrastructure). Examples include: Google AppEngine or Microsoft Azure.

Cloud Portability – The ability to move applications (and often their associated data) across cloud computing environments from different cloud providers, as well as across private or internal cloud and public or external clouds.

Cloud provider – A company that provides cloud-based platform, infrastructure, application, or storage services to other organizations and/or individuals, usually for a fee.

Cloud Providers – Computing service providers whose product/platform is based on virtualization of computing resources and a utility-based payment model.

Cloud Pyramid – A visual representation of Cloud Computing layers where differing segments are broken out by functionality. Simplified version includes: Infrastructure, Platform and Application layers.

Cloud Security - The same security principles that apply to on-site computing apply to cloud computing security.

Cloud Servers – Virtualized servers running Windows or Linux operating systems that are instantiated via a web interface or API. Cloud Servers behave in the same manner as physical ones and can be controlled at an administrator or root level, depending on the server type and Cloud Hosting provider.

Cloud Service Architecture (CSA) - A term coined by Jeff Barr, chief evangelist at Amazon Web Services. The term describes an architecture in which applications and application components act as services on the cloud, which serve other applications within the same cloud environment.

Cloud Sourcing – outsourcing storage or taking advantage of some other type of cloud service.

Cloud Standards - A standard is an agreed-upon approach for doing something. Cloud standards ensure interoperability, so you can take tools, applications, virtual images, and more, and use them in another cloud environment without having to do any rework. Portability lets you take one application or instance running on one vendor's implementation and deploy it on another vendor's implementation.

Cloud Storage – A service that allows customers to save data by transferring it over the Internet or another network to an offsite storage system maintained by a third party.

Cloud Storm – connecting multiple cloud computing environments. Also called cloud network.

Cloudstorming – The act of connecting multiple cloud computing environments.

Cloudware – A general term referring to a variety of software, typically at the infrastructure level, that enables building, deploying, running or managing applications in a cloud computing environment.

Cloudwashing – slapping the word “cloud” on products and services you already have.

Cluster – A group of linked computers that work together as if they were a single computer, for high availability and/or load balancing.

Consumption-based pricing model – A pricing model whereby the service provider charges its customers based on the amount of the service the customer consumes, rather than a time-based fee. For example, a cloud storage provider might charge per gigabyte of information stored. See also *Subscription-based pricing model*.

Customer self-service – A feature that allows customers to provision, manage, and terminate services themselves, without involving the service provider, via a Web interface or programmatic calls to service APIs.

Data in the cloud - Managing data in the cloud requires data security and privacy, including controls for moving data from point A to point B. It also includes managing data storage and the resources for large-scale data processing.

Detection and forensics - Separating legitimate from illegitimate activity.

Disruptive technology – A term used in the business world to describe innovations that improve products or services in unexpected ways and change both the way things are done and the market. Cloud computing is often referred to

as a disruptive technology because it has the potential to completely change the way IT services are procured, deployed, and maintained.

Elasticity and scalability – The cloud is elastic, meaning that resource allocation can get bigger or smaller depending on demand. Elasticity enables scalability, which means that the cloud can scale upward for peak demand and downward for lighter demand. Scalability also means that an application can scale when adding users and when application requirements change.

Elastic computing – The ability to dynamically provision and de-provision processing, memory, and storage resources to meet demands of peak usage without worrying about capacity planning and engineering for peak usage.

Encryption - Coding to protect your information assets.**External cloud** – Public or private cloud services that are provided by a third party outside the organization. A cloud computing environment that is external to the boundaries of the organization.

Funnel cloud – discussion about cloud computing that goes round and round but never turns into action (never “touches the ground”)

Google App Engine – A service that enables developers to create and run Web applications on Google’s infrastructure and share their applications via a pay-as-you-go, consumption-based plan with no setup costs or recurring fees.

Google Apps – Google’s SaaS offering that includes an office productivity suite, email, and document sharing, as well as Gmail, Google Talk for instant messaging, Google Calendar and Google Docs, spreadsheets, and presentations.

HaaS – Hardware as a service; see *IaaS*.

Hosted application – An Internet-based or Web-based application software program that runs on a remote server and can be accessed via an Internet-connected PC or thin client. See also *SaaS*.

Hybrid cloud – A networking environment that includes multiple integrated internal and/or external providers. Hybrid clouds combine aspects of both public and private clouds.

IBM Smart Business – IBM’s cloud solutions, which include IBM Smart Business Test Cloud, IBM Smart Analytics Cloud, IBM Smart Business Storage Cloud, IBM Information Archive, IBM Lotus Live, and IBM LotusLive iNotes.

Identity management - Managing personal identity information so that access to computer resources, applications, data, and services is controlled properly.

Infrastructure as a Service (IaaS) – Cloud infrastructure services or “Infrastructure as a Service (IaaS)” delivers computer infrastructure, typically a platform virtualization environment, as a service. Rather than purchasing servers, software, data center space or network equipment, clients instead buy those resources as a fully outsourced service. The service is typically billed on a utility computing basis and amount of resources consumed (and therefore the cost) will typically reflect the level of activity. It is an evolution of web hosting and virtual private server offerings.

Internal cloud – A type of private cloud whose services are provided by an IT department to those in its own organization.

Mashup – A Web-based application that combines data and/or functionality from multiple sources.

Microsoft Azure – Microsoft cloud services that provide the platform as a service (see PaaS), allowing developers to create cloud applications and services.

Middleware – Software that sits between applications and operating systems, consisting of a set of services that enable interoperability in support of distributed architectures by passing data between applications. So, for example, the data in one database can be accessed through another database.

On-demand service – A model by which a customer can purchase cloud services as needed; for instance, if customers need to utilize additional servers for the duration of a project, they can do so and then drop back to the previous level after the project is completed.

Pay as you go – A cost model for cloud services that encompasses both subscription-based and consumption-based models, in contrast to traditional IT cost model that requires up-front capital expenditures for hardware and software.

Personal cloud – synonymous with something called MiFi, a personal wireless router. It takes a mobile wireless data signal and translates it to wi-fi. It's pronounced ME-fi, as in "the personal cloud belongs to me — but if you're nice I'll let you connect."

Platform as a Service (PaaS) – Platform as a service — Cloud platform services, whereby the computing platform (operating system and associated services) is delivered as a service over the Internet by the provider. The PaaS layer offers black-box services with which developers can build applications on top of the compute infrastructure. This might include developer tools that are offered as a service to build services, or data access and database services, or billing services.

Private clouds – Private cloud virtualized cloud data centers inside your company's firewall. It may also be a private space dedicated to your company within a cloud provider's data center. An internal cloud behind the organization's firewall. The company's IT department provides softwares and hardware as a service to its customers — the people who work for the company. Vendors love the words "private cloud."

Public cloud – Services offered over the public Internet and available to anyone who wants to purchase the service.

Roaming workloads - the backend product of cloud centers.

SaaS Software as a Service - Cloud application services, whereby applications are delivered over the Internet by the provider, so that the applications don't have to be purchased, installed, and run on the customer's computers. SaaS providers were previously referred to as ASP (application service providers). In the SaaS layer, the service provider hosts the software so you don't need to install it, manage it, or buy hardware for it. All you have to do is connect and use it. SaaS Examples include customer relationship management as a service.

Salesforce.com – An online SaaS company that is best known for delivering customer relationship management (CRM) software to organisations over the Internet.

Self-service provisioning – Cloud customers can provision cloud services without going through a lengthy process. You request an amount of computing, storage, software, process, or more from the service provider. After you use these resources, they can be automatically deprovisioned.

Service migration – The act of moving from one cloud service or vendor to another.

Service provider – The company or organization that provides a public or private cloud service.

Service level agreement SLA - A contractual agreement by which a service provider defines the level of service, responsibilities, priorities, and guarantees regarding availability, performance, and other aspects of the service.

Standardized interfaces – Cloud services should have standardized APIs, which provide instructions on how two application or data sources can communicate with each other. A standardized interface lets the customer more easily link cloud services together.

Subscription-based pricing model – A pricing model that lets customers pay a fee to use the service for a particular time period, often used for SaaS services. See also *Consumption-based pricing model*.

Use Case - In software and systems engineering, a use case [...] is a list of steps, typically defining interactions between a role (known in UML as an "actor") and a system, to achieve a goal. The actor can be a human or an external system. In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. The detailed requirements may then be captured in SysML or as contractual statements.' <http://en.wikipedia.org/wiki/Usecase>

Utility computing – Online computing or storage sold as a metered commercial service in a way similar to a public utility

Vendor lock-in – Dependency on the particular cloud vendor and difficulty moving from one cloud vendor to another due to lack of standardized protocols, APIs, data structures (schema), and service models.

Vertical cloud – A cloud computing environment that is optimized for use in a particular industry, such as health care or financial services.

Virtual Private Cloud (VPC) – A term coined by Reuven Cohen, CEO and founder of Enomaly. The term describes a concept that is similar to, and derived from, the familiar concept of a Virtual Private Network (VPN), but applied to cloud computing. It is the notion of turning a public cloud into a virtual private cloud, particularly in terms of security and the ability to create a VPC across components that are both within the cloud and external to it. e.g., the Amazon VPC that allows Amazon EC2 to connect to legacy infrastructure on an IPsec VPN.

Virtual private data center – Resources grouped according to specific business objectives.

Windows Live Services – Microsoft's cloud-based consumer applications, which include Windows Live Mail, Windows Live Photo Gallery, Windows Live Calendar, Windows Live Events, Windows Live Skydrive, Windows Live Spaces, Windows Live Messenger, Windows Live Writer, and Windows Live for Mobile.

Note: Most terms taken from <http://cloudtimes.org/glossary/>

B. PaaSage Concepts

Adapter - The Adapter deploys the candidate to one or more platforms. If it is predicted that the SLA will not be met and there are sufficient resources, it deploys the next candidate. If possible within available resources, it should trigger the Reasoner to generate new candidates within parameter constraints.

Application Controller - The application controller implements high-level management policies that need global knowledge or involve multiple cloud providers, such as policies involving cross-cloud migrations

Application Designer / Developer User– The Application designer / developer is a user who engages with the IDE to deploy an application to the Cloud.

Business Application User – The business application user is the domain expert who engages with the Cloud to fulfil business goals. Such an example is a Flight Scheduler who uses PaaSage to better route flights.

Component Instance - The component instance is the code part (application component/artefact/instance) that is treated as a single (black) box by the PaaSage system.

Component Wrapper - Invocations and messages calls reach the Component Wrapper before being relayed to the Component Instance. This way, the Executionware can get

full control over the Component Instance even when the environment does not allow such fine-grained control.

Cloud Modelling Language (Cloud ML) – A domain specific language used to describe Cloud topologies.

Execution Engine - The Enforcement Engine is the management entity of the Executionware. It captures the monitoring stream from all instances and matches it against the specification of the local scalability rules

Executionware - The Executionware manages the execution of deployment to platforms within encoded a) local platform ruleset and b) constraints from the Reasoner. The Executionware also monitors the execution and triggers the adapter (and hence Reasoner) if necessary.

Integrated Development Environment (IDE) - The IDE is the user point of contact in PaaSage presenting the main Cloud Modelling tools linked to the Profiler components.

Metadata Database(MDDB) - The MDDB comprises the metadata model and the implementation of the distributed physical store (which includes federation capabilities); the Analytics layer, providing support for a variety of analytics over historical metadata; and interfaces to the Profiler, Reasoner, Executionware, and Social network infrastructure components. The MDDB is meant for long-term preservation of information. It is designed to associate mutations with a wall-clock timestamp and to trace the identity of the sources of mutations.

Monitors - Monitors gather the relevant data directly at the component instances and relay the data further to the Enforcement Engine (and from there to the meta-data database).

Organisational User – Sets policies such as data protection that the business user and application designer/developer must abide by when using PaaSage.

Cloud Application Modelling Execution Language (CAMEL) - A language used to group domain specific languages in PaaSage into Models used to link lifecycle phases and express requirements during Cloud Configuration, Deployment and Execution.

Profiler - The Profiler characterises the application, via analysis of source code if available and with some input from developer/sysadmin.

-It will need a module to characterise the platforms, incl. querying platforms to update PaaSage database and further input from developer/sysadmin.

-Also requires a module to characterise data characteristics/dependencies.

-As well as some module to characterise user preferences, permissions and responsibilities.

Reasoner - The Reasoner provides ranked deployment candidates for ≥ 1 platform. This is based on:

- Application profile

-SLA parameters from this instantiation of the application supplied by the end user

-Platform characterisation

-User profile

-Data profile

Upperware - Upperware is a collection of tools and components to assist the porting of models at design-time.

IX. ANNEX3: THE CLOUD OFFERINGS SURVEYED

X. ANNEX3: THE CLOUD OFFERINGS SURVEYED

USA-based IaaS providers

Amazon AWS: <http://aws.amazon.com>

AT&T Cloud Architect: <http://cloudarchitect.att.com>

Bit Refinery: <http://bitrefinery.com>

GoGrid: <http://www.gogrid.com/>

Google Compute Engine: <https://cloud.google.com/products/compute-engine>

Hosting.com: <http://www.hosting.com>

HP Cloud: <https://www.hpcloud.com>

IBM SmartCloud Enterprise: <http://www.ibm.com/services/us/en/cloud-enterprise/index.html>

Microsoft Windows Azure: <http://www.windowsazure.com/en-us>

Nephoscale: <http://www.nephoscale.com>

OpSource: <http://www.opsource.net>

RackSpace: <http://www.rackspace.com>

ReliaCloud: <http://www.reliacloud.com/>

Softlayer: <http://www.softlayer.com/>

Terramark: <http://www.terremark.com/>

Europe-based IaaS providers

Aruba Cloud: <http://www.cloud.it>

CloudSigma: <http://www.cloudsigma.com>

Gandi: <https://www.gandi.net/>

GreenQloud: <http://greenqloud.com/>

Lunacloud: <http://www.lunacloud.com/>

Memset: <http://www.memset.com/>

IaaS benchmarks

CloudHarmony: <http://cloudharmony.com/>

CloudSleuth: <https://cloudsleuth.net/>

IaaS stacks

CloudStack: <http://incubator.apache.org/cloudstack/>

Eucalyptus: <http://www.eucalyptus.com/>

OpenNebula: <http://opennebula.org/>

OpenStack: <http://www.openstack.org/>

VMWare vCloud: <http://www.vmware.com/products/datacenter-virtualization/vcloud-suite/overview.html>

PaaS stacks

Apprenda: <http://apprenda.com/>

Cloud Foundry: <http://www.cloudfoundry.org/>

Cloudify: <http://www.cloudifysource.org/>

Iron Foundry: <http://www.ironfoundry.org/>

OpenShift: <https://openshift.redhat.com/>

Stackato: <http://www.activestate.com/stackato/>

IaaS/PaaS libraries

Deltacloud: <http://deltacloud.apache.org/>

fog: <http://fog.io/>

jclouds: <http://www.jclouds.org/>

Libcloud: <http://libcloud.apache.org/>

Simple Cloud: <http://simplecloud.org/>

PaaS frameworks

Aeolus: <http://www.aeolusproject.org/>

Chef: <http://www.opscode.com/chef/>

Cloud Foundry: <http://www.cloudfoundry.org/>

Cloudify: <http://www.cloudifysource.org/>

Cloudyn: <http://www.cloudyn.com/>

CopperEgg: <http://copperegg.com/>

enStratus: <http://www.enstratus.com/>

Flexiant: <http://www.flexiant.com/>

Juju: <https://juju.ubuntu.com/>

RightScale: <https://www.rightscale.com/>

Scalr: <http://scalr.com/>

Cloud standards

DMTF Cloud Management Standards: <http://dmtf.org/standards/cloud>

OASIS Cloud Application Management for Platforms (CAMP): <https://www.oasis-open.org/committees/camp/>

OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA): <https://www.oasis-open.org/committees/tosca/>

Open Cloud Computing Interface (OCCI): <http://occi-wg.org/>

SNIA Cloud Data Management Interface (CDMI): <http://www.snia.org/cdmi>

Cloud EU projects

4CaaS: <http://4caast.morfeo-project.org/>

ARTIST: <http://www.artist-project.eu/>

Broker@Cloud: <http://www.broker-cloud.eu/>

CELAR: <http://www.celarcloud.eu/>

Cloud-TM: <http://www.cloudtm.eu/>

Cloud4SOA: <http://www.cloud4soa.eu/>

CloudScale: <http://www.cloudscale-project.eu/>

Contrail: <http://contrail-project.eu/>

CumuloNimbo: <http://www.cumulonimbo.eu/>

MODAClouds: <http://www.modaclouds.eu/>

mOSAIC: <http://www.mosaic-cloud.eu/> ""

Optimis: <http://www.optimis-project.eu/>

REMICS: <http://remics.eu/>

Reservoir: <http://www.reservoir-fp7.eu/>

VISION Cloud: <http://www.visioncloud.eu/>

XI. BIBLIOGRAPHY

- [1] N. Ferry and et al, “Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems.,” in *CLOUD 2013: IEEE 6th International Conference on Cloud Computing*, 2013.
- [2] N. Ferry, F. Chauvel, A. Rossini, B. Morin and A. Solberg, “Managing multi-cloud systems with CloudMF,” in *2nd Symposium on Cloud Computing and Internet Technologies*, Oslo, 2013.
- [3] W. W. Rocy, “Managing the development of large software systems,” in *IEEE WESCON*, 1970.
- [4] MODAClouds, “Project homepage,” [Online]. Available: <http://modacLOUDS.eu>. [Accessed August 2013].
- [5] A. e. A. Ferrer, “OPTIMIS: A holistic approach to cloud service provisioning,” *Future Generation Computer Systems*, pp. 66-77, 2012.
- [6] “Cloud4SOA project homepage,” [Online]. Available: www.cloud4soa.eu. [Accessed 01 09 2013].
- [7] “Contrail Homepage,” [Online]. Available: contrail-project.eu. [Accessed 1 09 2013].
- [8] “cloudTM project homepage,” [Online]. Available: www.cloudtm.eu. [Accessed 01 09 2013].
- [9] “Artist Project Homepage,” [Online]. Available: www.artist-project.eu. [Accessed 13 09 2013].
- [10] “Mosaic Cloud Project homepage,” [Online]. Available: www.mosaic-cloud.eu. [Accessed 01 09 2013].
- [11] Microsoft, “Windows Azure,” [Online]. Available: <http://www.windowsazure.com>. [Accessed August 2013].
- [12] Microsoft, “Windows Azure Service Level Agreements,” [Online]. Available: <http://www.windowsazure.com/en-us/support/legal/sla/>. [Accessed August 2013].
- [13] Google, “Google App Engine,” [Online]. Available: <https://developers.google.com/appengine/>. [Accessed August 2013].
- [14] Google, “Google App Engine SLA,” [Online]. Available: <https://developers.google.com/appengine/sla>. [Accessed August 2013].
- [15] CloudBees., “CloudBees Api. Developer Resources.,” [Online]. Available: <http://wiki.cloudbees.com/bin/view/RUN/API>. [Accessed August 2013].

- [16] GoPivotal, “Cloud Foundry Website,” [Online]. Available: <http://cloudfoundry.com>. [Accessed August 2013].
- [17] Heroku, “Heroku Dev Centre Platform API,” [Online]. Available: <https://devcentre.heroku.com/categories/platform-api>. [Accessed August 2013].
- [18] Jelastec, “Welcome to the Jelastec Documentation,” [Online]. Available: <http://jelastec.com/docs>. [Accessed August 2013].
- [19] “Mosaic Project Homepage,” [Online]. Available: <http://www.mosaic-cloud.eu/>. [Accessed 01 09 2013].
- [20] Armstrong D and K. Djemame, “Armst Performance issues in clouds: an evaluation of virtual image propagation and I/O paravirtualization,” *The Computer Journal*, vol. 54 , no. 6, pp. 836-849., 2011.
- [21] E. Council, “DIRECTIVE 95/46/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL,” 24-05-95. [Online]. Available: <http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:EN:HTML>. [Accessed 11 09 2012].
- [22] “ElasticHosts,” [Online]. Available: www.elastichosts.com. [Accessed 13 09 2013].
- [23] D. Luenberger and Y. Yinyu, *Linear and Nonlinear Programming*, Springer 2008.
- [24] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, Springer 2008.
- [25] F. Graybill, *Theory and application of the linear model*, Duxbury Press , 1976.
- [26] W. J. Conover, *Practical nonparametric statistics*, John Wiley and Sons, 1999.
- [27] T. W. Anderson, *An introduction to multivariate statistical analysis*, Wiley-Interscience, 2003.
- [28] L. Ljung, *System identification: theory for the user*, Prentice-Hall, 1998.
- [29] R. E. Bellman, “A {Markovian} decision process,” *J. Math. Mech*, vol. 6, no. 5, 1957.
- [30] R. Sutton and A. Barto, *Reinforcement Learning*, MIT Press, 1998.
- [31] K. S. Narendra and M. A. Thathachar, *Learning Automata*, Prentice Hall 1989.
- [32] M. A. Thathachar and P. S. Sastry, *Networks of Learning Automata: Techniques for Online Stochastic Optimization*, Kluwer Academic, 2004.
- [33] G. Horn, “A vision for stochastic reasoner for autonomic cloud deployment,” in *Proc Second Nordic Symposium on Cloud Computing and Internet Technologies*, NY, NY, 2013.

- [34] D. Wolpert and W. Macready, “No free lunch theorems for optimization,” *IEEE Trans on Evol Comput*, vol. 1, no. 1, 1997.
- [35] E. K. Burke and G. Kendall, *Search Methodologies - Introductory Tutorials in Optimization and Decision Support Techniques*, Springer , 2005.
- [36] H. H. Hoos and T. Stutzle, *Stochastic local search foundations and applications*, San Francisco: Morgan Kaufmann, 2005.
- [37] K. Geihs, P. Barone, F. Eliassen and e. al, “A comprehensive solution for application-level adaptation,” *oftw. Pr. Exp.*, vol. 39, no. 4, pp. 385–422, 2009..
- [38] J. Floch and et al, “Using architecture models for runtime adaptability,” *IEEE Softw.*, vol. 23, no. 2, pp. 62–70, 2006..
- [39] F. Fleurey and A. Solberg, “A Domain Specific Modeling Language Supporting Specification, Simulation and Execution of Dynamic Adaptive Systems,” in *Model Driven Engineering Languages and Systems: Proceedings of the 12 International conference (MODELS 2009)*, .
- [40] C. Pappis and C. Siettos, “Fuzzy Reasoning’, in *Search Methodologies*,” no. Sprniger 2005.
- [41] INRIA, “SimGrid,” [Online]. Available: <http://simgrid.gforge.inria.fr/>. [Accessed 11 09 2013].
- [42] “Engage Project Homepage,” [Online]. Available: <http://www.engagedata.eu/>. [Accessed 01 09 2013].
- [43] OPTIMIS, “OPTIMIS Homepage,” [Online]. Available: <http://www.optimis-project.eu/>. [Accessed August 2013].