

**ARTIST**

**FP7-317859**



***Advanced software-based seRvice provisioning and  
migratIon of legacy Software***

---

---

**Deliverable D7.2.3**

**Cloud services modelling and performance analysis framework**

---

---

<b>Editor(s):</b>	Nunzio Andrea Galante, Gabriele Giammatteo
<b>Responsible Partner:</b>	ENG
<b>Status-Version:</b>	Final - v 0.6
<b>Date:</b>	31/03/2015
<b>Distribution level (CO, PU):</b>	PU

<b>Project Number:</b>	FP7-317859
<b>Project Title:</b>	ARTIST

<b>Title of Deliverable:</b>	Cloud services modelling and performance analysis framework
<b>Due Date of Delivery to the EC:</b>	31/03/2015

<b>Work Package responsible for the Deliverable:</b>	WP7 – Meta-modelling for target definition and cloud delivery
<b>Editor(s):</b>	Engineering Ingegneria Informatica S.p.A.
<b>Contributor(s):</b>	ATOS, Fraunhofer, ICCS, INRIA, TECNALIA, TUWIEN, SPIKES
<b>Reviewer(s):</b>	Alexander Bergmayr (TUWIEN)
<b>Approved by:</b>	All Partners
<b>Recommended/mandatory readers:</b>	WP5, WP7, WP9, WP10, WP11

<b>Abstract:</b>	This deliverable is threefold: aiming initially at the validation of stereotypes and creating and incorporating the modelling framework for describing IaaS/PaaS providers and their attributes in the ARTIST platform, and highlighting, designing and implementing the benchmarking of selected PaaS/IaaS, and designing and implementing an abstract solution for the availability measurements of cloud service.
<b>Keyword List:</b>	Meta-modelling, CloudML@ARTIST, Benchmarking, Availability, Model-updater
<b>Licensing information:</b>	The reported code will be licensed under open source with following licenses: - Benchmarking under Apache v2

	<ul style="list-style-type: none"><li>- 3ALib under Apache v2</li><li>- CloudML@ARTIST under EPL</li><li>- Model Updater under EPL</li></ul> <p>The document itself is delivered as a description for the European Commission about the released software, so it is not public.</p>
--	---

---

---

## Document Description

---

---

### Document Revision History

<i>Version</i>	<i>Date</i>	<i>Modifications Introduced</i>	
		<i>Modification Reason</i>	<i>Modified by</i>
v 0.1	26/02/2015	TOC	ENG
v 0.2	10/03/2015	Contributions integrated	ENG
v 0.3	13/03/2015	Contributions integrated	ICCS
v 0.4	13/03/2015	Contributions integrated	TECNALIA
v 0.5	17/03/2015	Contribution integrated	ICCS
v 0.6	31/03/2015	Final version	ENG



---

## Table of Contents

---

Table of Contents.....	5
Table of Figures .....	8
Table of Tables .....	10
Terms and abbreviations .....	11
Executive Summary .....	13
Introduction.....	15
1.1    About this deliverable .....	16
1.2    Document structure .....	16
1.3    Main innovations.....	16
1.3.1    CloudML meta-model.....	16
1.3.2    Benchmarking Suite.....	17
1.3.3    3Alib.....	17
1.4    Fitting into overall ARTIST architecture.....	18
1.4.1    Management of Cloud models and benchmarking results .....	19
1.4.2    Exploitation of cloud models in modernization assessment process.....	19
1.4.3    Usage of cloud models in the forward engineering process .....	21
1.4.4    Usage of cloud models in product testing and validation .....	22
1.4.5    Integration with the other Artist repository.....	22
2    CloudML@ARTIST Implementation.....	24
2.1    Functional Description .....	24
2.2    Technical Description .....	25
2.2.1    Meta-model Structure.....	25
2.2.2    UML Profiles Description.....	27
2.2.3    Technical Specification .....	28
3    CloudML@ARTIST Delivery and Usage.....	29
3.1    Package information.....	29
3.2    Installation instructions.....	29
3.3    Licensing information .....	35
3.4    Download .....	35
4    Benchmarking Tools Implementation .....	36
4.1    Functional Description .....	36

4.2	Technical Description .....	36
4.2.1	Prototype Architecture.....	36
4.2.2	Components Description.....	37
4.2.3	Technical Specification .....	45
4.2.4	Third-party benchmarking tools.....	45
4.2.5	Benchmarking experiments and related performance model population.....	53
5	Benchmarking Tools Delivery and Usage.....	56
5.1	Package information.....	56
5.2	Installation instructions.....	56
5.3	User Manual .....	57
5.4	Licensing information .....	57
5.5	Download .....	58
6	3ALib Implementation.....	59
6.1	Functional Description .....	59
6.2	Technical Description .....	59
6.2.1	Prototype Architecture.....	60
6.2.2	Components Description.....	60
6.2.3	Technical Specification .....	62
6.2.4	Availability experiments on Amazon AWS EC2.....	63
6.2.5	SLA metrics .....	65
7	3ALib Delivery and Usage.....	68
7.1	Package information.....	68
7.2	Installation instructions.....	68
7.3	User Manual .....	69
7.3.1	Configuration.....	69
7.3.2	Executions .....	71
7.4	Licensing information .....	72
7.5	Download .....	72
8	Model updater Implementation.....	73
8.1	Functional description.....	73
8.2	Technical description.....	74
8.2.1	Technical Specification .....	75
9	Model Updater Delivery and Usage .....	76
9.1	Package information.....	76

9.2	Installation instructions.....	76
9.3	User manual .....	76
9.4	Licensing information .....	77
9.5	Download .....	77
10	Conclusions .....	78
	References.....	79
	Appendix A – CloudML@Artist core profile .....	81
	IaaS .....	90
	PaaS .....	96
	SaaS .....	101
	Appendix B – Availability profile .....	105
	Appendix C – Performance Profile .....	112
	Appendix D – Cost profile.....	117
	Appendix E – Security profile .....	123
	Appendix F – Provider model example .....	124

---

## Table of Figures

---

FIGURE 1: OVERALL CLOUD MODEL GENERATION PROCESS .....	19
FIGURE 2: OVERALL METHODOLOGY FOR EVALUATING NON-FUNCTIONAL GOALS.....	22
FIGURE 3: HIERARCHICAL VIEW OF CLOUDML@ARTIST .....	25
FIGURE 4: CORE PROFILE.....	26
FIGURE 5: SPECIFIC PROVIDERS' PROFILES.....	26
FIGURE 6: SUPPORTING PROFILES.....	27
FIGURE 7: OVERALL BENCHMARKING SUITE ARCHITECTURE .....	37
FIGURE 8: SCENARIO FOR THE ETICS AND ARTIST REPOSITORIES .....	39
FIGURE 9: BENCHMARKING GUI FILTERING OF HISTORICAL DATA.....	40
FIGURE 10: RETRIEVAL OF INFORMATION AND DEPICTION.....	41
FIGURE 11: INSTANCETYPES FOR CLOUD PROVIDER .....	41
FIGURE 12: CLOUD TEST FOR DAcAPO .....	42
FIGURE 13: CLOUD TEST FOR FILEBENCH.....	42
FIGURE 14: CLOUD TEST FOR YCSB .....	42
FIGURE 15: GUI – CONTROLLER INTEGRATION.....	43
FIGURE 16: BENCHMARK DATABASE SCHEMA.....	44
FIGURE 17: EXAMPLE OF BENCHMARK DATABASE VIEW.....	44
FIGURE 18: EXAMPLE OF REST INTERFACE FOR THE DATABASE.....	45
FIGURE 19: YCSB ARCHITECTURE .....	46
FIGURE 20: YCSB OUTPUT EXAMPLE .....	47
FIGURE 21: DWARFS SCORES.....	48
FIGURE 22: DWARFS STATISTICS .....	48
FIGURE 23: INDIVIDUAL EXECUTION STATISTICS .....	48
FIGURE 24: CLOUDSUITE DATA CACHING OUTPUT EXAMPLE .....	49
FIGURE 25: FILEBENCH OUTPUT EXAMPLE .....	50
FIGURE 26: FINCoS FRAMEWORK ARCHITECTURE .....	51
FIGURE 27: FINCoS OUTPUT EXAMPLE .....	52
FIGURE 28: PERFORMANCE TIME IN MS FOR DAcAPO WORKLOADS .....	54
FIGURE 29: BENCHMARK PROFILE IS APPLIED TO THE DIAGRAM.....	54
FIGURE 30: BENCHMARK PROFILE IS APPLIED TO THE AMAZON EC2 PROFILE.....	55
FIGURE 31: INSTALLATION PROCEDURE OF BENCHMARKING TOOLS .....	57
FIGURE 32: 3ALib GENERAL DESIGN.....	60
FIGURE 33: 3ALib CLASS DIAGRAM .....	61
FIGURE 34: SAMPLE INTERARRIVAL TIME PROBABILITY DISTRIBUTION.....	64
FIGURE 35: 3ALib BACK-END DB VIEW .....	69
FIGURE 36: 3ALib CONFIGURATION FILE .....	69
FIGURE 37: 3ALib AVAILABILITY CALCULATION INTERVAL DEFINITION.....	72
FIGURE 38: MODEL UPDATER PROCESS.....	73
FIGURE 39: ABSTRACT CLASS BENCHMARKRESULTS.....	74
FIGURE 40: MODEL UPDATER PACKAGES.....	76
FIGURE 41: TXT FILE SAMPLE .....	76
FIGURE 42: CLOUDML@ARTIST'S HIGH LEVEL STEREOTYPES.....	82
FIGURE 43: CLOUDML@ARTIST CORE DATA TYPES .....	89
FIGURE 44: IAAS STEREOTYPES .....	91
FIGURE 45: PAAS STEREOTYPES.....	98
FIGURE 46: SOFTWARESERVICE STEREOTYPE.....	102
FIGURE 47: AVAILABILITY MEASUREMENT STEREOTYPES.....	105
FIGURE 48: GENERAL INFORMATION OF SLA DEFINITION REGARDING PAAS, IAAS AND STORAGE	

PROVIDERS .....	107
FIGURE 49: ILLUSTRATION OF DISCOUNT TYPE AND DIFFERENT ENUMERATION FIELDS THAT COMPLETE THE SLA DEFINITION MODEL .....	109
FIGURE 50: ILLUSTRATION OF THE BASIC PART OF PERFORMANCE MODEL AND THE OCL CONSTRAINT .....	112
FIGURE 51: BENCHMARK RESULTS INCLUDED IN PERFORMANCE PROFILE .....	115
FIGURE 52: ILLUSTRATION OF THE UNIVERSAL ENUMERATION WITH THE DEFAULT WORKLOADS .....	115
FIGURE 53: COST PROFILE STEREOTYPES .....	118
FIGURE 54: WINDOWS AZURE MODEL MAIN STEREOTYPES .....	125
FIGURE 55: WINDOWS AZURE INSTANCE TYPES.....	128

Table of Tables

TABLE 1: VM INSTANCE CHARACTERISTICS..... 53

TABLE 2: BENCHMARKING SUITE LICENSES ..... 58

TABLE 3: AVAILABILITY EXPERIMENT STATISTICS..... 63

TABLE 4: SAMPLE STATISTICS FOR THE AMAZON SLA EXPERIMENT (US EAST) ..... 64

TABLE 5: INDICATIVE APPLICATION OF THE SLA STRICTNESS METRIC ON EXISTING PUBLIC CLOUD  
SLAs..... 67

TABLE 6: 3ALIB ACCEPTED PROVIDER NAMES ..... 70

---

## Terms and abbreviations

---

AGPL	Affero General Public License
APT	Advanced Packaging Tool
ARTIST	Advanced software-based seRvice provisioning and migraTion of legacy SoftWare
AWS	Amazon Web Service
BSD	Berkeley Software Distribution
CDDL	Common Development and Distribution License
CloudML	Cloud Modelling Language
CPU	Central Processing Unit
DB	DataBase
DRAM	Dynamic Random Access Memory
EC	European Commission
EPL	Eclipse Public License
ESB	Enterprise Service Bus
ETICS	eInfrastructure for Testing, Integration and Configuration of Software
GAE	Google App Engine
GNU	GNU's Not Unix
GPL	General Public License
GUI	Graphical User Interface
I/O	Input/Output
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
noSql	Not Only SQL
OS	Operating System
PaaS	Platform as a Service

PSM	Platform-Specific Model
R/W	Read/Write
RAM	Random Access Memory
RDBMS	Relational DataBase Management System
REST	REpresentational State Transfer
RG	Research Group
RPM	Redhat Package Manager
SaaS	Software as a Service
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SQL	Structured Query language
SPEC	Standard Performance Evaluation Corporation
UML	Unified Modelling Language
VM	Virtual Machine
WG	Working Group
WML	Workload Model Language
WP	Work Package
XML	eXtensible Markup Language
XMI	XML Metadata Interchange
YCSB	Yahoo! Cloud Serving Benchmark
YUM	Yellow dog Updater, Modified



## Executive Summary

This document is associated to four prototypes delivered in Work Package 7. The prototypes aim at providing a modelling and performance analysis framework to support the creation of cloud models and to measure performance or availability aspects of cloud services. Implementation and usage instructions details are included in this document.

The first prototype is a meta-model that will be used as baseline for creating models of cloud models (Section 2). The meta-model defines the concepts and relationships that describe the main capabilities of resources offered by cloud platforms. The meta-model is realized as an extension to the UML meta-model. Therefore, it has been realized in terms of a profile / collection of profiles regarding specific aspects (e.g. Availability concepts –Appendix B, Performance Concepts, Appendix C etc.). Profiles created starting from this meta-model (Appendix F) will be used in ARTIST during the migration of an application in order to select the target platform that matches best the requirements and functionalities needed by the re-engineered application.

The second prototype consists of a software suite for benchmarking cloud resources in order to extract performance-related data and to include it in the cloud models. Since performance aspects are considered as key criteria that are taken into consideration in the selection of the target platform, the availability of this data in the cloud models simplifies and makes more accurate the migration phase. The software suite includes an installation, configuration and execution tool, incorporating a set of third-party benchmarking tools specifically selected for their effectiveness in evaluating cloud resources performances, a database for the data storage and a user interface to automate the management of executions and results of tests. It also includes a GUI for end users to visualize the results of the performance experiments.

The third prototype consists of an abstracted software library for the measurements of availability of services regardless of the supported provider on which they are deployed. The library may measure availability based on the respective providers SLA definitions and thus conclude for potential SLA violations, including also the evidence of this violation. It can also provide recommendations as to whether a specific service instance is viable for an SLA based on the provider's preconditions.

The fourth prototype consists of a tool to update the provider models/profiles that are stored in the repository with new results coming from the benchmarking experiments. While the overall data of the experiments are kept in an internal Raw Data DB and not stored in the provider profiles in order to keep the latter lightweight, the average performance information contained in them needs to be periodically updated based on the new range of measurements conducted each time.

Furthermore, In the third year of the project, we focused also in the integration aspects and better coordination of the individual outcomes of each tool, in order to offer a more unified experience to the tool users. This integration covered:

- Raw performance data exposed through a GUI for configurable queries
- Provisioning of the results to WP11, in the format needed by the GMF (more info on this can be found in D11.3.2)
- Linking of the profiles with the update in the experiments results
- A detailed experiment on the usage of 3ALib on a 1.5 month usage of AWS

- Extension of 3ALib to support Google Compute Engine SLAs
- A definition of a new metric for SLA strictness
- Inclusion of SaaS instances for billing and monitoring
- Integration of the provider models in the Cloud Target Selection process in WP9

## Introduction

The main objective of Work Package 7 is to provide the instruments for modelling target cloud environments where migrated applications will be deployed. Cloud models will be used in other work packages as inputs for processes, such as the business and technical feasibility analysis (Work Package 5), forward engineering (Work Package 9), and migrated application testing and validation (Work Package 11). Cloud models must capture all the characteristics and data of cloud providers needed by these other processes in the project. Requirements concerning the type and the format of information needed in cloud models have been investigated and elicited in project deliverable “D7.2 - PaaS/IaaS Meta-modelling Requirements and SOTA” [1] thanks to a close collaboration with the other technical work packages.

Following the Model Driven Architecture (MDA) approach, Work Package 7 aims at defining a meta-model that specifies all the concepts and relationships of interest when modelling a cloud provider. This meta-model is called CloudML@ARTIST since it is directly derived from a continuation of the work started in the REMICS project. In particular, the CloudML@ARTIST meta-model contains some concepts not covered by the original CloudML; in such sense we have followed an iterative / incremental approach and significantly improved CloudML in the second year of the project. In particular:

- PaaS and SaaS offering: the original CloudML only focuses on resources at IaaS level.
- performance and monitoring aspects: they are not taken in consideration in the CloudML meta-model, but needed in ARTIST to make decisions about the best target environment for the migration and the evaluation of the effectiveness of the migration.
- other aspects like pricing, scalability, availability, that are of interest in ARTIST for the non-technical evaluation of the migration process (e.g., the business feasibility).

The second prototype released by Work Package 7 is a Benchmarking Suite with the main objective of standardizing and automating as much as possible the gathering and modelling of performance data of cloud resources. The Benchmarking Suite is composed of four main components:

- a set of selected third-party benchmarking tools that have been packaged and made available in the ARTIST Repository ready to be executed on target cloud environments.
- a repository for results that is part of the ARTIST Repository where results for all executions are stored for later consultation and querying.
- a GUI that is used for configuring and submitting benchmark tests (automatically, when possible) on remote cloud infrastructure and access results.
- a controller component that will orchestrate the execution of benchmarks and manage the collection of results from tools output and publication on the ARTIST Repository.

The third prototype released by Work Package 7 is the 3Alib software; the main objective of such library is to measure the availability of services using an abstracted approach, regardless of the supported provider on which they are deployed. Such library will be able to calculate the actual levels of availability as defined in the respective providers SLA. The goals of the library are:

- consult users on why their services do not adhere to the SLA (if applicable)
- keep detailed logs of the availability of services

- calculate availability levels based on the providers definition and be used for compensation claims (together with the aforementioned logs)

The fourth prototype is a Model updater tool, which is used in order to periodically refresh the average performance values that are included in the provider models. In order to keep the latter lightweight, it was decided that the raw data extracted from the measurements should be stored in an internal (WP7) DB, while their averages would be included in the actual profiles. Thus this prototype is responsible for:

- calculating the average values from the raw data DB
- retrieving the provider models from the repository, updating their values and restoring them in the latter, so that they are available to the overall ARTIST tool chain with the most up-to-date information.

## 1.1 About this deliverable

This document is delivered along with software prototypes released in Work Package 7 at the end of the third year. It describes the main functional and technical aspects of the three prototypes released: a) the CloudML@ARTIST meta-model, b) the Benchmarking Suite c) 3alib library.

## 1.2 Document structure

The remainder of this document is structured as follow:

- Section 2 and 3 explain respectively the implementation of CloudML@ARTIST and how it fits with the other tools and processes of ARTIST. Furthermore, the information is given about related delivered packages in terms of installation instructions, licensing and download locations.
- Section 4 and 5 describe respectively the Benchmarking Suite tools implementation and usage.
- Section 6 and 7 describe respectively the 3Alib library implementation and usage.
- Section 8 and 9 describe the Model Updater implementation and usage.
- Appendixes contain the main profile introduced in the CloudML@ARTIST meta-model and an example of how it can be applied to describe Windows Azure

## 1.3 Main innovations

### 1.3.1 CloudML meta-model

CloudML@Artist meta-model has been created in order to respond to two existing needs at ARTIST project level. On one side, modelling cloud providers in order to select the most appropriate, depending on the specific service needs of an application. On the other side, modelling the deployment characteristics and values of an application in a concrete cloud provider, once it has been selected.

To accomplish this, CloudML@Artist is composed of an extensible set of UML profiles through which it is possible to model the main features of cloud providers, as well as modelling capabilities to describe services that can be offered under the saas model. These same profiles can be used also to specify values related to concrete deployments.

Despite having many aspects in common with CloudML meta-model created in the context of MODAClouds, PaaSage and REMICS FP7 projects for multi-cloud systems, CloudML@Artist presents some innovative elements that extend the capacities offered by CloudML.

The most relevant innovative elements are related to the capacity offered by CloudML@Artist to model the SLA offered by providers in terms of availability and performance. Furthermore, the metamodel defines the necessary structure, in order to directly insert performance information of Cloud offerings in a variety of application types, through the use of relevant benchmarks. This structure includes also the workload under which the test was performed. In addition, placeholders have been included in order to keep historical information on availability statistics of the Cloud offerings, either in the form of generic availability measurements (e.g. performed through the CloudSleuth web site) or adapted to each provider's definition, which is an indication of how frequently SLAs are violated by each respective provider for the specific offering.

### **1.3.2 Benchmarking Suite**

The most innovative aspect of the Benchmarking Suite is in the management of the benchmarking process. In fact, the Benchmarking Suite is a fully automated solution that, through connectors, is able to provision cloud resources, install and execute benchmarking tools, retrieve results and, eventually, destroy provisioned resources.

Compared to other available benchmarking solutions for Cloud, the Artist Benchmarking Suite drastically reduces the manual intervention in the benchmarking process allowing massive and large-scale tests.

Such suite allows carrying out tests and getting performance metrics in a homogeneous and independent way (without regards to those published by the providers). Metrics can then be used to collect, over the time, quantitative information about the performances offered and constitute the baseline for the selection of Cloud services during the optimization of the whole business process.

These results will be eventually directly injected into the Artist models representing providers, thereby enriching the modelling capabilities of the meta-model and covering a significant lack existing until now.

### **1.3.3 3ALib**

The Abstracted Availability Auditor Library (3ALib) innovations can be summarized as follows:

- It is an abstracted library, meaning that a single method is exposed to the user to utilize for measuring the SLA levels of their services independent from the used provider, while hiding in the backend the differences between provider APIs, through the use of Apache JClouds library. This enables its ease of use.
- It adapts to each provider's definition of the SLA, a very important aspect for someone to claim compensation from the respective provider. These definitions have significant differences and preconditions, that are checked per provider in the library, through the implementation of respective drivers.
- It logs the availability samples that are needed as forensic evidence by the providers for the acknowledgement of the violation and calculates the availability levels (according to the provider's definition) from the samples.
- It can instruct users as to whether their current usage/application deployment of Cloud services is actually eligible for an SLA, and can consult on what is needed for this to be realized

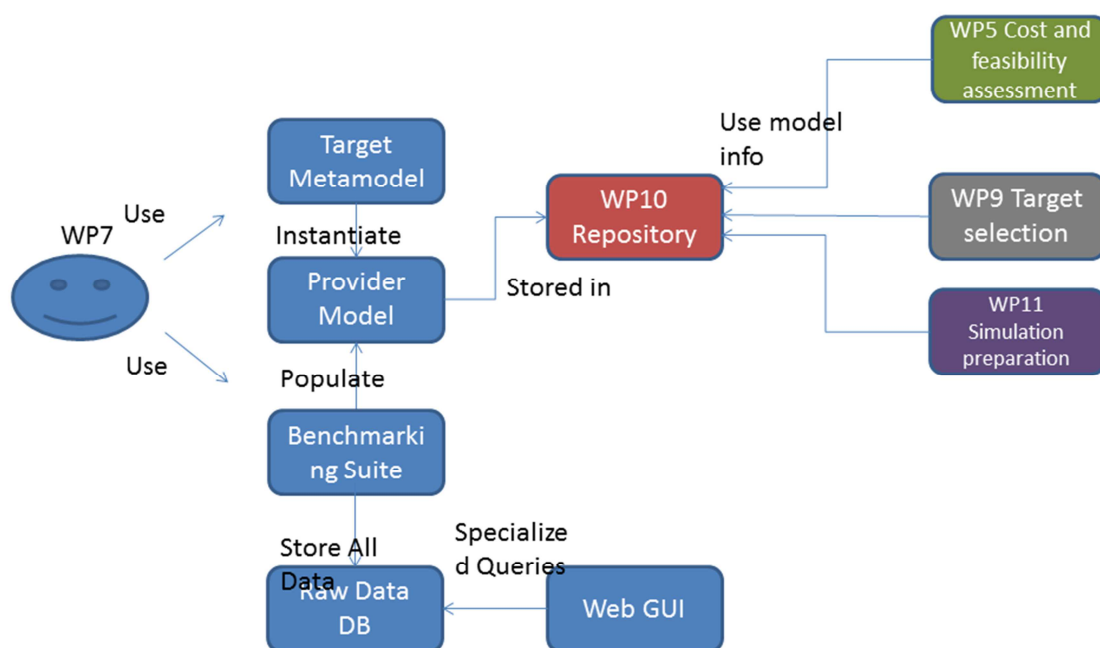
- It can be used to centrally gather statistics from multiple users regarding the behaviour of Cloud services, thus enabling accumulated statistics and potentially exploitation scenarios through Business Intelligence analytics.

## 1.4 Fitting into overall ARTIST architecture

The generation of the Cloud models is expected to be a semi-automatic process. In the beginning, in order to create a model from the provided meta-model, one should use the meta-model definition and select/populate the different required and optional fields.

In the context of Work Package 7, we have created such models for the most popular Cloud providers like Amazon, Azure and GAE.

Finally, the performance aspects of the models can be acquired through the benchmarking suite provided also by Work Package 7. This benchmarking suite will be responsible for installing the necessary benchmarks on the target platforms. Tests can be launched and results will be automatically stored in the model that will be available through the ARTIST Repository. The overall cloud model generation process is depicted in **Figure 1.4: Cloud model generation process**. WP7 will define the Cloud target metamodel (CloudML@ARTIST) and will use it to instantiate the provider models, in order to describe services and features. Furthermore, information from the Benchmarking Suite will be used to populate the performance fields of the provider models. The detailed results will also be directed towards a central DB, from which users will be able to query based on their interests, by using a Web GUI. Models will be included also in the ARTIST Repository, in order to be considered in the related Work Packages of ARTIST such as Work Package 5 for the initial estimation (e.g. by exploiting elements of cost), Work Package 9 for the final selection of the offering based on all the needed criteria (e.g., constraints, performance attributes) and Work Package 11 for the model-level validation and testing of the migrated application.



**Figure 1: Overall cloud model generation process**

### 1.4.1 Management of Cloud models and benchmarking results

During an ARTIST migration project a great number of different artifacts, such as provider descriptions, benchmarking tools or benchmarking results are used and produced. The ARTIST repository provides a central place to keep and organize these artifacts thus keeping the different work products manageable. In addition to storing the artifacts content the ARTIST repository can enrich them with meta-data that can be used to further describe them and to organize and find artefacts more efficiently. A benchmarking result set can for example be stored together with information about the used benchmarking tool, the platform model and used parameters.

In the ARTIST repository, artefacts are organized in collections and packages on a physical level. Projects can store their artefacts in their own dedicated collection and configure the access rights to fit their privacy needs. There is one special collection for the publicly available artefacts that are featured on the ARTIST Marketplace. Inside a collection the artefacts are organized in packages. Artefacts can be addressed by their physical location using a URI of the form

```
aar://[collectionName]/[packageName]/[ArtefactName]
```

Benchmark results for the ATOS use case could, for example, be stored using the following pattern:

```
aar://UseCaseATOS/benchmarks.[providerName].[toolName]/result-[timestamp]
```

Artefacts can also be addressed using a freely assignable logical URI independent of the organization of the repository. This logical URI has to be unique in the repository. A logical URI pattern for the benchmarking results of the previous example could be

```
http://www.artist-project.eu/benchmark/result/<provider>/<tool>/<timestamp>
```

In addition to the actual document and additional meta-data can be stored alongside the artefact.

The ARTIST repository can be accessed from an Eclipse workspace via a repository client plug-in. Most of the functionality of the repository will also be available via a web service interface.

### 1.4.2 Exploitation of cloud models in modernization assessment process

The modernization assessment process consists of three main steps:

1. **Maturity assessment.** The objective of this step is to analyse how mature the application is in terms of technology and business and how the customer wants the application to be in those two axes once the application is migrated. The evaluation of the current situation and the ideal situation allows ARTIST to perform a gap analysis, described in terms of a technical feasibility analysis and the business feasibility analysis.
2. **Technical Feasibility Analysis.** This analysis aims to provide a snapshot of how well / bad designed an application is, how complex and tightly coupled it is, how interrelated

etc., with the main objective of having a high level design of the application via Reverse Engineering techniques and a static analysis of the source code. These data in combination with the ideal technological maturity identified in the maturity assessment as well as with the target platform requirements will provide some metrics such as how much effort will be needed to perform this migration and how complex it will be.

3. **Business Feasibility Analysis.** Based on the technical feasibility and the results from the ideal situation identified in the maturity assessment and the identification of the characteristics that the target platform shall have, a business feasibility analysis is performed. This business feasibility analysis is aimed to provide not only economic information such as ROI, or payback, but also which are the main risks to be faced with the migration and the organizational processes affected by the uptake of the new business model.

Two of this steps: the *Technical Feasibility Analysis* and the *Business Feasibility Analysis*, the information related to the target description will be used as input to the process.

In the **Technical Feasibility Analysis**, target platform requirements are needed in order to propose the correct migration task for each component, propose changes in the component model and estimate the complexity required to carry out the migration project into one concrete platform, or to be able to propose the target platform that fulfils the migration goals. More concretely it is envisioned to require at least the following data from the target platform model:

4. **Database type:** In order to check whether or which of the target platforms fulfils the user requirements expressed through the migration goals (e.g., multi-instance RDBMS, multi-tenant RDBMS, NoSQL RDBMS). This information will be also useful to determine the complexity of migrating from the current database technology to the one required by the user considering the database technology/ies supported by the target platform.
5. **Multi-tenancy support:** In order to check if or which of the target platforms fulfils the user requirements expressed through the migration goals (e.g., One-Middleware/One-instance, One-Middleware/Several-instances, virtualization by tenant). This information will be also useful to determine the complexity of achieving the required multi-tenancy level considering the multi-tenancy level supported by the target platform.
6. **SLA requirements & performance indicators:** The *Technical Feasibility Analysis* will select the most appropriate target platform (if the decision is not already made by the user) or will check if the selected target platform fulfils the user requirements.
7. **Authorization services:** in the *Maturity Assessment* the user will express her needs in terms of authorization mechanisms for the migrated application. In the technical feasibility analysis this information will be considered in order to calculate the complexity of supporting such mechanisms by developing a dedicated new component or using a service provided by the target platform. The proposed possible authorization mechanisms are: SAML, OpenID, others.

In the **Business Feasibility Analysis** the target platform information will be used to determine the most suitable business model, to check if the selected platform (if it is already decided by the user) accomplish the business related migration goals, or to propose the most adequate target platform that supports the license model and the business model that the user is willing to offer to their customers. The envisioned required information from the target platform to the Business Feasibility Analysis is:



- **Billing information** of the target platform: The Business Feasibility Analysis will make some calculations to provide estimations of the Cost Benefit Analysis. In order to perform this analysis information about the cost of the service.
- **Pricing model** supported by the target platform: The user expresses her needs and wishes in the maturity analysis and the target platform should be able to support them. In the context of the Business Feasibility Analysis the more convenient target platform (in terms of supporting the business model required by the user) will be selected or it will be checked if the target platform already selected by the user supports user's requirements. For this, the following information is envisioned to be required from the target platform:
  - provided resources measurement models (i.e., time, volume, sessions, requests, users, others).
  - Supported billing rules (i.e., flat rate, cumulus rating, by use, by use + minimum, effective bandwidth pricing, cumulus pricing, responsive pricing, others).
  - Supported license models (i.e., by subscription, by use, by transaction, by value, flat rate, hybrid, others).

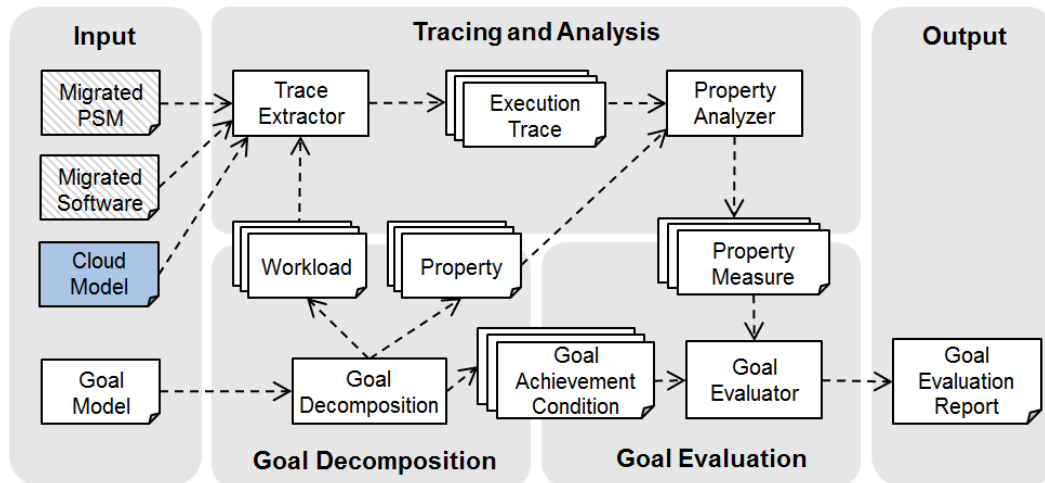
### 1.4.3 Usage of cloud models in the forward engineering process

Several tasks in the forward engineering process provided by WP9 rely on the information provided by this deliverable. In particular, the knowledge about the functional and non-functional properties of Cloud services is crucial for migrating legacy models produced by components of WP8 to Cloud-based applications. Here we would like to highlight three tasks that are the cornerstones of the modernization process.

*Manual service selection.* The first basic step when migrating legacy models to the Cloud is to select manually the services that are desired for the operation of the system on the Platform-Independent Modelling (PIM) level. In this respect, one may select generic services (that are generalizations of common Cloud services) in a first step, and subsequently, refine these selected generic services by finding the most appropriate offered services of Cloud providers. Thus, a catalogue of services offered by Cloud providers is needed to generalize the common services as well as to document the specific realizations and their associated quality.

*Automated service selection.* In order to provide a higher level of automation in the forward engineering process, instead of manually selecting abstract services and refining them to concrete services, requirements may be specified that should be fulfilled by the systems or by components. These requirements are the main driver for guiding architectural redesign decisions and are the foundation for a match-making process for computing the best fitting set of services to fulfil these given requirements. By this, the user does not have to define how the system should be realized; instead she can focus on what should be realized. Thus, the functional and non-functional properties of the services have to be defined in a format that is automatically processable.

*Deployment optimization.* Once a set of services is selected, additional configurations and deployment options may be automatically examined in order to improve performance measures or costs. Current Cloud platforms offer a huge set of different options that leads to an overwhelming search space for the user. Thus, automation support is highly needed for this task. This may be achieved by pure static analysis of the system and the used services, e.g., by analysing the models' structures. An alternative approach is to have dynamic analysis in terms of simulations to determine the best configurations and deployments. Thus, in order to achieve a meaningful and automated deployment space exploration process, detailed information of



the non-functional properties of the offered Cloud services is needed on the model level in order to use this information in static analysis algorithms or simulators. [2] [3]

**Figure 2: Overall methodology for evaluating non-functional goals**

#### 1.4.4 Usage of cloud models in product testing and validation

The task of testing and validation in Work Package 11 is to evaluate whether the specified functional and non-functional goals of the migration have been achieved. Functional goals are implicitly given by the legacy software, whose observable functional behaviour should not change during the migration. These goals are evaluated before the deployment by generating test cases from the migrated model and after the deployment via end user-based testing.

Non-functional goals, i.e., goals that are concerned with quality aspects of the software, such as performance, are collected from the user with the maturity assessment tool during the pre-migration phase and stored in a so called goal model. In general, non-functional goals are evaluated for a given workload by extracting the respective execution traces from the executed migrated model or from monitoring the deployed migrated software, as depicted in Figure 2.

Model-level validation can be done before the actual deployment and thus enables to avoid costly deployments that will most probably not fulfil the migration goals set by the user. This is done by simulating the behaviour of the migrated software based on the migrated models. For estimating the expected non-functional properties by model simulation as accurately as possible, the cloud model will serve as valuable input regarding several characteristics of the underlying cloud platform (e.g., hardware characteristics, scaling strategies, etc.). Therefore, cloud models can be used to obtain more accurate measures for certain metrics (e.g., execution time of a specific operation on a given platform) by providing detailed descriptions of the platform or infrastructure used for the deployment. This is especially true for the aspects concerning cloud resources, the physical infrastructure or scalability metrics.

Furthermore, the 3ALib library may also be used directly for measuring a specific deployment's availability service levels.

#### 1.4.5 Integration with the other Artist repository

In order to finalize the integration between the raw data DB and the provider models in the repository, an extra step is necessary, through the Model Updater tool. The purpose of this is to

be run periodically and update the performance values that exist in the provider models.

Given that the models should be as lightweight as possible, it was decided to provide in them information regarding the average values of a provider's service type (e.g. VM size type) on the higher possible level of benchmark abstraction. Thus the categories in this case would be as generic as:

- Java benchmarks average score
- Filesystem benchmarks average score
- Databases benchmarks average score

If a user would need more fine grained information (e.g. Java benchmark on a specific workload, for a specific period in time), then they should forward their request through the Raw Data Db in Figure 1.

## 2 CloudML@ARTIST Implementation

### 2.1 Functional Description

In the context of the ARTIST project, and following the analysis on the state of the art documented in the ARTIST Deliverable D7.2 [1], it was decided to base our modelling framework on the CloudML [4] definition, thus extending it to cover aspects specifically investigated in the project (e.g. performance) or better describe service offerings on different levels (PaaS and SaaS).

A gap that was identified in the aforementioned document was the lack of adequate description frameworks for capturing performance characteristics of cloud services and resources. For example, for CPU resources typical descriptions (like in CloudML) include only number and frequency of CPU cores. However this is far from sufficient for accurately describing the actual performance of a computing resource. Furthermore, fluctuation in the actual output of these services due to cloud environment issues (e.g. noisy neighbour effect, multi-tenancy, migration) is a severe aspect that has begun to take notice in the cloud users.

Except for these measurable aspects, a variety of services is available in the cloud market with different characteristics, costs and purposes. Thus an adequately rich description model should be in place to address these aspects (e.g. different storage services for archiving, fast retrieval, low cost etc.).

A number of supporting services may exist that can indicate the flexibility for managing cloud resources and should be included in order for a developer to make an informed decision of the suitable cloud provider/service, based on her interests and intentions for manageability. For example, billing alerts, advanced monitoring, ability to project into the future, automated scalability rules are characteristics that can be helpful to an application developer who intends to migrate an application to the cloud but may or may not be needed for a specific use case. By having this information the application developer may select in an optimal manner the target platform, based for example on different aspects such as cost or richness of supporting services.

The conceptual structure and hierarchical division of the CloudML@ARTIST is depicted in [1]. The main point to be aware is the *Multitypes* part. This contains attributes that can be included in more than one \*aaS levels. In order to reduce schema size and provider descriptions, these characteristics may be divided in two categories:

1. attributes that are generic with respect to the provider and do not need to be repeated in every offering of the latter (e.g., billing granularity etc.).
2. attributes that may be different in form or content for each service offering and must be included in each case (e.g., cost of the specific offering).

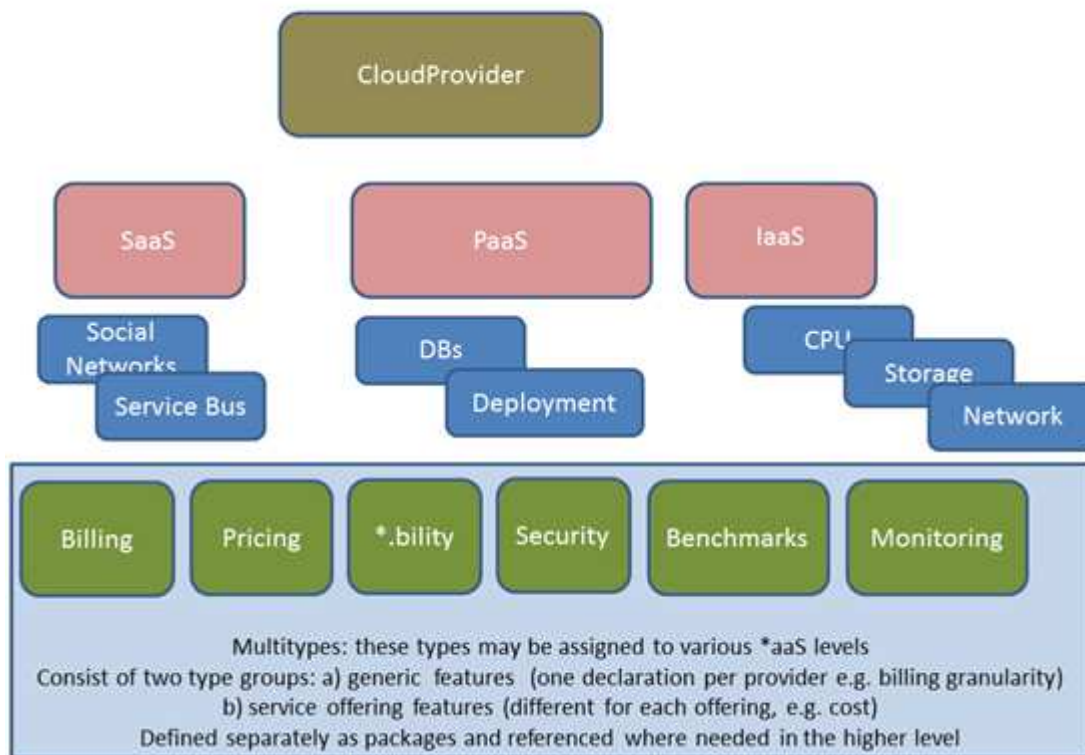


Figure 3: Hierarchical view of CloudML@ARTIST

For easier management, these types will be defined in separate sub-packages and will be referenced in the basic part of the meta-model.

## 2.2 Technical Description

**CloudML@ARTIST** has been created as a set of UML profiles able to cover the modelling needs both at WP7 and WP9 level.

At WP7 level **CloudML@ARTIST** should be focused at creating profiles for each provider under study in order to select the optimal one for migration depending on the characteristics of the legacy application

At WP9 level **CloudML@ARTIST** (CAML) should allow to enrich the PIM (Platform Independent Model) generated by reverse engineering techniques (WP8). As a result, a Platform Specific Model (PSM) will be generated in order to allow making semi-automatic deployments of the legacy application to the selected provider by applying M2T transformations.

### 2.2.1 Meta-model Structure

Next figures describe how **Cloudml@Artist** is structured as a series of interconnected UML profiles, making possible to create models with great flexibility.

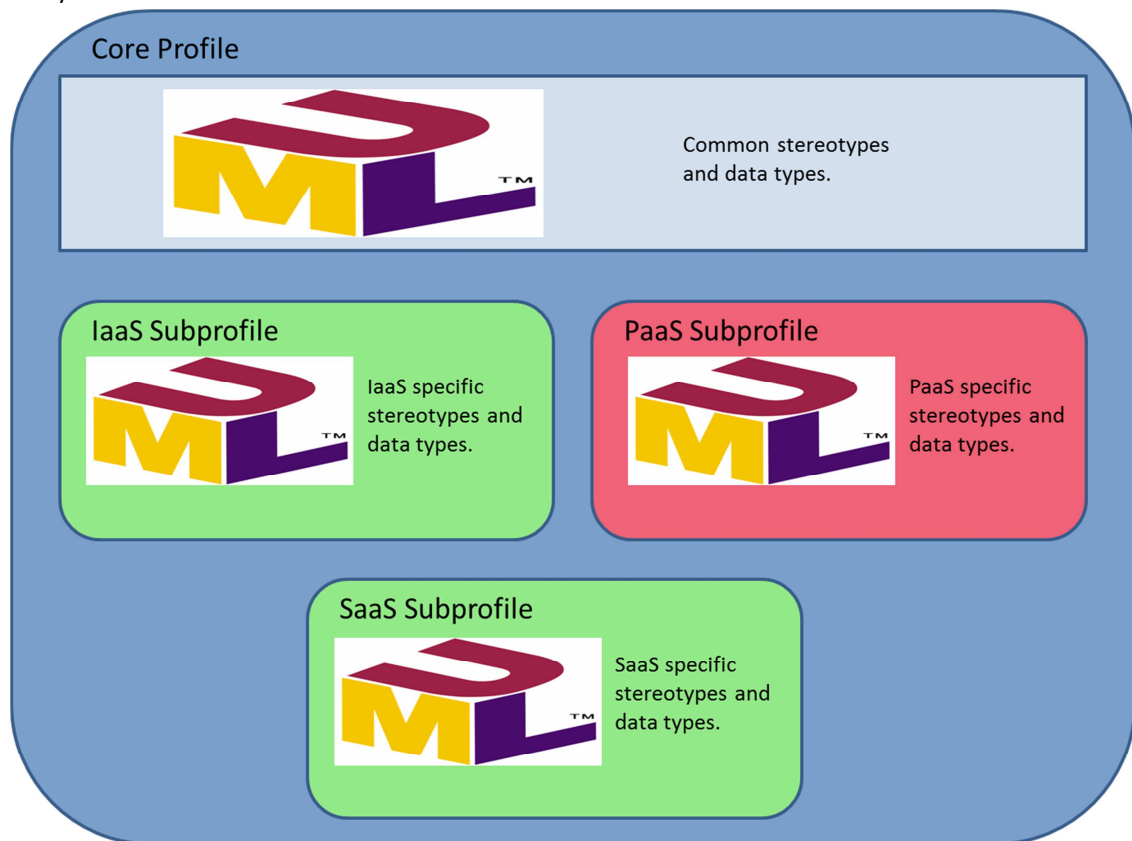


Figure 4: Core Profile

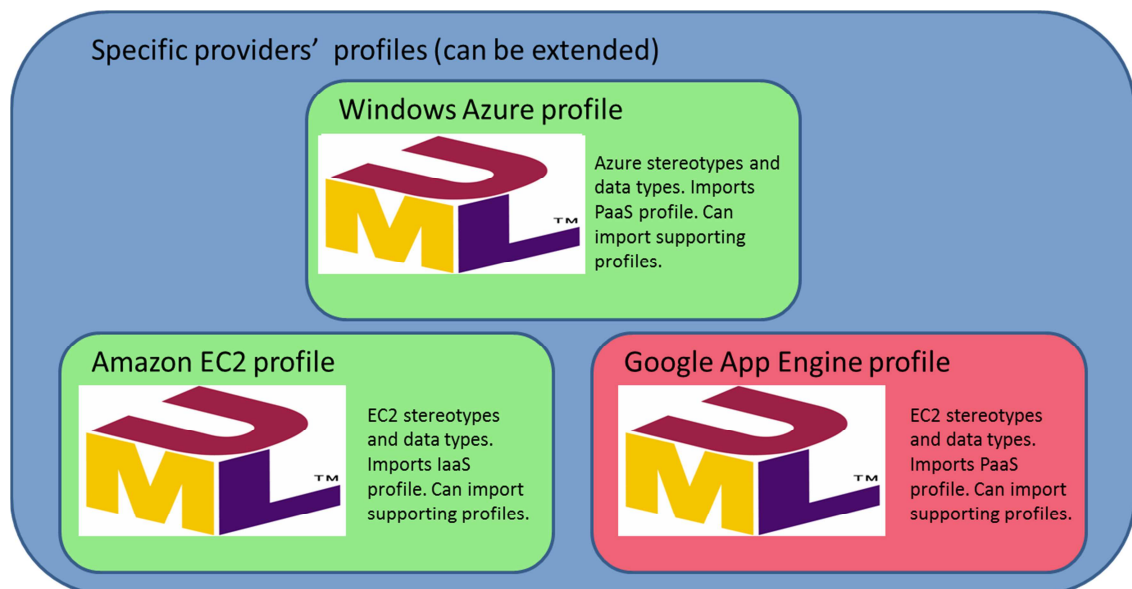


Figure 5: Specific providers' profiles

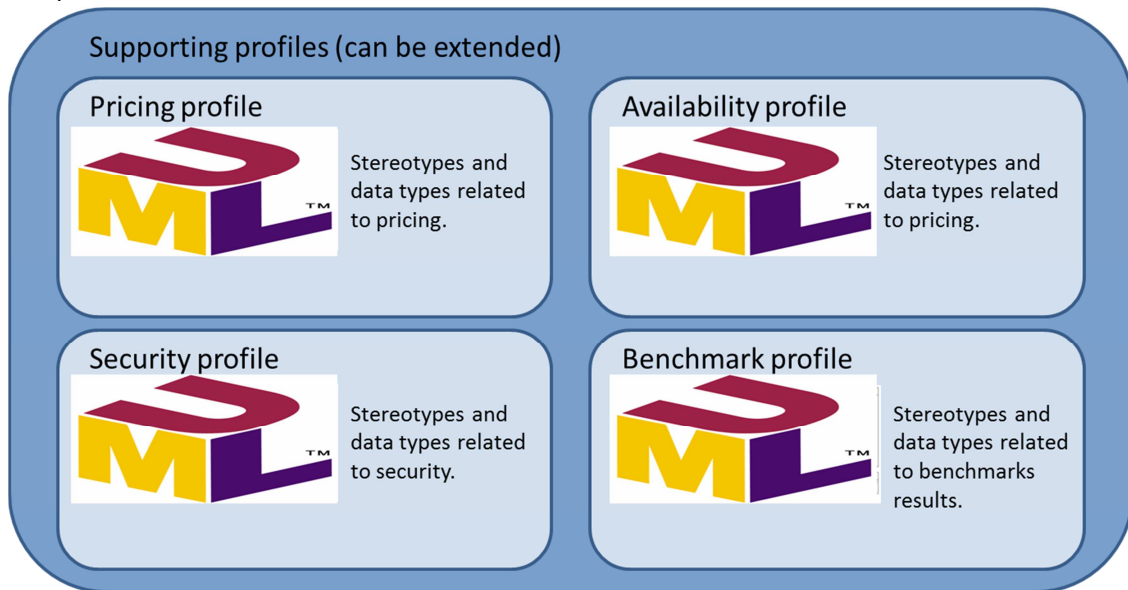


Figure 6:Supporting profiles

### 2.2.2 UML Profiles Description

As described in previous section, CloudML@Artist is organized as a set of UML profiles with hierarchical relationships between them. Next a brief description of each profile:

#### Core profile

UML Profile containing generic stereotypes and data types that can be applied to characterize entities belonging to different cloud providers.

As can be seen in figure 4, the Core profile is divided into 3 sub-profiles for a better understandability and usage:

- IaaS: contains specific IaaS stereotypes and data types. As it is a sub-profile contained in the Core Profile, the stereotypes contained in it can extend directly those stereotypes (common stereotypes) defined at a higher level and can also make use of the common data types at that level. This is applicable also to PaaS and SaaS subprofiles.
- PaaS: contains specific PaaS stereotypes and data types.
- SaaS: contains specific SaaS stereotypes and data types.

#### Amazon EC2 profile

Such profile describes Amazon EC2 provider and allows to create models to specify values for concrete deployments on this provider.

As Amazon EC2 is an IaaS provider, this profile imports IaaS profile and makes use of generic stereotypes defined at that level in the same way any other IaaS provider could do. The use of this inheritance mechanism is very convenient in order to not repeat the creation of stereotypes that have been defined at a higher level.

#### WindowsAzure profile

UML profile with same purpose than Amazon EC2 but oriented to specify Azure's characteristics.

#### Google App Engine profile

This profile has the same objective than Amazon EC2 and Windows Azure's, but it is focused on Google App Engine specification needs. The main difference is that, taking into account that GAE is a PaaS profile, it imports and makes use of PaaS stereotypes instead of IaaS ones.

Next is described a set of “supporting profiles”. These profiles have been created in order to respond to the representation needs of the project at this stage. For now the set of “supporting profiles” is composed of Pricing, Availability, Security and Benchmark profile, but it will be possible to extend it by adding other profiles in a quite simple way in case new requirements arise. Furthermore, they are independent of the CloudML@ARTIST main profile, thus can be individually used (e.g. by other approaches).

#### Pricing profile

Included in "supporting profiles" category, this profile can be applied to any cloud provider to model pricing related aspects.

#### Availability profile

Profile that permits to model cloud provider availability related aspects, as these are expressed in the SLAs. The stereotypes of this profile can be applied on different service elements (e.g. ServiceOfferings), in order to independently describe different SLAs that may apply to different types of services (e.g. Compute SLA, Storage SLA etc.).

#### Security profile

This profile is used to specify security related characteristics at provider level. At the moment the amount of modelled characteristics can be significantly enriched.

#### Benchmark profile

This profile can be included when modelling a cloud provider to specify results of benchmark tests, when attached to specific service instance types.

### **2.2.3 Technical Specification**

The meta-model is delivered in the format of a set of UML profiles created with Papyrus plug-in for Eclipse IDE. It is fully compliant with XMI standard and it is ready to be imported as a profile in any model created with Papyrus.



## 3 CloudML@ARTIST Delivery and Usage

### 3.1 Package information

The CloudML@ARTIST meta-model is available as a set of UML profiles compatibles with the Eclipse IDE and ready to be imported and used to create new models. In order to improve the portability, the last version of CloudML@Artist has been encapsulated as an eclipse plugin, so the profiles are automatically registered in eclipse.

### 3.2 Installation instructions

CloudML@ARTIST has been created by making use of Eclipse ecosystem, more in concrete by using Papyrus 0.10.1 (<http://www.eclipse.org/papyrus/>) design tool plugin installed inside Eclipse Modelling Kepler SR1, that can be downloaded freely from <http://www.eclipse.org/downloads/>.

Once the design environment has been installed it is necessary to download Cloudml@Artist project from the github repository in order to be able to start creating models by using the uml profiles defined in the meta-model.

Last version of Cloudml@Artist can be downloaded from next url:

<https://github.com/artist-project/ARTIST-Modeling/tree/master/Cloud%20Provider%20Metamodel/CloudML%40Artist>

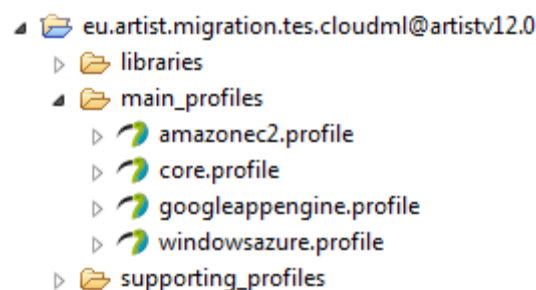
There can be found both the CloudML@Artist eclipse project and also the CloudML@Artist plugin.

As stated out in section 2, for now we have created profiles for Amazon EC2, Windows Azure and GoogleAppEngine cloud providers.

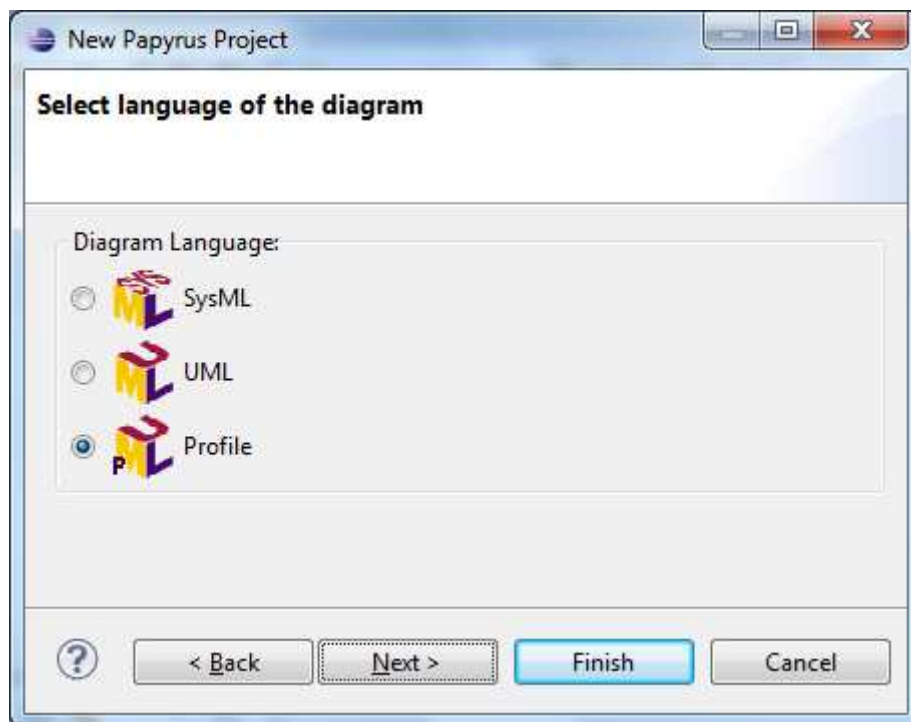
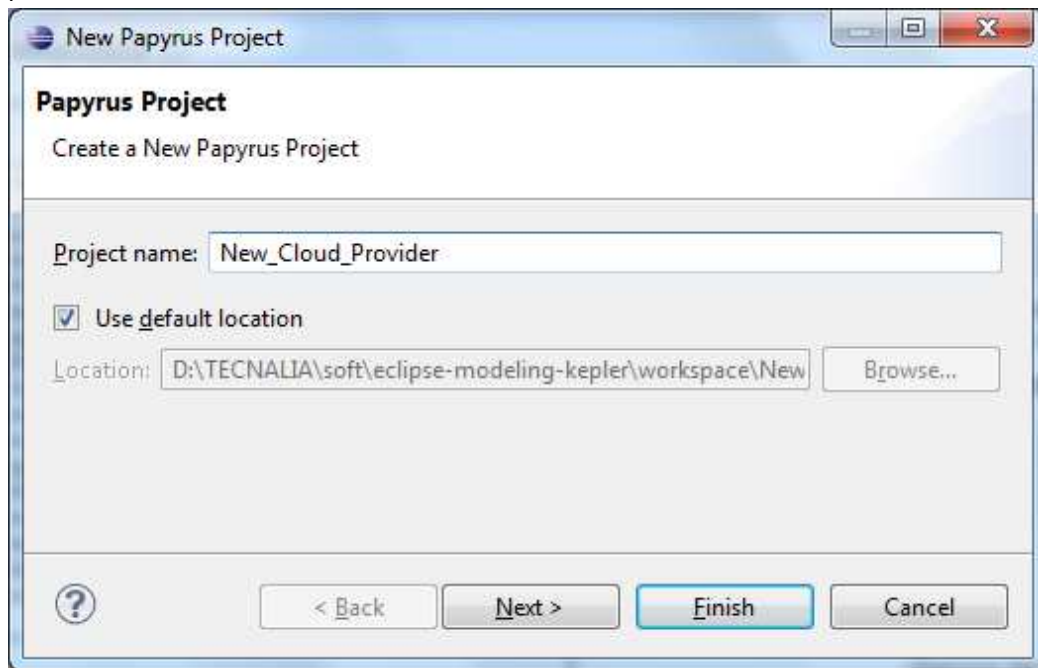
In this section we will go through the steps to create a profile for a new cloud provider using Cloudml@Artist. (more concrete an IaaS provider in the example).

For this, it is needed to:

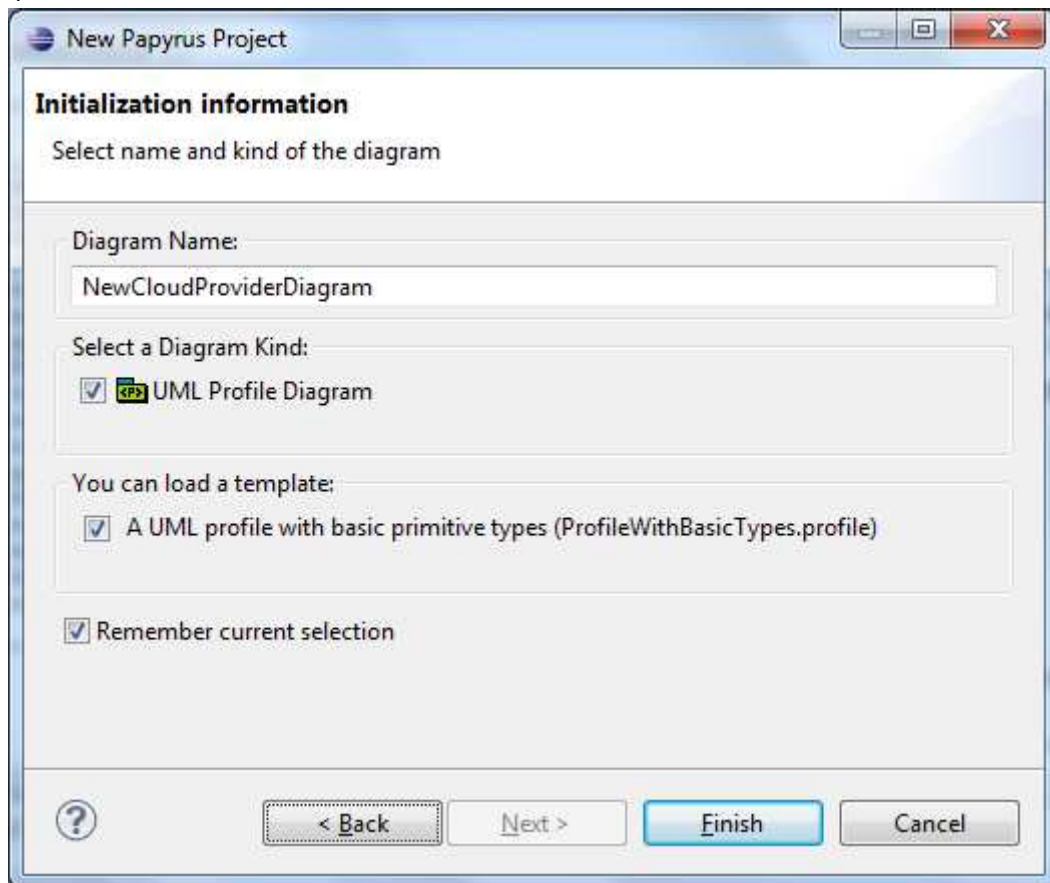
- Import Cloudml@Artist (previously downloaded from github repository) project into Eclipse Eclipse with the aim of complete it with the new profile. As can be seen in next screen, the meta-model is structured as a set of uml profiles stored under two folders: main\_profiles and supporting\_profiles.



- Create a new Papyrus project using the CloudML@ARTIST meta-model as follows:



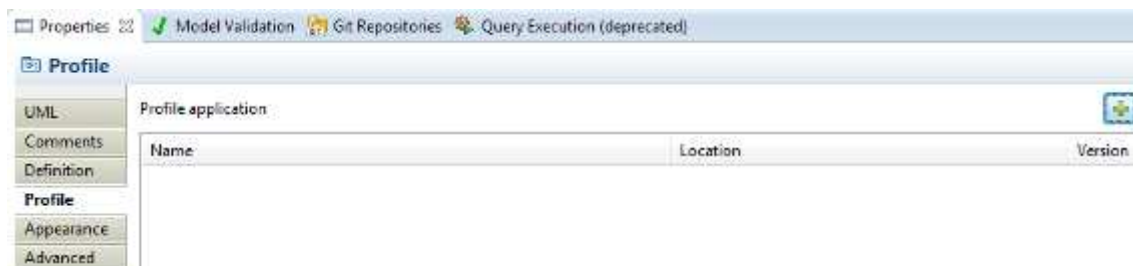
- Create an UML Profile Diagram.

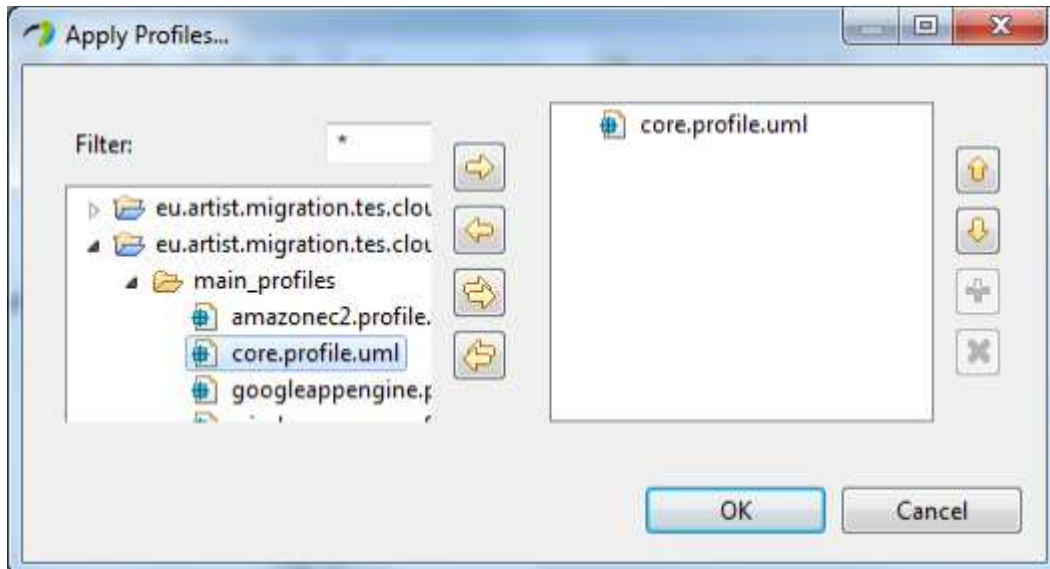


- Make sure to work from the Papyrus perspective within Eclipse environment (In Model Explorer view, we can see all the components that make up the just created diagram).

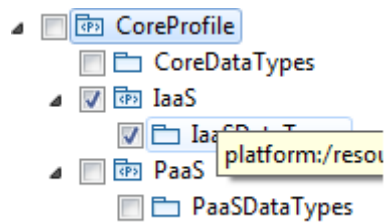


- Apply the *core.profile* profile to the Diagram by pressing [+] button on Diagram's Properties Profile tab.

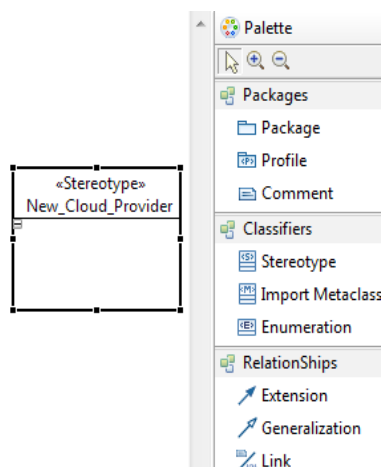


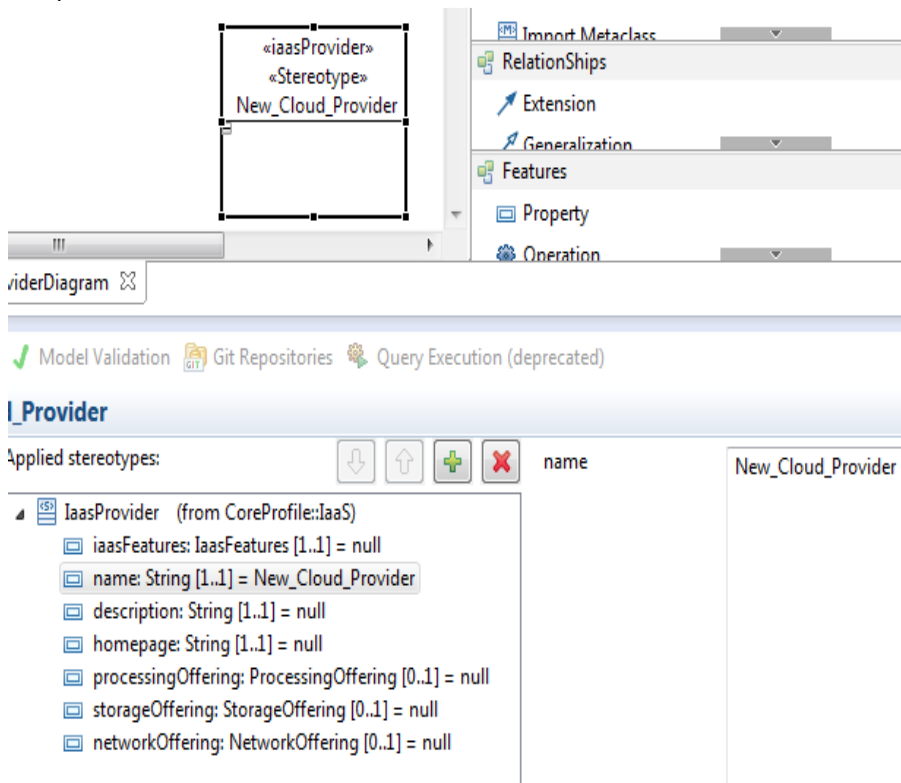


- In next screen, it has to be selected IaaS profile and IaaSDataTypes, as the cloud provider to be modelled is of IaaS type.



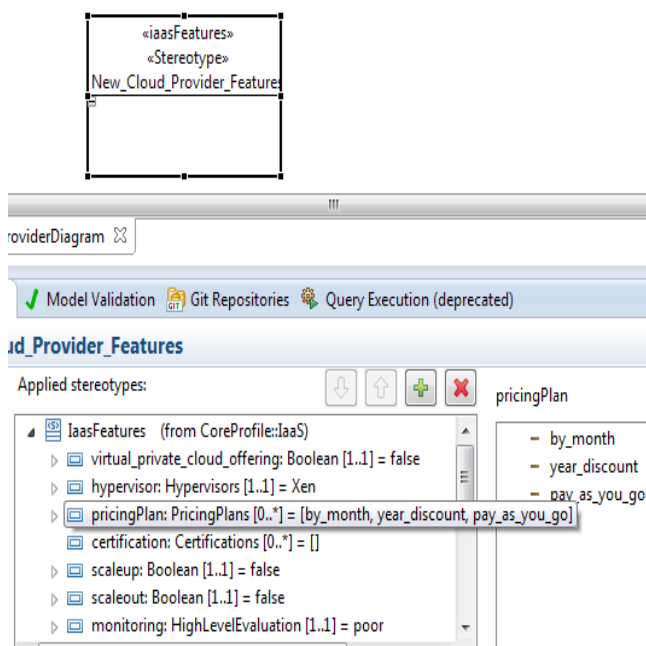
- Once the IaaS profile has been included, it is time to model the provider by including stereotypes describing its characteristics.
- As can be seen in next screenshots, Stereotypes can be dragged from the palette to the diagram and, after this, it is possible to apply higher level stereotypes to them (at IaaS level), so it is not necessary to define again common properties. In the example it is applied *IaaSProvider* stereotype to the new created entity.

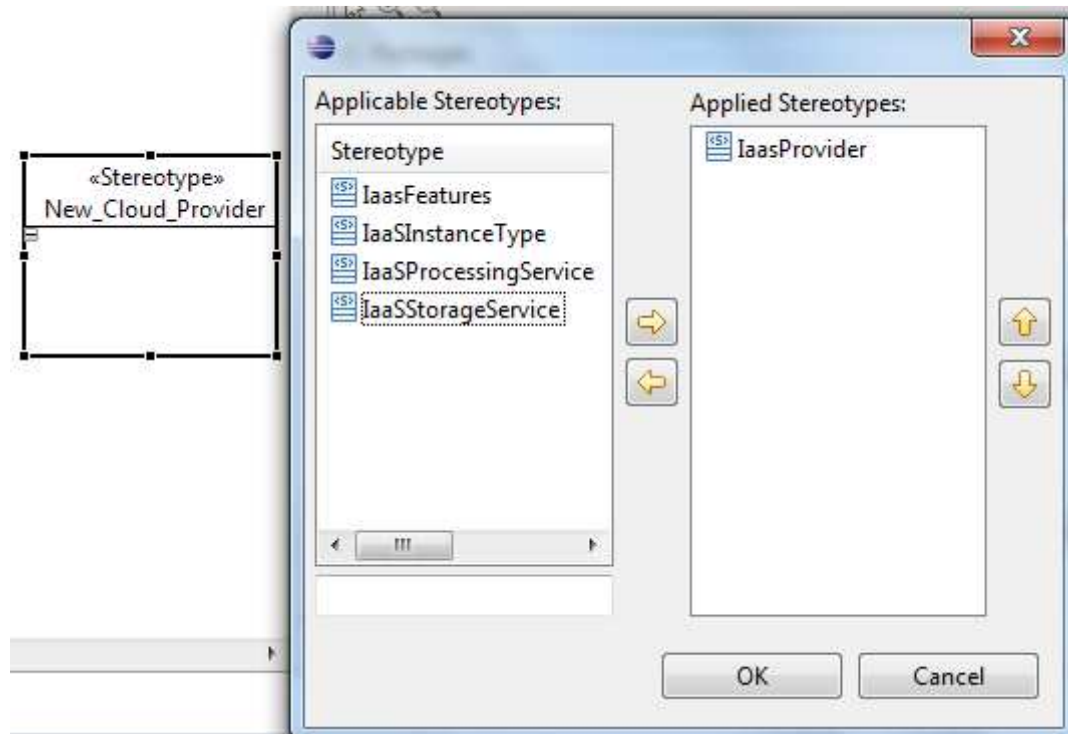




- After doing this, there can be assigned values to the properties defined in the selected stereotype (*IaaSProvider* in this case).

There are some properties whose type corresponds to one of the stereotypes that have to be defined in the profile (for example *iaasFeatures* property in the case of *New\_Cloud\_Provider*).

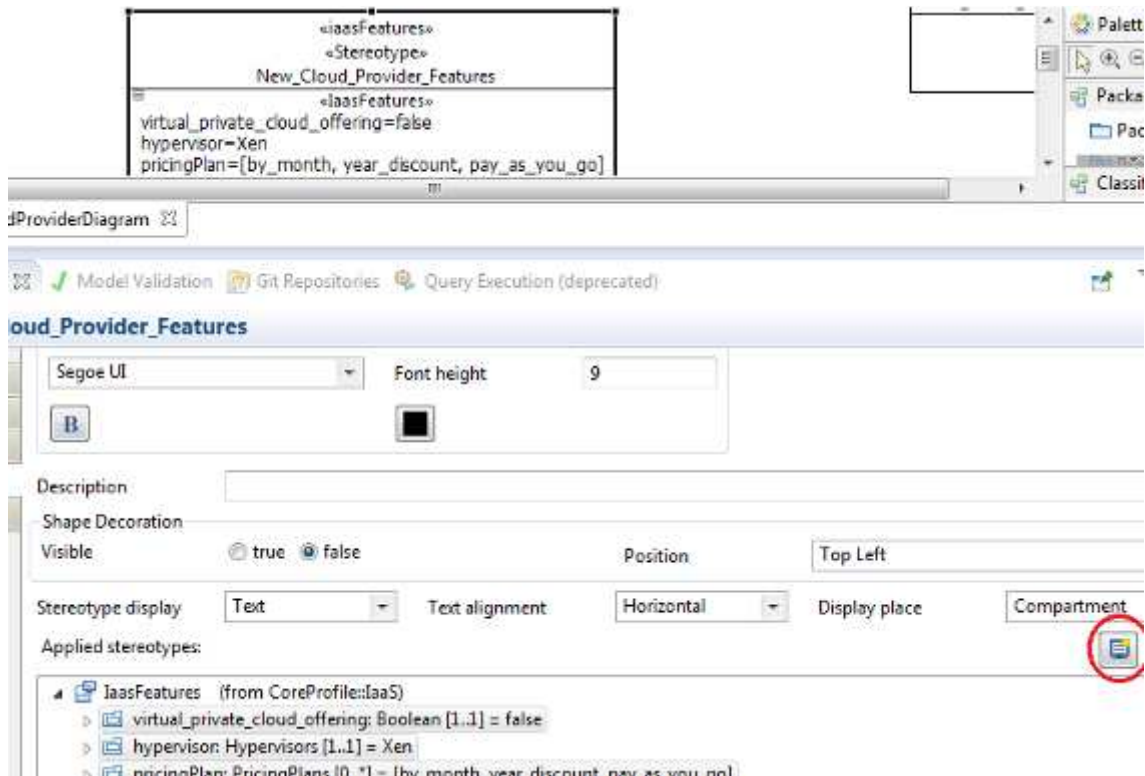




In those cases, before applying a value to the property, first it will be necessary to include a stereotype implementing the needed stereotype (here *iaasFeatures* stereotype).

After the stereotype's properties have been given values it is necessary to specify which ones of those properties have to be shown in the class diagram.

For this, "Appearance" tab has to be clicked and, after that, next steps are: select the properties to show and press first button on the right.



Following these steps it is possible to model the main characteristics of existing IAAS providers by adding new stereotypes to the diagram and giving values to the needed parameters. The same process can be followed to create profiles for PaaS providers.

As can be seen, the metamodel can be extended in an easy way. After adding new providers' profiles, it will be regenerated the eclipse plugin, so these profiles will be able to be imported and applied in deployment models (WP9).

### 3.3 Licensing information

ARTIST project promotes CloudML@ARTIST as a standard for creating cloud models. This may change depending on future needs, but for the moment, CloudML@ARTIST is licensed under EPL.

### 3.4 Download

The CloudML@ARTIST meta-model is available as an Eclipse project in the source code repository hosted at GitHub<sup>1</sup>. The access to the repository is, at the moment of writing, restricted to project members only and to the European Commission. The repository is reachable at the following link:

<https://github.com/artist-project/ARTIST/tree/master/source/Modelling/Cloud%20Provider%20Meta-model/CloudML%40Artist/eu.artist.migration.tes.cloudml%40artist>

<sup>1</sup><https://github.com/>

## 4 Benchmarking Tools Implementation

### 4.1 Functional Description

One of the extensions ARTIST contributes to the CloudML@ARTIST meta-model is the possibility of modelling performance aspects of services offered by cloud providers. Along with the extensions defined in the meta-model, in order to ease the collection of performance data, Work Package 7 will provide also a tool to configure, execute tests and collect results, based on a set of third party benchmarking tools, in order to evaluate performance values that will be added to the cloud provider models.

The Benchmarking Suite has the following two main goals:

- **Homogenization of the tool-kit.** We aim at providing a homogeneous, consistent tool-kit for measuring the performance of cloud services. An initial investigation about benchmarking tools [5] revealed that existing benchmarking tools are very heterogeneous concerning their distribution methods, installation, configuration, and execution, as well as their approach to collecting and formatting their results. With the ARTIST's Benchmarking Suite we aim at homogenizing as much as possible the workflow for executing a benchmark from the download of the tool to the collection of results. This will ease and enable the automation of execution of benchmarks. Where possible existing third-party benchmarking approaches and tools will be used to collect performance data. There exist excellent tools that became *de facto* standards in the benchmarking field (e.g. YCSB [6] for databases benchmarking). We want to include these tools in the ARTIST's Benchmarking Suite because a) they have been proved to work fine, b) they are supported by a large community of experts, c) there is a lot of documentation and tests already carried out and performance data already available, and d) users are already familiar with them.
- **Automation.** The Benchmarking Suite is designed to automate as much as possible and for those tests that allows it the execution of testing tools. Automation aspect is very important in this context because benchmarking tools provided by ARTIST are meant to be executed several times on all the infrastructures modelled also to capture variation in the performance values. We believe that making it possible to manage execution and collection of data automatically saves a lot of time to users and produces better quality results.

### 4.2 Technical Description

#### 4.2.1 Prototype Architecture

Benchmarking Suite architecture is depicted in Figure 7.



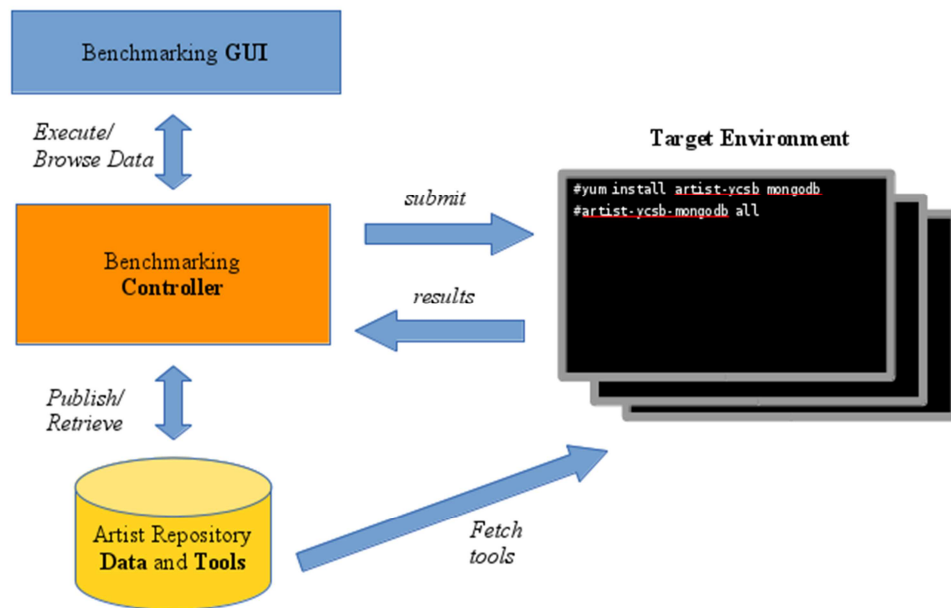


Figure 7: Overall Benchmarking Suite Architecture

At top level there are following components:

- **ARTIST Repository** contains benchmarking tools used in the project to measure performance of cloud services. It also contains results of tests in a standard format that can be consumed by other components.
- **Target Environment** is the system under test. In IaaS offerings, the target environment is a virtual machine running a given Operating System. For PaaS the target environment is represented by services running in the cloud provider infrastructure and usable by applications deployed.
- **Benchmarking Controller** is the main component of this architecture. It automates execution of benchmarking tests.
- **Benchmarking GUI** this component has two main functionalities: a) provides a user interface for submitting execution of new benchmark tests and b) provide a user interface for browsing performance data collected.
- **Backend Raw Data DB** for storing the raw data from the executions. A separate component is used in order not to include all the measurement data in the provider models, for better manipulation of the latter. Thus average values may be included in the models, while more detailed queries may be addressed towards the backend raw data.

The entire architecture constitutes a **framework** for managing the execution and storage of data of benchmarking tools. The modular and flexible architecture allows to incrementally add new tools and automation scripts in the repository extending the coverage of benchmark tests offered by the ARTIST Benchmarking Suite.

## 4.2.2 Components Description

### 4.2.2.1 Benchmarking Repository

The Benchmarking repository component is part of the ARTIST Repository component. The advantage of using the ARTIST Repository also for benchmarking is twofold: a) it is consistent

with other ARTIST tools and b) makes the content indexable, searchable and usable by the other tools.

The repository will contains binary packages of benchmarking tools that can be downloaded and installed on the machine where the benchmark is executed. Each package is identified by four coordinates:

- *package\_name*: the name of the tool (e.g. YCSB, Dwarfs, Filebench).
- *version*: the version of the tool. Multiple versions of a given tool may exist in the repository, mainly for compatibility reasons.
- *platform*: the platform on which the package is meant to be installed (e.g. CentOS, Debian, Windows).
- *format*: the format of the package. Depending on the platform the installation package might be in different formats (e.g. tar.gz, zip, rpm, deb, msi).

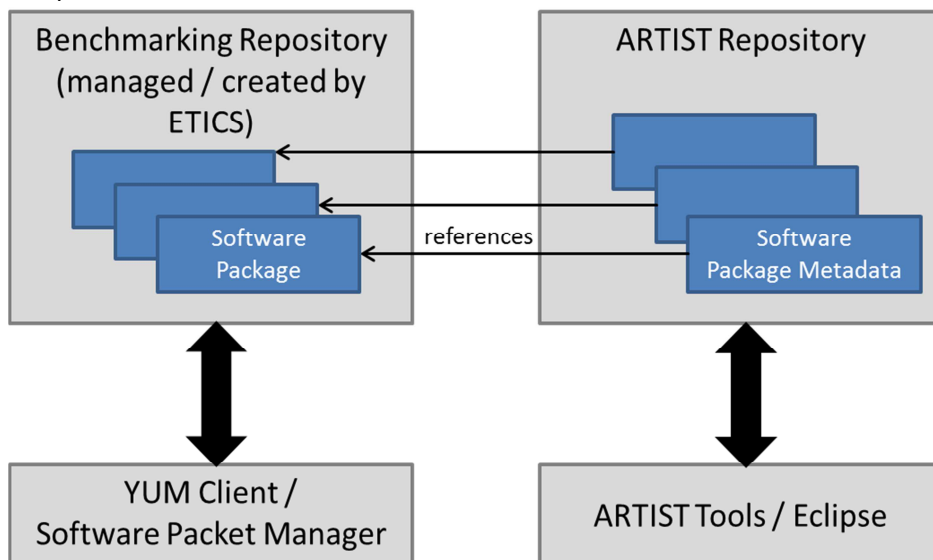
For a given tool, binary packages stored in the repository can be obtained either by compiling the source code or by taking original distribution packages or by repackaging original distribution packages.

A possible layout for a http front-end for the repository that makes it possible to download benchmarking tools from the ARTIST repository is the following:

```
[repository_host]/[package_name]/[version]/[platform]/[package_name].<tar.gz|rpm|deb|msi|...>
```

Moreover, in case of platforms that allows it, a *view* of the repository in a platform-dependant software repository might be provided. For instance, an access to the repository in the APT format (used by Debian platforms for installing software) and/or YUM format (used by RedHat platforms) might be provided.

In order to simplify and partially automatize the creation of binary packages and software repository, we decided to rely, where possible, on an automation tool named ETICS [7] which provides functionalities for creating distribution packages in various formats such as tar.gz, rpm, deb. ETICS also provides a repository that satisfies the requirements listed above. The ETICS repository, even if not integrated in the ARTIST Repository, will be used in this third release as benchmarking tools repository. In next releases of the Benchmarking Suite, tools will be either made available in the ARTIST Repository or the ETICS Repository will be integrated in the ARTIST Repository. A possible integration scenario for the ETICS repository is depicted in Figure 8. The software packages are created and maintained with ETICS. In periodic intervals the ETICS repository contents are scanned and corresponding artefacts are created in the ARTIST repository that contains the search-relevant meta-data and a link to the software package in the ETICS repository. This approach makes the software packages available to packet management systems as well as the ARTIST tools.



**Figure 8: scenario for the ETICS and ARTIST repositories**

An example of download link for YCSB 0.1.4 tool for CentOS platform from the ETICS Repository is the following:

```
http://etics.res.eng.it/eticssoft/repository/artist/artist-ycsb/0.1.4/centos6_x86_64_gcc447/artist-ycsb-0.1.4-1.centos6.x86_64.rpm
```

#### **4.2.2.2 Benchmarking Controller**

The Benchmarking Controller component is in charge of managing executions of benchmarks.

The main objective of this component is to relieve the user from the usual work-flow of benchmarking execution that needs to be done manually: 1) creation of target environment, 2) installation of benchmarking tools, 3) execution of benchmarks and 4) retrieval of results. 5) data storage.

Complete automation of the work-flow will be possible only on certain combination of cloud provider and benchmarking tool. In all the other cases, some step will still require manual intervention of the user. For instance if an IaaS cloud provider does not provide an API to create/destroy virtual machines. In this case, the user has to create the environment manually and then provide the endpoint of the virtual machines to the Benchmarking Controller that will continue the work-flow.

#### **4.2.2.3 Benchmarking GUI**

The benchmarking GUI is the web front-end for launching benchmark tests against target services, by using the predefined benchmarks per application category. It will also be responsible for collecting and parsing results, and storing them in the ARTIST repository.

The GUI will consist of four main areas:

- The test selection tab area. This will be used by the user in order to select one of the benchmarks to run. It will also lead to different configuration pages in the second area.

- The test configuration area. In this part the user will insert the needed configuration parameters for each test, e.g. the number of iterations, periodicity of test, DB size etc.
- The target selection area. This will be used to insert details for the target services, e.g. provider details, login account details etc.
- The results area. This will be used to communicate with the repository and store the results and can also be used to perform queries on the measurements.

The GUI form for filtering historical data is depicted in Figure 9. The GUI form for retrieval of information is depicted in Figure 10. The GUI is provided in its final integrated form in M30 of ARTIST.

Reports

Report Creator

Input Weights for Average: 0.5

<< < [Close] > >>

July 2014

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Input Weights for Cost: 0.5

Start Date: 2014-03-27

End Date: 2014-05-28

Select VM Instance: m1.medium

Select Cloud Test: Database

Select Database: MongoDB

Select Workload: Workload A

Figure 9: Benchmarking GUI Filtering of Historical Data

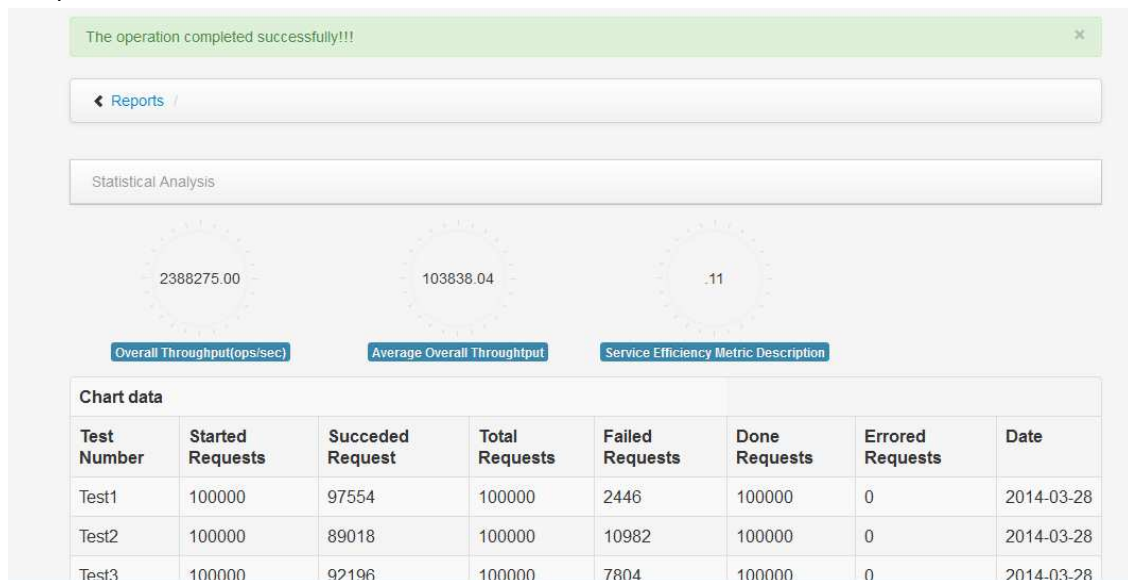


Figure 10: Retrieval of information and depiction

#### 4.2.2.3.1 GUI-Controller Integration

As far as the integration between GUI and controller concerns, it has been obtained by linking the GUI Reporting form with the Artist benchmarking database containing whole the experimentations results. In particular, the user can select the test period of interest, the cloud test as is the benchmark application field (e.g. Java Applications) and the Instance types. In the following the target instance types that have been object of benchmark, grouped by Provider:

		Cloud Provider				
		Amazon	Azure	Openstack	Flexiant	Ovirt
InstanceTypes	t1.micro	X				
	m1.tiny	X		X		
	m1.small	X		X		
	m1.medium	X		X		
	m1.large	X		X		
	m1.xlarge	X		X		
	1Gb-1CPU				X	
	2Gb-2CPU				X	
	4Gb-2CPU				X	
	4Gb-3CPU				X	
	4Gb-4CPU				X	
	default		X			X

Figure 11: InstanceTypes for Cloud Provider

Regarding the Cloud Test, three new options have been introduced according to benchmarking tools used by the controller:

- **Java Applications** for DaCapo Suite tool and all related workloads



Select VM Instance t1.micro

Select Cloud Test Java Applications

Select tool DaCapo Suite

Select workload avroa

**Figure 12: Cloud test for DaCapo**

- **Filesystem** for Filebench tool and all related workloads



Select VM Instance t1.micro

Select Cloud Test Filesystem

Select tool Filebench

Select workload varmail

**Figure 13: Cloud test for Filebench**

- **YCSB** for database to test and all related workloads



Select VM Instance t1.micro

Select Cloud Test YCSB

Select Database MongoDB

Select Workload Workload A

**Figure 14: Cloud test for YCSB**

An example of output is shown in Figure 15. It is related to DaCapo historical data filtering and consists of:

- a table describing the test number, the Cloud Provider, the target Platform, the Date of execution and the response time (milliseconds in this case)
- an indicator of average for the performance results (it could be more than one in case of more performance data)

- an indicator related to the Service Efficiency Metric (it could be more than one in case of more performance data)
- a chart describing the performance trend for each test (it could be more than one in case of more performance data)

Obviously the views will be different for each tool depending on own database structure and performance metrics to measure.

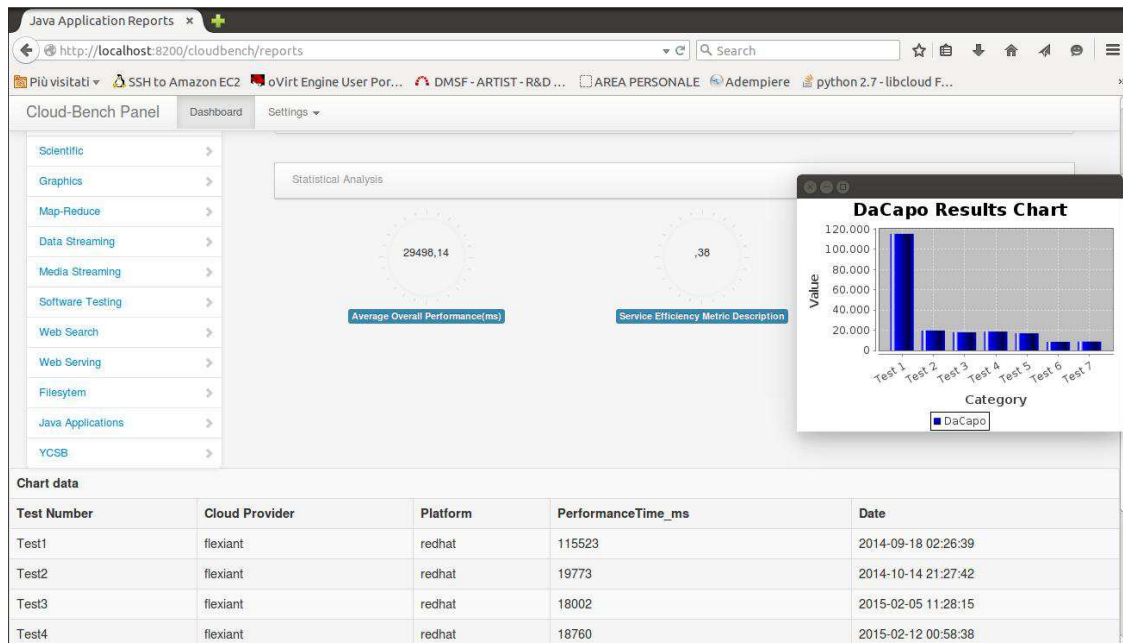


Figure 15: GUI – controller integration

So far it is not possible to link directly the GUI with the Controller in order to carry out the test directly by the interface; however, the component related to the execution is in plan as future development.

#### 4.2.2.4 Benchmark Database

The results obtained by the execution will be parsed first and transferred back then in order to be processed and included in the CloudML model descriptions. In addition, a mysql raw database schema has been created and provided in case the results from the benchmark tests needed to be stored locally. The database structure is depicted in Figure 16. Actually the GUI and Controller components are not integrated; however the benchmarking controller, the main backend component can also be used standalone to perform measurements on the various application types. An example of mysql table containing benchmark results is depicted in Figure 17.



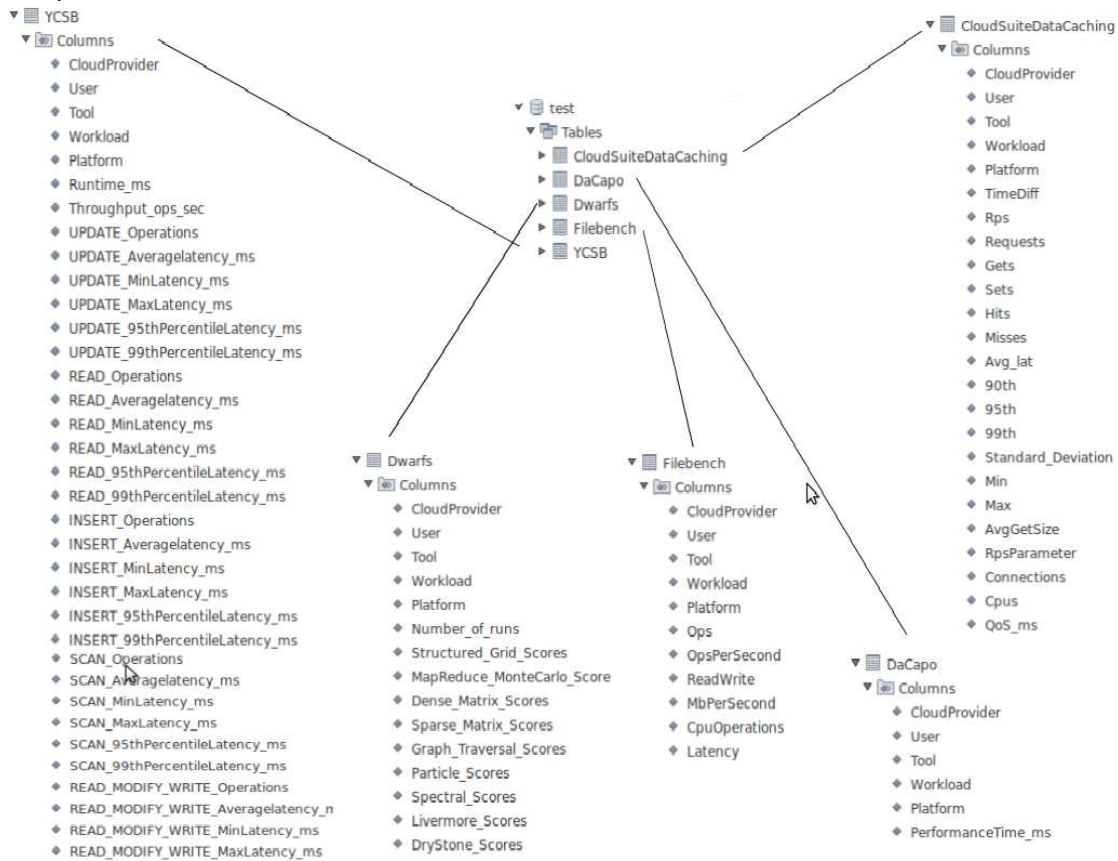


Figure 16: Benchmark database schema

#	CloudProvider	User	Tool	Workload	Platform	PerformanceTime_ms
1	Amazon	ec2-user	DaCapoSuite	Avrora	CentOS	18199
2	Openstack	root	DaCapoSuite	Avrora	CentOS	6906
3	LocalVM	root	DaCapoSuite	Avrora	CentOS	7630
4	Amazon	ec2-user	DaCapoSuite	Tomcat	CentOS	21801
5	Openstack	root	DaCapoSuite	Tomcat	CentOS	11980
6	LocalVM	root	DaCapoSuite	Tomcat	CentOS	18905

Figure 17: Example of Benchmark database view

#### 4.2.2.4.1 Database REST interface

The results obtained by the execution will be visible also via web; in such sense, a REST interface using FLASK has been developed; FLASK<sup>2</sup> is a micro-framework for Python based applications. In particular it is possible to distinguish the following absolute paths:

- <http://ipaddress:port/benchmarking/ui> will return the menu page
- <http://ipaddress:port/benchmarking/ui/<tool>> will return the entire result-set for the tool

<sup>2</sup> <http://flask.pocoo.org/>



- `http://ipaddress:port/benchmarking/rest/results/<tool>` will return data in json format

As an alternative it is possible to query the db directly from the browser:

`http://localhost:5000/benchmarking/ui/dacapo?InstanceType=m1.small&&CloudProvider=Amazon`

Such path will return the performance data for DaCapo as case of Amazon instance m1.small.

Date	CloudProvider	InstanceType	Workload	Platform	PerformanceTime_ms
'2014-04-09 11:59:59'	Amazon	m1.small	Avrora	RedHat	48634
'2014-04-09 12:10:06'	Amazon	m1.small	Avrora	RedHat	44488
'2014-04-09 12:14:12'	Amazon	m1.small	Avrora	RedHat	43422
'2014-04-09 15:26:17'	Amazon	m1.small	Avrora	RedHat	50679
'2014-04-10 12:01:01'	Amazon	m1.small	Avrora	RedHat	46633
'2014-04-11 15:47:57'	Amazon	m1.small	Avrora	RedHat	42405
'2014-04-11 15:53:32'	Amazon	m1.small	Avrora	RedHat	46019
'2014-04-11 16:01:31'	Amazon	m1.small	Avrora	RedHat	46679
'2014-04-11 16:07:21'	Amazon	m1.small	Avrora	RedHat	46645
'2014-04-11 17:40:19'	Amazon	m1.small	Fop	RedHat	21449

Figure 18: Example of REST interface for the database

### 4.2.3 Technical Specification

The version of the Benchmarking Suite delivered is a set of selected third-party benchmarking tools packaged and delivered thorough the ARTIST Repository. The ARTIST Repository is compliant with the two most common standards for software repositories in the Linux environment: *Apt* and *Yum*. Packages are available in following formats: *deb*, *rpm* and *tar.gz*.

The technologies used in the third-party benchmarking tools being part of the Benchmarking Suite distribution varies from tool to tool. The most common technology used is, by far, the *C/C++* programming language, preferred because of its low access to machine resources and the small memory footprint. Also *Java* and *Python* technologies are largely used.

### 4.2.4 Third-party benchmarking tools

We have implemented the repackaging of a selected set of third-party benchmarking tools delivered through the ARTIST Repository. We will complete the overall architecture (integration between benchmarking controller and GUI) during the third year of ARTIST.

#### 4.2.4.1 YCSB

YCSB [4] is one of the most popular database benchmarking tool. Basically, it is an open source workload generator and can be considered a standard benchmark able to evaluate different systems on common workload.

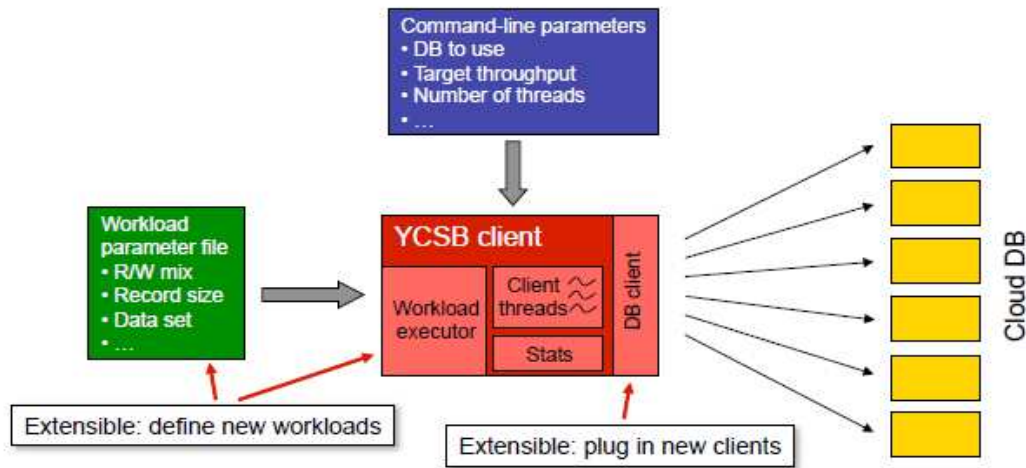


Figure 19: YCSB Architecture

In a nutshell, it is a Java framework used for generating the data to be loaded to the database and the operations which make up the workload.

A key feature of the YCSB framework is that it is extensible; it supports easy definition of new workloads, in addition to making it easy to benchmark new systems.

There are two ways to define workloads:

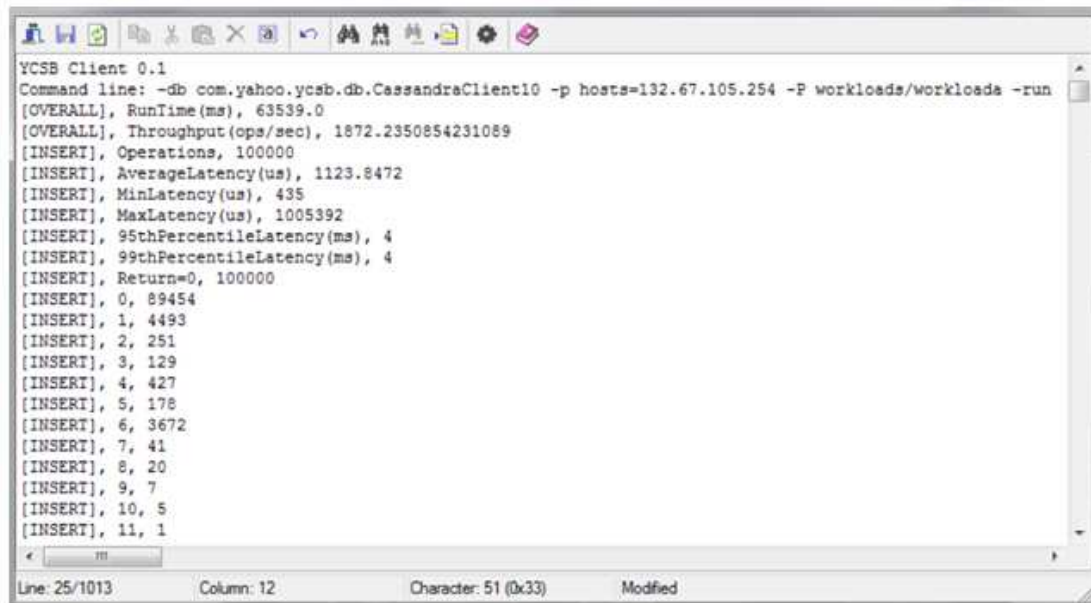
- Adjust parameters to an existing workload (via properties file)
- Define a new kind of workload (by writing Java code)

In terms of metrics, YCSB focus is related to performance and scale out; we can consider two different benchmark tiers:

- **Tier 1 – Performance:**
  - Increase offered throughput until saturation
  - Measure resulting latency/throughput curve
- **Tier 2 – Scalability:**
  - Scaleup – Increase hardware, data size and workload proportionally
  - Measure latency (should be constant)
  - Elastic speed-up – Run workload against N servers
  - Measure timeseries of latencies

The ARTIST Repository contains the version 0.1.4 of YCSB and the related test results; it has been possible to test such version both on Windows and Linux Oss.

Figure 20 shows an example of YCSB output against a Cassandra database.



```
YCSB Client 0.1
Command line: -db com.yahoo.ycsb.db.CassandraClient10 -p hosts=132.67.105.254 -P workloads/workloads -run
[OVERALL], RunTime(ms), 63539.0
[OVERALL], Throughput(ops/sec), 1872.2350854231089
[INSERT], Operations, 100000
[INSERT], AverageLatency(us), 1123.8472
[INSERT], MinLatency(us), 435
[INSERT], MaxLatency(us), 1005392
[INSERT], 95thPercentileLatency(ms), 4
[INSERT], 99thPercentileLatency(ms), 4
[INSERT], Return=0, 100000
[INSERT], 0, 89454
[INSERT], 1, 4493
[INSERT], 2, 251
[INSERT], 3, 129
[INSERT], 4, 427
[INSERT], 5, 178
[INSERT], 6, 3672
[INSERT], 7, 41
[INSERT], 8, 20
[INSERT], 9, 7
[INSERT], 10, 5
[INSERT], 11, 1
Line: 25/1013 Column: 12 Character: 51 (0x33) Modified
```

Figure 20: YCSB output example

#### 4.2.4.2 Dwarf

Berkeley dwarfs is a benchmarking suite aiming to cover a variety of elementary operations that correspond to higher-level applications. In ARTIST, we will be utilizing an extended version created in the context of the BONFIRE project [8]. This version enables the remote installation of the executables plus a number of additions in the result reporting (e.g. incremental load adjustment).

The tests involve the following applications/areas:

- Dense and sparse matrix operations
- Spectral analysis
- MapReduce
- Structured grids
- And N-body methods

The configuration can be performed via a control file, in which various parameters (such as test sizes, incremental workloads and iterations) can be set. Results can be directly downloaded from the target environment through a provided script. Result reporting include various statistics per test (system time, wall clock time, CPU % etc.), system information (memory usage etc.) in the same reporting structure as Figure 21.

```

GNU nano 2.2.6                                     File: dwarf_scores.csv
DWARF NAME, SCORE
Structured Grid, 279.292690673
MapReduce (Monte Carlo), 224.137077205
Dense Matrix, 284.450994842
Sparse Matrix, 243.501881806
Graph Traversal, 206.086561547
Particle, 209.161911061
Livermore, 145.350925926
Drystone, 190.597988643
MemoryBDW, 230.138461364
IOZONE, 1009.55903398

```

Figure 21: Dwarfs scores

Also for each test performed there are detailed statistics with regard to its configuration and specific timings such as minimum, maximum and average times, standard deviation and coefficient of variation (Figure 22) and the statistics for each individual run (Figure 23).

```

GNU nano 2.2.6                                     File: MapReduce(MonteCarlo)_stats.csv
DWARF NAME, ALGORITHM NAME, WORKLOAD, SAMPLE SIZE, MIN TIME, MAX TIME, AVERAGE TIME, STDEV TIME, COEFFICIENT OF VARIATION
MapReduce (Monte Carlo), Quasi-Monte Carlo integration, 20, 10.00, 1998.00, 2026.00, 2019.30, 7.75, 0.0038

```

Figure 22: Dwarfs statistics

```

GNU nano 2.2.6                                     File: quicksortOutput.csv
benchmark exe: quicksort
# args: 10000000, 10000000, 0, 10, /home/artist/dwarfs/BonFIREBenchmarkWithoutApplications/benchmarking/Output/
DATE, DWARF NAME, ALGORITHM NAME, SIZE, EXEC TIME (millisec), CPU CLOCK TIME (millisec)
2013-06-13T18:16:58, Graph Traversal, Quicksort, 10000000, 1540, 1500
2013-06-13T18:16:59, Graph Traversal, Quicksort, 10000000, 1536, 1500
2013-06-13T18:17:01, Graph Traversal, Quicksort, 10000000, 1551, 1510
2013-06-13T18:17:03, Graph Traversal, Quicksort, 10000000, 1543, 1520
2013-06-13T18:17:04, Graph Traversal, Quicksort, 10000000, 1539, 1500
2013-06-13T18:17:06, Graph Traversal, Quicksort, 10000000, 1543, 1510
2013-06-13T18:17:08, Graph Traversal, Quicksort, 10000000, 1539, 1510
2013-06-13T18:17:09, Graph Traversal, Quicksort, 10000000, 1536, 1510
2013-06-13T18:17:11, Graph Traversal, Quicksort, 10000000, 1542, 1510
2013-06-13T18:17:13, Graph Traversal, Quicksort, 10000000, 1542, 1520

```

Figure 23: Individual execution statistics

These results should be parsed and added to the appropriate structures of ARTIST (e.g. ARTIST repository) in order to aid in the selection process for the most appropriate offering performed by WP9.

#### 4.2.4.3 CloudSuite

CloudSuite [9] is a benchmark suite for emerging scale-out applications; the suite consists of eight applications selected based on their data-center popularity:

- Data caching
- Media streaming
- Data serving
- Graph analytics
- Data analytics
- Software testing
- Web search
- Web serving

We have tested the release 1.0 concerning the first component on Linux; the ARTIST repository contains the tool (included the prerequisite software packages) and the related test results.

#### 4.2.4.4 Data Caching

Traditionally main storage is too slow to meet the QoS requirements by modern applications; in order to solve this kind of problem most of today's server systems dedicate separate caching servers that store informations in their DRAM.

*CloudSuite* Data Caching is an open source benchmark used to measure QoS in aforesaid context; basically such application uses Memcached data caching server (and Libevent, a library necessary to its execution) to simulate the behaviour of a *Twitter* caching server using the twitter dataset. The metric of interest is throughput expressed as the number of requests served per second.

A sample output of Data Caching tool output is shown in Figure 24.

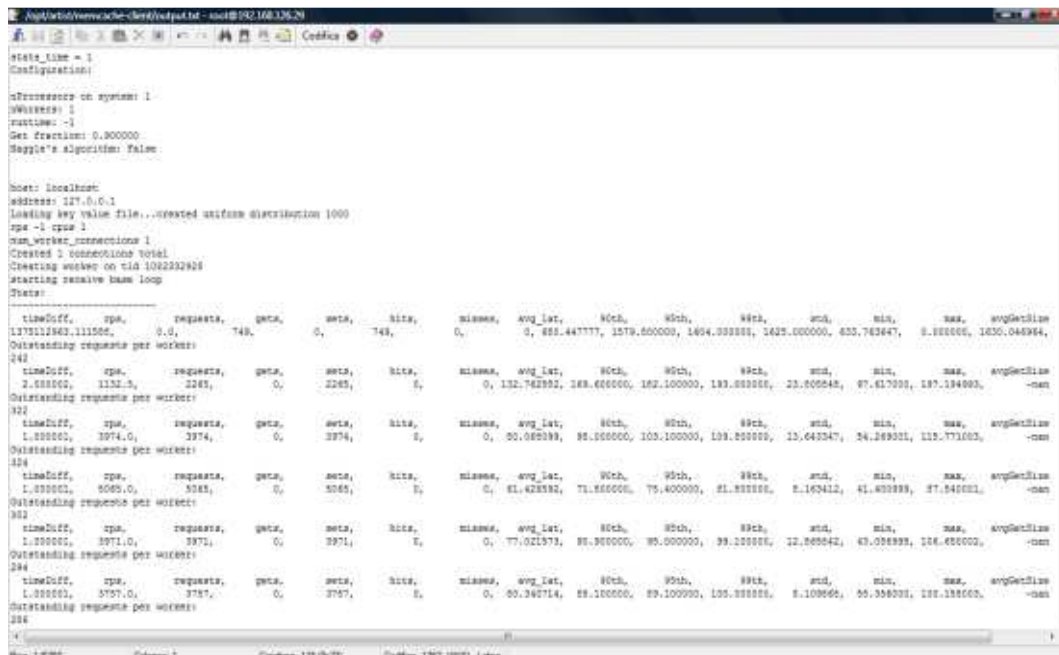


Figure 24: CloudSuite Data Caching output example

#### 4.2.4.5 Filebench

Filebench [10] is a very flexible file system and storage benchmarking tool. Basically it is an open source C frameworks that uses loadable workload personalities to allow easy emulation of complex applications. We have tested the last version and it's resulted quick to set up and easy to use.

Filebench includes many features to facilitate file system benchmarking:

- Multiple workload types support via loadable personalities .
- Ships with a library of more than 40 pre-defined personalities, including the ones that describe mail, web, file, and database servers behaviour.

Workload personalities define the workload to apply to the system; they include tunables for scaling workloads to specific systems.

- Easy to add new personalities using reach Workload Model Language (WML) [11]
- Multi-process and multi-thread workload support.

- Configurable directory hierarchies with depth, width, and file sizes set to given statistical distributions.
- Support of asynchronous I/O and process synchronization primitives.
- Integrated statistics for throughput, latency, and CPU cycle counts per system call.
- Tested on Linux, FreeBSD, and Solaris platforms.

The ARTIST Repository contains the version 1.4.9 of Filebench tool and the related test results carried out on Linux. In the following an example (Figure 25) of output concerning Fileserver emulator benchmark:

```
root@user# go_filebench
Filebench Version 1.4.9
12324: 0.000: Allocated 170MB of shared memory
filebench> load fileserver
12462: 2.869: FileServer Version 2.2 personality successfully loaded
12462: 2.869: Usage: set $dir=<dir>
12462: 2.869:      set $meanfilesize=<size>      defaults to 131072
12462: 2.869:      set $nfiles=<value>                defaults to 10000
12462: 2.869:      set $nthreads=<value>                defaults to 50
12462: 2.869:      set $meanappendsize=<value>          defaults to 16384
12462: 2.869:      set $iosize=<size>                   defaults to 1048576
12462: 2.869:      set $meandirwidth=<size>             defaults to 20
12462: 2.869: (sets mean dir width and dir depth is calculated as log (width, nfiles)
12462: 2.869:      run runtime (e.g. run 60)
filebench> set $dir=/mnt
filebench> run 60
12462: 4.909: Creating/pre-allocating files and filesets
12462: 4.918: Fileset bigfileset: 10000 files, avg dir width = 20, avg dir depth = 3.1, 1240.757MB
12462: 5.280: Removed any existing fileset bigfileset in 1 seconds
12462: 5.280: making tree for filset /tmp/bigfileset
12462: 5.290: Creating fileset bigfileset...
12462: 6.080: Preallocated 7979 of 10000 of fileset bigfileset in 1 seconds
12462: 6.080: waiting for fileset pre-allocation to finish
12466: 6.080: Starting 1 filereader instances
12467: 6.081: Starting 50 filereaderthread threads
12462: 7.137: Running...
12462: 67.142: Run took 60 seconds...
12462: 67.145: Per-Operation Breakdown
statfile1      128311ops    2138ops/s    0.0mb/s      0.0ms/op    2320us/op-cpu [0ms - 0ms]
deletefile1    128316ops    2138ops/s    0.0mb/s      0.2ms/op    2535us/op-cpu [0ms - 458ms]
closefile3     128323ops    2139ops/s    0.0mb/s      0.0ms/op    2328us/op-cpu [0ms - 0ms]
readfile1     128327ops    2139ops/s    283.8mb/s    0.1ms/op    2460us/op-cpu [0ms - 267ms]
openfile2     128329ops    2139ops/s    0.0mb/s      0.0ms/op    2332us/op-cpu [0ms - 2ms]
closefile2     128332ops    2139ops/s    0.0mb/s      0.0ms/op    2332us/op-cpu [0ms - 0ms]
appendfilerandl 128337ops    2139ops/s    16.6mb/s    0.1ms/op    2377us/op-cpu [0ms - 559ms]
openfile1     128343ops    2139ops/s    0.0mb/s      0.0ms/op    2353us/op-cpu [0ms - 2ms]
closefile1     128349ops    2139ops/s    0.0mb/s      0.0ms/op    2317us/op-cpu [0ms - 1ms]
wrtfile1      128352ops    2139ops/s    265.2mb/s    0.1ms/op    2601us/op-cpu [0ms - 268ms]
createfile1    128358ops    2139ops/s    0.0mb/s      0.1ms/op    2396us/op-cpu [0ms - 267ms]
12462: 67.145: IO Summary: 1411677 ops, 23526 ops/s, (2139/4278 r/w), 565mb/s, 393us cpu/op, 0.2ms latency
12462: 67.145: Shutting down processes
root@user#
```

Figure 25: Filebench output example

#### 4.2.4.6 DaCapo

In order to evaluate Java-based applications can be used the DaCapo benchmark suite; it is designed to facilitate performance analysis of Java Virtual Machines, compilers and memory management. This benchmark suite is intended as a tool for Java benchmarking by the programming language, memory management and computer architecture communities. It consists of a set of open source, real world applications with non-trivial memory loads. The DaCapo suite consists of the following benchmarks:

- AVRORA: simulates a number of programs running on a grid of AVR micro-controllers
- BATIK: produces a number of Scalable Vector Graphics (SVG) images based on the unit tests in Apache Batik
- ECLIPSE: executes jdt performance tests for the Eclipse IDE
- FOP: parses/formats XSL-FO file and generates a PDF file
- H2: executes a JDBC benchmark using a number of transactions against a banking model application



- JYTHON: interprets pybench Python benchmark
- LUINDEX: uses Lucene to indexes a set of documents
- LUSEARCH: uses Lucene to search of keywords over a data corpus
- PMD: analyzes a set of Java classes for a range of source code problems
- SUNFLOW: renders a set of images using ray tracing
- TOMCAT: runs a set of queries against a Tomcat server retrieving and verifying the resulting webpages
- TRADEBEANS: runs the Daytrader benchmark via Java Beans to a GERONIMO back-end
- TRADESOAP: runs the Daytrader benchmark via SOAP to a GERONIMO backend
- XALAN: transforms XML documents into HTML ones

#### 4.2.4.7 FINCoS

FINCoS<sup>3</sup> framework is a java-based set of benchmarking tools for load generation and performance measurement of Event Processing systems management. Its architecture is shown in Figure 26.

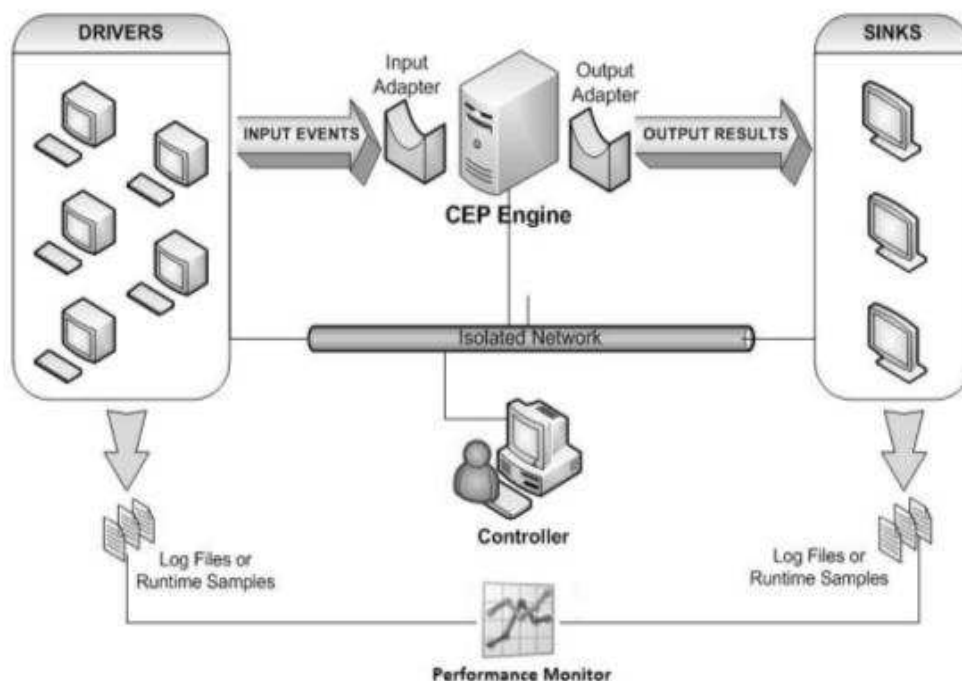


Figure 26: FINCoS framework architecture

It is possible to distinguish five different components:

- Driver: simulates external source of event; its configuration allows to specify the load to submit to the system to test, and the typology of workload (synthetic generated by user directives or based on external files)

<sup>3</sup> <https://code.google.com/p/fincos>

- Sink: receives output events resulting from the queries running on CEP engines and stores results in log files
- Controller: interface between framework and user; responsible for the environment configuration phases
- Adapters: converts events from the framework into a target system compliant format; the tool communicates with CEP engines directly (via customized plug-and-play adapters) or via JMS messages
- Performance Monitor: collects performance metrics, both real-time and offline.

In the following an example of performance results related to a simple test scenario:

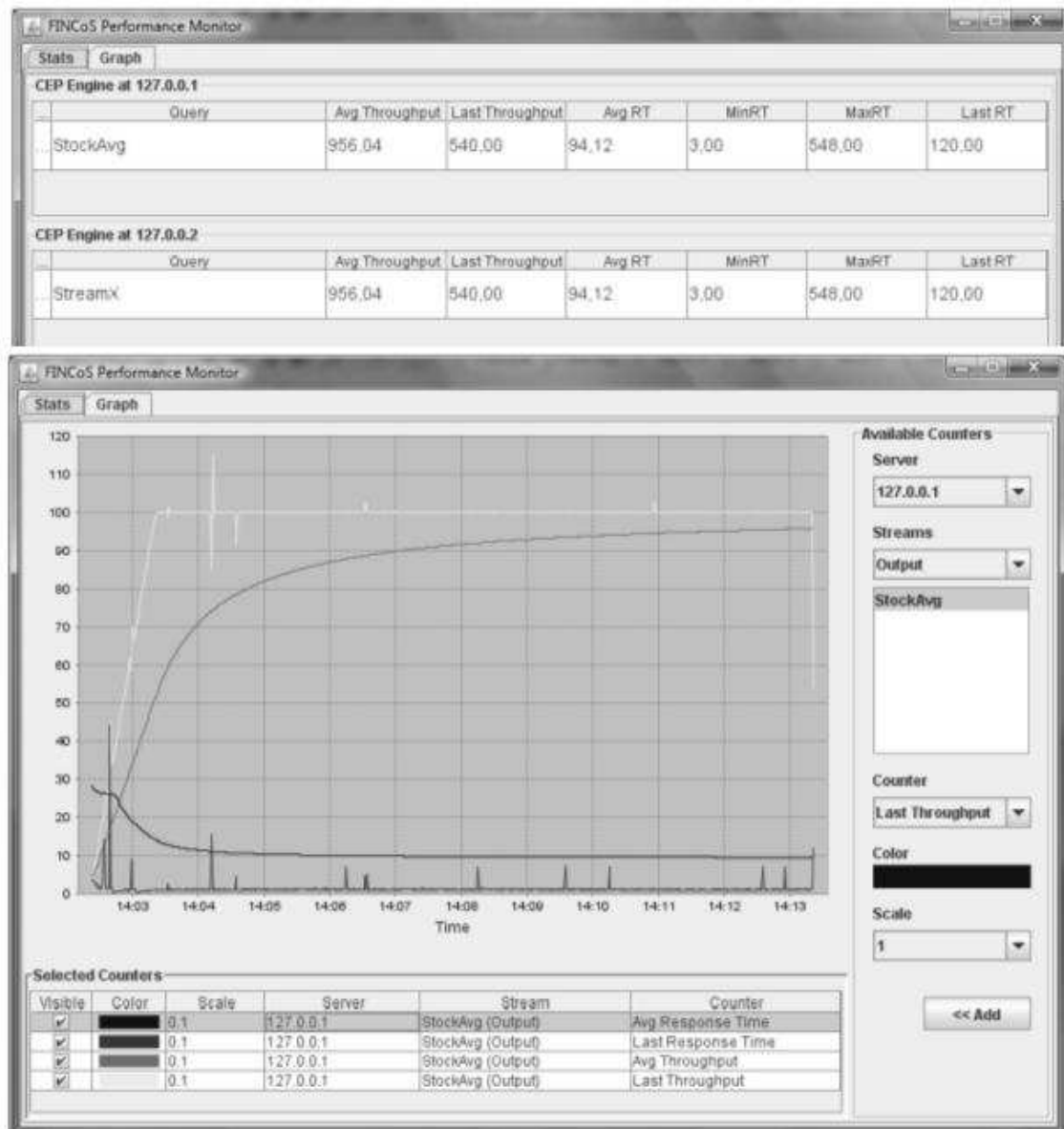


Figure 27: FINCoS output example

Actually the execution is allowed through an interactive GUI; this means that the usage does not fit totally with the benchmarking controller functionalities cause it is not possible to use the own API to configure the several scenarios. So far we could use the controller to install the CEP engine remotely and retrieve the results of the performance test locally. Unfortunately the



configuration steps remains manual. This issues will be solved in the future since a new version of the framework including any ready-to-run benchmark should be released. Currently, we are evaluating a draft solution<sup>4</sup> proposed by FINCoS itself that would allow the extension of the framework in order to run tests in a not-interactive mode.

#### 4.2.5 Benchmarking experiments and related performance model population

In order to estimate the performance of the cloud resources for a successful migration, the usage of the benchmarking tools described in the previous sections, is needed. The measurement results from the benchmarking process are incorporated in the CloudML@ARTIST framework in order to assist with provisioning decisions for cloud users.

During the execution process, we utilized workloads from DaCapo benchmarking Suite, YCSB framework and Filebench. The selected workloads from each test were running on instances in three different cloud environments: Amazon EC2, Microsoft Azure and Flexiant. In Flexiant's case, cloud resources were provided by MODAClouds<sup>5</sup> in order to execute benchmarking experiments, as part of ARTIST and MODAClouds eu projects collaboration. For each cloud provider, different types of VM instances were selected in order to examine as much cloud resources as possible and give a complete view of the performance of the provided cloud offerings. Information regarding the selected VM instance characteristics is presented in Table 1.

The execution of the tests took place at specific hours (at different time intervals) during a period of eight months (from July 2014 to 28<sup>th</sup> of February 2015) and the average values were extracted for each case. Moreover, the different time zones of the three respective regions were taken into consideration so that the peak hours were the same in each zone. In Amazon EC2 case, the virtual machines were running in North Virginia datacenter while for Microsoft Azure and Flexiant in Ireland and United Kingdom respectively.

**Table 1: VM instance characteristics**

Cloud Provider	VM instance	Region
Amazon EC2	t1.micro	N.Virginia
	m1.medium	N.Virginia
	m1.large	N.Virginia
Microsoft Azure	A1	Ireland
	A2	Ireland
Flexiant	4GB RAM- 3CPU	United Kingdom
	2GB RAM-2CPU	United Kingdom
	4GB RAM -2CPU	United Kingdom
	2GB RAM -2CPU	United Kingdom

After completing the benchmarking process the results are retrieved from the local database, processed and the appropriate graphs are depicted.

In order to draw conclusions from the execution of the benchmarks, one should compare between same colour bars, indicating similar workloads (). From the graphs it is evident that the performance for a specific workload varies and depends on both the type of workload and

<sup>4</sup> <https://code.google.com/p/fincos/issues/detail?id=3>

<sup>5</sup> [www.modaclouds.eu](http://www.modaclouds.eu)

the VM instance size. For instance for DaCapo benchmark the workloads performance across Azure (A2 Standard), Amazon m1.large and Amazon (m1.medium) is almost similar apart from some cases where Amazon provides better results for workload h2 while Azure for workload avrora. Extended information for the benchmarking results is provided at the Deliverable 7.4.

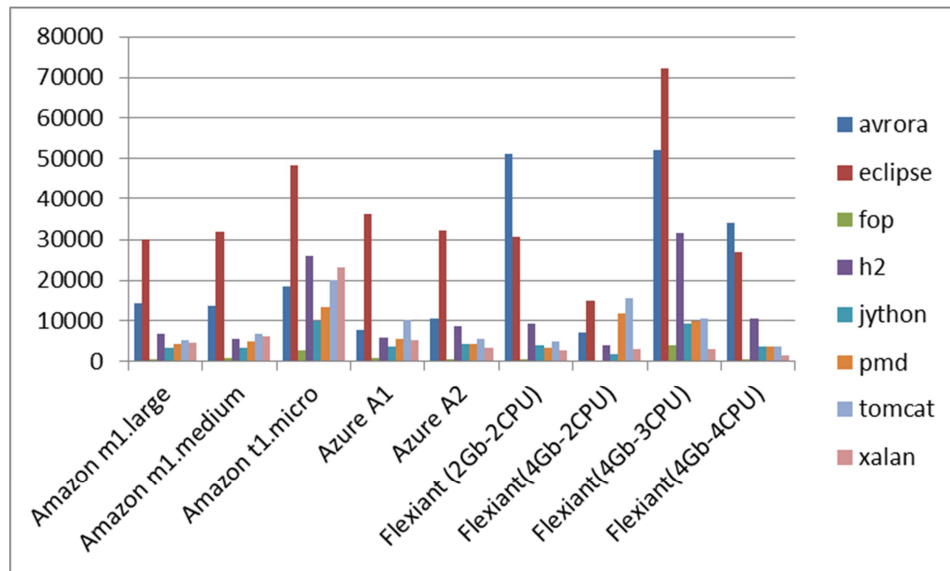


Figure 28: Performance time in ms for DaCapo workloads

Having the performance information extracted from the benchmarking process, we create the respective placeholders in the CloudML@ARTIST model framework and populate them with the benchmarking results. For example, in Amazon EC2 case, in order to include the performance placeholders in the model the following steps should be undertaken.

- As already mentioned in section 2, profile for Amazon EC2 IaaS cloud provider has already been created and included in the CloudML@ARTIST model.
- Apply the *benchmark.profile* to the Diagram by pressing [+] button on Diagram's Properties Profile tab. The aforementioned uml profile is stored in the supporting\_profiles folder (Figure 29).

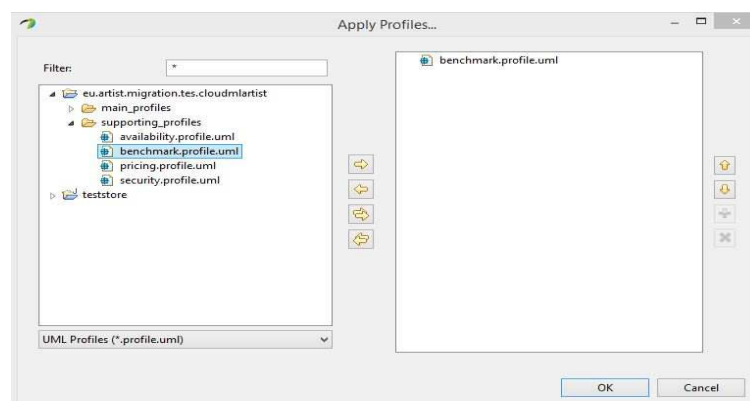


Figure 29: Benchmark profile is applied to the Diagram.

- Next step includes the Appliance of the *benchmark* profile to the Amazon EC2 profile

by pressing [+] button on Properties Profile tab. To be more precise DaCapoResult, FilebenchResult and YCSBResult Stereotypes from benchmark profile should be applied to one of the IaaSInstanceType Stereotypes of Amazon EC2 profile. For instance, we can follow this process for M1MediumInstance Stereotype ().

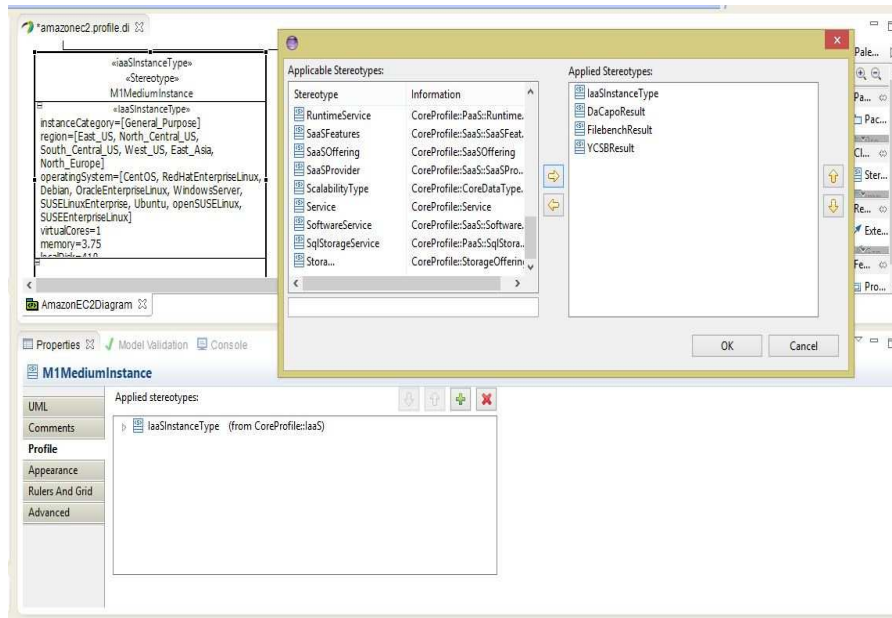


Figure 30: Benchmark profile is applied to the Amazon EC2 profile.

- Finally, it is possible to assign the respective average values from the benchmarking process to the properties defined in the M1MediumInstance stereotype. The average values can be retrieved from the local database and placed to the respective placeholder of the uml profile in an automated way by using the Model Updater prototype that is described in Section 8 and 9 extensively.

## 5 Benchmarking Tools Delivery and Usage

The Benchmarking tool is public and the Git source repository is on GitHub at the following link:

```
https://github.com/artist-project/ARTIST/tree/master/source/Tooling/pre-migration/benchmarking%20controller
```

- The structure of the delivered artefact is simple: it is possible to identify four main folders and three files.
- `benchmarks_folder`: such folder contains the workloads' configuration files and scripts useful for the execution
- `cloud_providers`: such folder contains an example of provider's configuration; values as the *image* to use or the *platform*, will define the target environment object of benchmark
- `src`: such folder contains the core source code
- `test`: this folder contains a single file that allows the user to choose between the main function to execute.
- `LICENSE`: such file contains information about the Apache 2.0 license
- `README`: such file contains the characteristics of this version of the released tool

### 5.1 Package information

Concerning the benchmarking tools used, in most cases they have been re-packaged (respecting licenses constraints of the single tools) and made available through the ARTIST repository.

In order to address installation requirements on all platforms, packages have been created in three formats: *rpm* (for RedHat-based platforms), *deb* (for Debian-based platforms) and *tar.gz* for all the others platforms (e.g. Windows). Although it required an initial effort to re-package the tools, this makes it much easier for users to install and use benchmarking tools in the context of ARTIST project. More information about generation and maintenance of packages for benchmarking tools are available in Section 5.

### 5.2 Installation instructions

Installation of benchmarking tools is highly simplified by the fact that these tools have been re-packaged to meet format of installation packages of most common platforms.

For example in the following we describe installation procedure for a CentOS machine:

1. access to target machine previously provisioned by the cloud provider.
2. add ARTIST Repository to machine' software repositories.

```
cd /etc/yum.repos.d/
```

```
wget http://etics.res.eng.it/repository/pm/registered/repomd/name/artist-benchmarking/etics-registered-build-by-name.repo
```

Now we will be able to use the typical yum commands (e.g., *search* or *install*) to manage ARTIST packages (e.g Figure 31). All software released by ARTIST will be installed in the prefix */opt/artist*.

```
[root@localhost yum.repos.d]# yum install artist.filebench.x86_64
Loaded plugins: fastestmirror, presto
Loading mirror speeds from cached hostfile
 * base: mirrors.prometeus.net
 * epel: fr2.rpmfind.net
 * extras: mirrors.prometeus.net
 * updates: mirrors.prometeus.net
Setting up Install Process
Resolving Dependencies
--> Running transaction check
--> Package artist.filebench.x86_64 0:0.0.0-0.centos6 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                        Arch      Version              Repository
=====
Installing:
artist.filebench               x86_64    0.0.0-0.centos6      ETICS-name-artist-dev

Transaction Summary
=====
Install      1 Package(s)

Total download size: 115 k
Installed size: 339 k
Is this ok [y/N]:
```

Figure 31: Installation procedure of benchmarking tools

For platforms not directly supported (Redhat- and Debian-based), it is still possible to use the *tar.gz* package to install software.

## 5.3 User Manual

Concerning the Benchmarking Suite:

- it is possible to refer to this document and its future updates for installation and configuration.
- to properly run a benchmark using the third-party tools, users can refer directly to the user's documentation of the tools usually available on-line on the tools homepage.

## 5.4 Licensing information

As far as the Benchmarking Suite is concerned, it has been released under Apache 2.0. The third-party benchmarking tools repackaged in a standard format and stored in the ARTIST Repository are open source and released under licenses that allow repackaging and redistribution.

Next table lists licenses for each third-party tool redistributed in the Benchmarking Suite release. Last column also lists, if any, the dependencies and correspondent licenses that the tool uses at run-time. These dependencies are not distributed through the Benchmarking Suite but downloaded from other software repositories and/or distribution locations.

**Table 2: Benchmarking Suite Licenses**

Tool	License	Dependencies
YCSB	Apache 2.0	<ul style="list-style-type: none"> <li>MySQL (GPL 2.0)</li> <li>Cassandra (Apache 2.0)</li> <li>MongoDB (AGPL 3.0)</li> </ul>
Bonfire's Dwarfs	GPL	
Filebench	CDDL	
CloudSuite Data Caching	CloudSuite 2.0	<ul style="list-style-type: none"> <li>Memcached (BSD)</li> </ul>
DaCapo	Apache 2.0	

## 5.5 Download

Benchmarking Suite is available on the ARTIST Repository and accessible following instructions presented in section 5.2. An additional access point for browsing and downloading the third-party tools is the following link:

<http://etics.res.eng.it/repository/pm/registered/repomd/name/artist-benchmarking/index.html>

## 6 3ALib Implementation

### 6.1 Functional Description

The 3ALib Java library is an abstracted way of taking measurements regarding the availability of services, regardless of the supported provider on which they are deployed. Furthermore, the library in its final version will be able to calculate the actual levels of availability as defined in the respective providers SLAs (initially IaaS Compute level SLAs are supported), based on the latter's definition of the term (for more details on the differences between providers consult D7.1 [12]). An example of an IaaS level SLA is Amazon EC2 Compute SLA (<http://aws.amazon.com/ec2/sla/>). What is more, the library may check for which running instances of the user the SLA applies, based on the preconditions and necessary deployment options that are considered obligatory by the providers in order for an SLA to apply. So overall, the goals of the library are:

- inform users on why their services do not adhere to the SLA
- keep detailed logs of the availability of services
- calculate availability levels based on the providers definition and be used for compensation claims (together with the aforementioned logs)

In order to enable all this functionality, the user only needs to run the provided executable, after he has created a local configuration file (as described in Section 7) with the necessary fields. Alternatively, if the functionality needs to be included in a Java program, the user must only create one object of the type `AvailabilityAuditor` and set the properties for it from the configuration details and start the monitoring thread. Logs are provided with the detailed measurements of availability and are stored in a backend MongoDB database, that can be deployed anywhere. Login credentials for this DB are expected to be provided by the library user upon configuration. A separate DB system was selected (in contrast to local log files) for a variety of reasons:

- Better separation and decoupling of the functional parts of the system
- Ability to have a centralized repository of availability information from multiple users
- Possibility to run business intelligence queries or in general analytics over the results
- Enablement of exploitation scenarios in which a 3rd party service may offer the aforementioned capabilities and provide insights into users or Cloud providers

### 6.2 Technical Description

3ALib is a Java library, that uses Apache Jclouds API as a means to abstract from the specifics of Cloud providers APIs and data format. Furthermore, it implements specific drivers for each provider, in order to enforce the specific preconditions that are mentioned in their respective SLAs and the specific calculation of availability followed by each provider.

### 6.2.1 Prototype Architecture

The design of 3ALib appears in Figure 32. Following the analysis of the providers SLAs, as it appears in D7.1 [12], the common concepts and terms are abstracted to generic methods that are used in all providers. Specificities are included as separate methods for the enhancement of the library's generic and extensible nature.

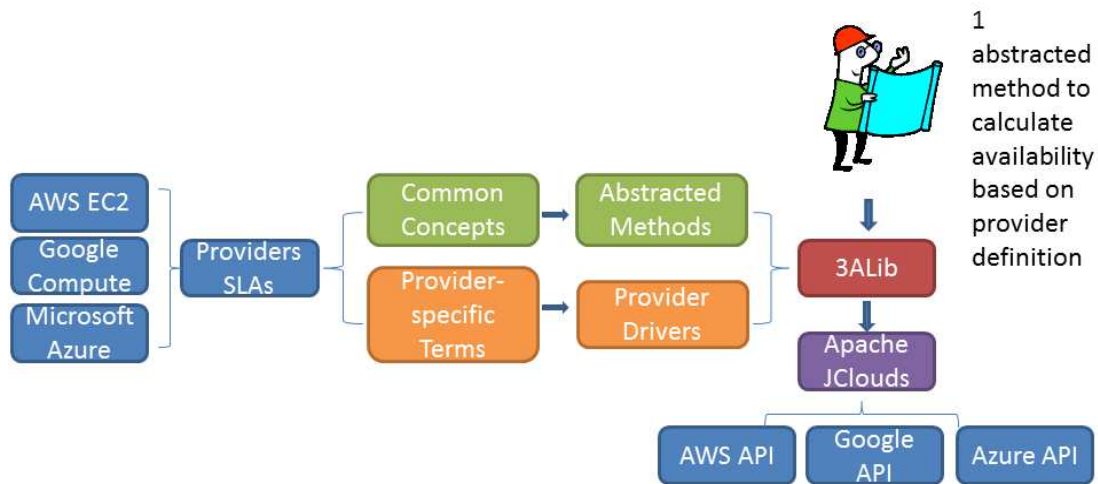


Figure 32: 3ALib General Design

### 6.2.2 Components Description

The class diagram of the library appears in Figure 33. The Main class may be used only in case the user needs to launch the executable directly, or as an example of how to instantiate an AvailabilityAuditor and configure it. The Availability Auditor uses one of the separate provider auditors based on the user's selection. These individual auditors implement the same interface (AuditingInterface) for consistency purposes, however they implement its methods based on each respective provider's logic and SLA. They also use abstracted methods like the JcloudsExecutorClient (used to retrieve information from the providers regarding the user's services and their status, based on Apache Jclouds), and the AbstractedAvailabilityLogger. The latter is responsible for getting the detailed status report and storing it. Furthermore, it is responsible for hiding the differences/variations in reported status messages from providers and including only a REACHABLE or UNREACHABLE status. The storage part is based on a MongoDB database backend that may be created on the user's side, or may be offered by a relevant service in a more exploitable scenario. The purpose of using a NoSQL solution is that it is more scalable and can be used for parallel processing of the logs, which is expected to be a time consuming process. Further analytics queries can also be implemented based on advanced business intelligence scenarios. The final calculation of the SLA adherence levels is performed by the AbstractedAvailabilityCalculator class, which is abstracted since all providers follow a similar formula for the calculation of availability, but with a number of differentiated parameters (such as the minimum continuous time interval for which a resource may be considered unavailable). Thus provided that the methods in this class take these parameters as arguments, they may be abstracted and be used for all cases. For a calculation to be performed, the user must enter the month of interest in date format. The PingReachabilityRuler is a class hiding the implementation of the reachability determination, which is implemented in the PingThread class. It was



designed in this way so that the way reachability is determined can be easily replaced in the future in case of a more appropriate method than pinging is considered. The StreamGobbler class is necessary to consume the output of the system command thread to execute the ping.

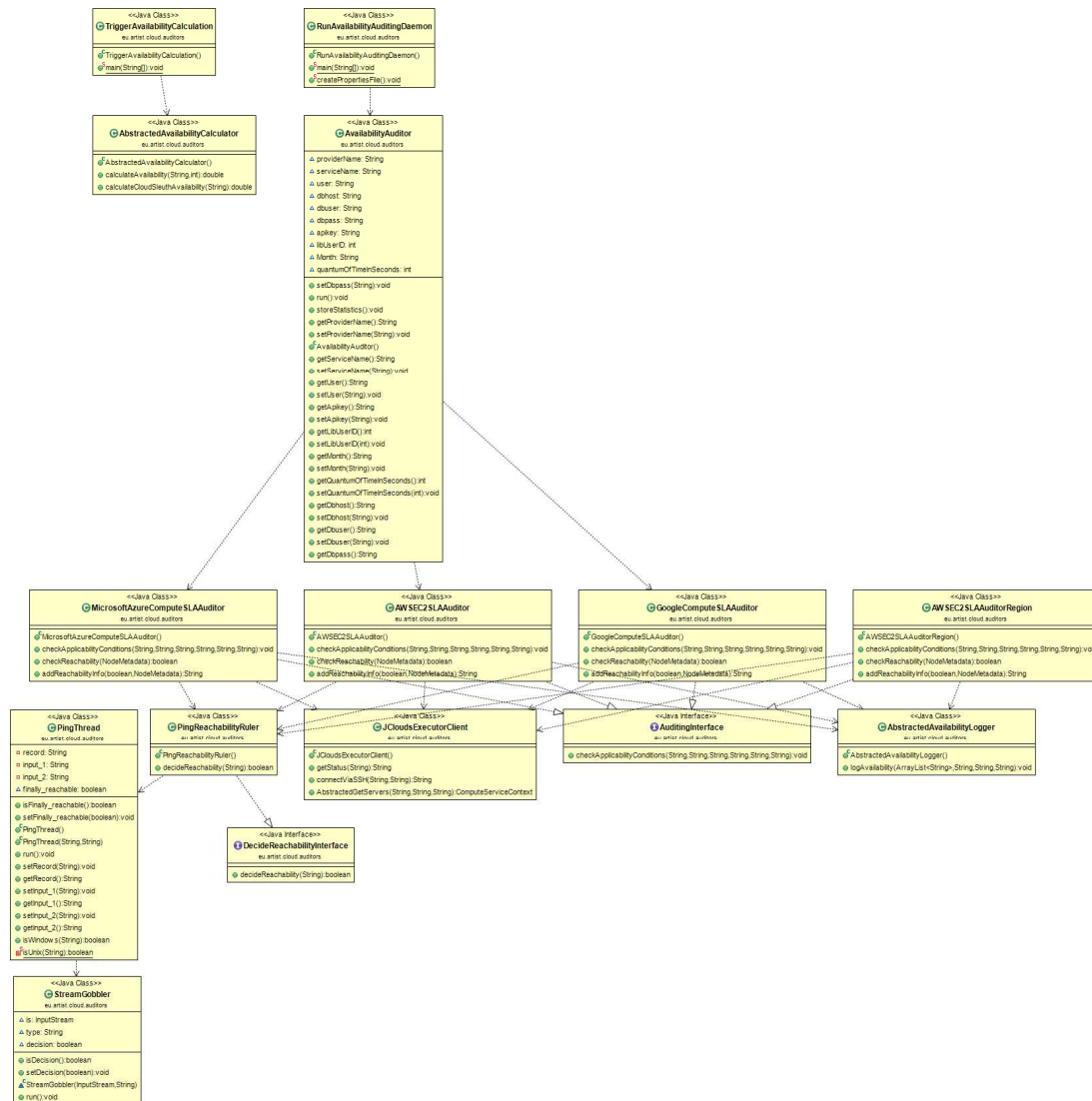


Figure 33:3ALib Class Diagram

### 6.2.2.1 Key assumptions and implementation considerations

Due to the ambiguous nature of legal contracts or the inability to define every technical detail in the latter, during development we have made the following assumptions/considerations:

During each loop, we check every time template IDs that are valid according to the provider preconditions and are admissible for the SLA. This happens because the valid IDs may vary over time due to user actions (e.g. starting/stopping a service instance affects the number of running VMs per template, which is a basic precondition for an SLA to apply). Only for these valid

template IDs we log the availability sample of that specific period, in order to keep only sensible and usable information and reduce the amount of data to be stored and later processed.

We assume an arbitrary interval of 5 minutes, for which consecutive timestamps of the logs are considered in the same interval. This practically means that if we find that 2 consecutive database entries have difference in timestamps larger than 5 minutes we assume that during this time the monitoring was stopped or other actions have deemed the SLA inapplicable for the specific template ID. Thus the second sample is not counted in any previously identified violation interval.

It is not determined in the SLAs of the providers whether the discount in case of violation applies to the entire account amount or to the part (valid template IDs) that is applicable to an SLA. However given that this is a dynamic list that may vary over time and that providers report the overall amount for a type of resource, calculating the potential discount (in case of partial discount) can be a very challenging, if not impossible (due to lack of necessary information), task.

One key consideration is the incorporation of checks regarding connectivity at the client side. This is extremely important since in case of poor connectivity at the client side, the samples may appear as unreachable. Thus once a sample is considered as unreachable, an extra check must be performed on the client (e.g. pinging of a well-known address like Google DNS at 8.8.8.8) in order to ensure that it is the service side that is unresponsive.

Another key consideration is the analysis per regions, since at that level the SLAs apply. Thus a separate layer of grouping results based on the region ID must be applied.

The main aspect mentioned by all the providers is that the resources need to be reachable. This is not completely determined as to what it means (could be based on pinging for example, ssh connection etc.). In our implementation this determination of connectivity will be decided via a separate class, that can be altered or adapted at any time based on how the connectivity validation is decided. The possibility to create an abstracted method (based on Jclouds) in order to check via ssh the connection ability will be investigated, however there may be some implications for the resources that have been created without the usage of Jclouds. Thus we followed the generic approach of pinging. However, in order for this to apply, one must have properly configured the VM instances to allow ICMP traffic, so that the ping messages can reach the resources.

Special attention must be given also to the way the results are retrieved in the calculation of availability. Small differences in implementation may result in significant delays in the query processing. For example, cursor count should be stored in a local variable rather than computed in each for loop, since in the MongoDB driver case, it is not included in any metadata but is calculated each time from the resultset.

### 6.2.3 Technical Specification

3ALib is a java based library (Java 7). It also has a strong dependency from Apache Jclouds library (1.7), for abstracting from the Cloud providers API and uses MongoDB 2.4.4 as a backend for storing the analytical logs for availability. It also has dependencies from external jars. Indica-

tively, we use mongo 2.4 java client (for interaction with the DB backend), gson 2.2.4 (for creating gson objects for db storage), guava 16.0 and guice 3.0.

#### 6.2.4 Availability experiments on Amazon AWS EC2

In order to test the implementation of the tool, we executed a 1.5-month experiment against an Amazon EC2 deployed service, consisting of 2 virtual machines running on the North Virginia Datacenter. The two VMs were configured to be launched in different availability zones (us-east-1c and us-east-1b), in order to comply with the Amazon SLA prerequisites, and they were continuously running. The experiment included also other VMs consideration, in the Oregon Datacenter, that were used for other tasks and were also considered, that had more dynamic deployment aspects (were being started and stopped at arbitrary times). The experiment was executed from November 2014 to 15<sup>th</sup> of December 2014.

The results of the experiment appear in Table 3. From these it is evident that there is significant difference between the availability calculations based on the specific provider's SLA and the generic formula used by Cloudsleuth. It is evident that with only one extra unavailable sample in the Oregon case, the Cloudsleuth defined availability would indeed violate the 99.95% boundary and thus appear to have an SLA violation. However, because the unavailable samples are not consecutive (they are indeed isolated), the minimal 1 minute interval of continuous unavailability is not met, thus the provider-defined availability is at 100%.

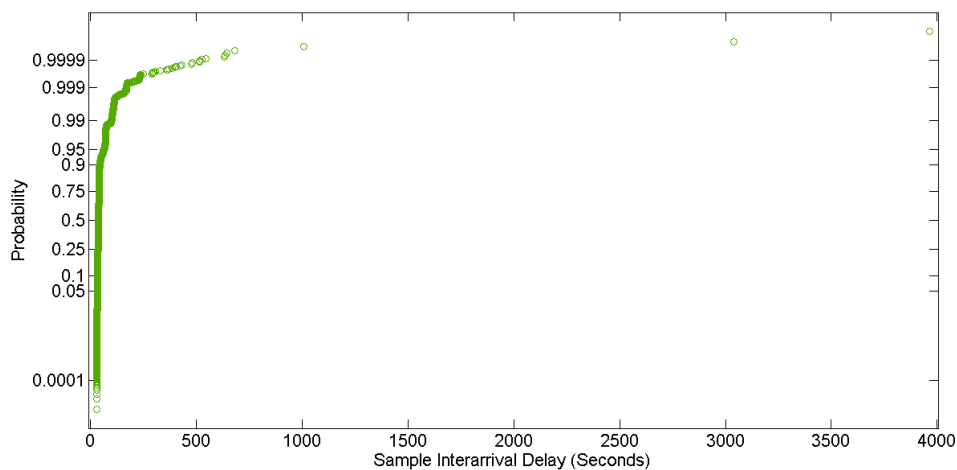
**Table 3: Availability Experiment Statistics**

	Data Center	Month	Samples	Unavailable Samples	AWS SLA Defined Availability	CloudSleuth Defined Availability
2 Testing VMs	N. Virginia (AWS EC2)	November 2014	58042	0	100%	100%
Arbitrary VMs	Oregon (AWS EC2)	November 2014	4424	2	100%	99.9547%
2 Testing VMs	N. Virginia (AWS EC2)	December 2014	31852	1	100%	99.9968%

As an indication, if the SLA defined availability was between 99 and 99.94%, our experiment (for the N. Virginia part) would cost 114 \$, if it was below 99% it would cost 88.9\$, whereas now the estimated cost is about 127 \$.

### ***Sampling Details and Statistics***

The probability of the samples in terms of interarrival times, obtained from subtracting the server-timestamped data as they were inserted in MongoDB at the client side, appears in Figure 34. This delay includes networking delay from the client location (Athens, GR) to the data center (N. Virginia, US), Amazon API delay for acquiring the server list and pinging delay for the two VMs, pinging delay for the external known locations that were used to ensure client connectivity and Mongo insertion delay. Potential loss of this client –side connectivity is probably responsible for the extreme cases of sample interarrival delays, e.g. in the range of almost 4000 seconds observed at the maximum value. Before each VM is pinged, the general connectivity is tested via 4 packets towards 8.8.8.8 (Google DNS) with an average time of 45 milliseconds per packet, resulting in 0.36 seconds of overhead for this step.



**Figure 34: Sample interarrival time probability distribution**

**Table 4: Sample Statistics for the Amazon SLA experiment (US East)**

Sample Statistics	Value
Overall Samples (US East)	89892
Minimum Sample Delay	29 seconds
Maximum Sample Delay	3965
Average Sample Delay	40,065 seconds

Standard Deviation	21.488 seconds
--------------------	----------------

One interesting aspect for future work would be the comparison of the Cloud providers API responses (to get the server list), since this seems to be the bottleneck of the overall process. This action needs to be performed before each sample, in order to ensure that no intermediate user action (such as shutting down one or more VMs) has resulted in a service setup that is not eligible for an SLA. Better response times would result in the ability to sample with higher frequency than the one observed in this case (average of almost 40 seconds per sample).

## 6.2.5 SLA metrics

### 6.2.5.1 SLA Adherence Levels

Given the differences in the providers availability definitions as described in D7.1, it is not feasible to directly compare provider-defined availability metrics, since these differ in definition. For this reason, it is more meaningful to either follow a more generic direct definition as done in CloudSleuth for example (Equation 1), or abstract to a more generic concept which is the SLA adherence level (Equation 2). This can be defined as the ratio of violated SLAs over the overall examined SLAs. Since SLA period is set to monthly cycles, this may be the minimum granularity of observation.

$$Availability = \frac{TotalSamples - UnavailableSamples}{TotalSamples}$$

Equation 1: Generic Availability Definition used in CloudSleuth

$$SLA\_Adherence = \frac{violatedSLAs}{overallObservedSLAs}$$

Equation 2: SLA Adherence Metric

Special attention must be given for cases that sampling is not continuous, indicating that the client did not have running services for a given period, applicable for an SLA. These cases must be removed from such a ratio, especially for the cases that no violations are examined in the limited sampling period, given that no actual testing has been performed. If on the other hand even for a limited period a violation is observed, then this may be included.

### 6.2.5.2 SLA Strictness metric

Besides SLA adherence, other metrics may be defined in order to depict the strictness of an SLA. As a prerequisite, we assume that the metric must follow a “higher is stricter” approach. Stricter implies that it is more difficult for a provider to maintain this SLA. In order to define such a metric initially one needs to identify what are the critical factors that may affect strictness levels. These factors are more highlighted in D7.1, based on the SLA analysis from existing public Cloud services. Factors should be normalized to a specific interval (e.g. 0 to 1) and appropriate levels for them may be defined. Indicative factors may include:

- Size of the minimum continuous downtime period (Quantum of downtime period  $q$ ). A higher size means that the SLA is more relaxed, giving the ability to the provider to hide outages if they are less than the defined interval. The effect of such a factor may be of a linear fashion (e.g.  $1-q$ ). Necessary edges of the original interval (before the normalization process) may be defined per case, based e.g. on existing examples of SLAs.
- Ability to use the running time of the services and not the overall monthly time, denoted by a Boolean variable  $t$ . This would be stricter in the sense that we are not considering the time the service is not running as available. The effect of such a factor may be of a Boolean fashion (0 false, 1 true)
- Percentage of availability that is guaranteed. Again this may be of a linear fashion, with logical intervals defined by the examined SLAs.
- Existence of performance metrics (e.g. response timing constraints). This may be a boolean feature  $x$ , however its values may be set to higher levels (0 or 5). The importance of this will be explained briefly.

The added value of such a metric may be in the case we have to deploy applications with potentially different characteristics and requirements (as one would expect). For example, having *soft real-time applications* would imply that we definitely need to have feature 4. Other less demanding applications may be accommodated by services whose SLAs are less strict. Thus suitable **value intervals** may be adjusted for each feature. If we use a value of 5 for the true case of feature 4, and all the other features are linked in such a manner that their accumulative score is not higher than 5, then by indicating a necessary strictness level of 5 implies on a numerical level that feature 4 needs definitely to be existent in the selected Cloud service.

Depending on the application types and their requirements and based on the metric definition, one can define categories of strictness based on the metric values that correspond to according levels (e.g. medium strictness needs a score from 2 to 3 etc.). It is evident that such a metric is based only on the SLA analysis and is static, if the SLA definition is not changed. Thus they can be easily included in the provider models defined in CloudML@ARTIST. The indicative formula for the case of equal importance to all parameters appears in Equation 3.

$$S = t + (1 - s_1 q) + s_2 p + x \text{ where}$$

$s_i$  : normalization factor for the continuous variables so that  $(s_1 * q) \in [0,1]$  and  $(s_2 * p) \in [0,1]$

$t \in \{0,1\}$ ,  $x \in \{0,1\}$

#### Equation 3: SLA Strictness definition formula

For the normalization intervals, for  $p$  we have used 99% and 100% as the edges, given that these were the ranges encountered in the examined SLAs. For  $q$  we have used 0 and 10 minutes as the edges. 0 indicates the case where no minimum interval is defined (thus abiding by the formula in **¡Error! No se encuentra el origen de la referencia.**) and 10 the maximum interval in examined Compute level SLAs. However there are larger intervals (e.g. 60 minutes) in terms of other layer SLAs (Azure Storage). The limit to 60 has been tried out in the  $q'$  case that is included in Table 5, along with the example of the other factors and the overall values of the SLA strictness metrics in the 3 examined public SLAs.

**Table 5: Indicative application of the SLA Strictness metric on existing public Cloud SLAs**

Provider/Service	$t$	$q$	$q'$	$p$	$x$	$S$	$S'$
Google Compute	0	5 (normalized:0.5)	5 (normalized:0.0833)	99.95 (normalized:0.5)	0	1	1.4167
Amazon EC2	0	1(normalized:0.1)	1(normalized:0.0167)	99.95 (normalized:0.5)	0	1.4	1.4833
Microsoft Azure Compute	1	1(normalized:0.1)	1(normalized:0.0167)	99.95 (normalized:0.5)	0	2.4	2.4833

An example of  $x$  not being 0 would be the Azure Storage SLA, where unavailability is also determined by response time limits to a variety of service calls.

## 7 3ALib Delivery and Usage

### 7.1 Package information

3ALib is distributed as a jar file, containing the source code and dependencies. It is also available as an Eclipse Java project, for extension of the code base.

### 7.2 Installation instructions

In order to run the jar file, only Java 7 needs to be installed in the target machine (and set as environment variable). In order to be installed as an Eclipse project, the standard import instructions of Eclipse need to be followed [13]. The Apache Jclouds project needs to be also included as a dependency, based on the instructions that can be found in [14]

For the database part, the steps are the following:

- Install MongoDB following the documentation instructions [15]
- Start MongoDB shell with the following instruction (optional):

```
mongo
```

- Create a new DB with the name 3alib (optional)

```
use 3alib
```

- Create a new collection (equivalent to table in SQL) with the name log\_samples (optional)

```
db.createCollection("log_samples");
```

The steps after installation are **optional**, since in Mongo if the specific db and collection are not created beforehand in the database, they will be once an initial attempt is made to write to them. Thus the first time one runs the 3alib tool, the relevant collection and db will be automatically created.

Since it is a NoSQL-based approach, no further schema or structure needs to be defined. The information is inserted in a row based logic throughout 3alib. The location of the DB along with the credentials needs to be provided to the configuration file of 3alib. An interesting GUI tool that we have used for viewing the contents of the database (and can be used also to perform various management operations) is MongoVUE [16]. An indicative screen-shot is provided below (Figure 35). The id column includes also the timestamp of inclusion, which is also the timestamp used for the comparison of time differences. In case we need to minimize logged data, we can also reduce the number of columns stored. Ideally we only need a limited number of the available information of each line (timestamped id, Status, template ID, availability region and resource ID).



_id	status	backendStatus	loginPort	publicAddresses	privateAddress...	imageId	hardware	os	id	type
537cbc3d0d64...	AVAILABLE	stopped	22	Array[0]	Array[0]	us-east-1/ami-...	{10 Keys}	{5 Keys}	us-east-1/i-6a...	NODE
537cbc3d0d64...	AVAILABLE	stopped	22	Array[0]	Array[0]	us-east-1/ami-...	{10 Keys}	{5 Keys}	us-east-1/i-9a...	NODE
53985e458e7f...	UNAVAILABLE	stopped	22	Array[0]	Array[0]	us-east-1/ami-...	{10 Keys}	{5 Keys}	us-east-1/i-6a...	NODE
53985e458e7f...	UNAVAILABLE	stopped	22	Array[0]	Array[0]	us-east-1/ami-...	{10 Keys}	{5 Keys}	us-east-1/i-9a...	NODE
53986a34c4b7...	UNAVAILABLE	stopped	22	Array[0]	Array[0]	us-east-1/ami-...	{10 Keys}	{5 Keys}	us-east-1/i-6a...	NODE
53986a34c4b7...	UNAVAILABLE	stopped	22	Array[0]	Array[0]	us-east-1/ami-...	{10 Keys}	{5 Keys}	us-east-1/i-9a...	NODE
53986af8eefec...	UNAVAILABLE	stopped	22	Array[0]	Array[0]	us-east-1/ami-...	{10 Keys}	{5 Keys}	us-east-1/i-6a...	NODE
53986af8eefec...	UNAVAILABLE	stopped	22	Array[0]	Array[0]	us-east-1/ami-...	{10 Keys}	{5 Keys}	us-east-1/i-9a...	NODE
53986bc52994c...	UNAVAILABLE	stopped	22	Array[0]	Array[0]	us-east-1/ami-...	{10 Keys}	{5 Keys}	us-east-1/i-6a...	NODE
53986bc52994c...	UNAVAILABLE	stopped	22	Array[0]	Array[0]	us-east-1/ami-...	{10 Keys}	{5 Keys}	us-east-1/i-9a...	NODE
53986c8a0eefc...	UNAVAILABLE	stopped	22	Array[0]	Array[0]	us-east-1/ami-...	{10 Keys}	{5 Keys}	us-east-1/i-6a...	NODE
53986c8a0eefc...	UNAVAILABLE	stopped	22	Array[0]	Array[0]	us-east-1/ami-...	{10 Keys}	{5 Keys}	us-east-1/i-9a...	NODE
53986d7c838ec...	UNAVAILABLE	stopped	22	Array[0]	Array[0]	us-east-1/ami-...	{10 Keys}	{5 Keys}	us-east-1/i-6a...	NODE
53986d7c838ec...	UNAVAILABLE	stopped	22	Array[0]	Array[0]	us-east-1/ami-...	{10 Keys}	{5 Keys}	us-east-1/i-9a...	NODE
53986f91aa8fc...	UNAVAILABLE	stopped	22	Array[0]	Array[0]	us-east-1/ami-...	{10 Keys}	{5 Keys}	us-east-1/i-6a...	NODE

Figure 35: 3ALib back-end DB view

## 7.3 User Manual

### 7.3.1 Configuration

3ALib includes a configuration file (3alibconfig.properties) that is automatically stored in the top level folder of the project. The file contains critical information on numerous issues such as provider keys, IP for the log database etc. The user must edit this file in order to include their own specific information. The form of the file is as follows:

```
#Mon May 12 20:34:05 EEST 2014
```

```
DBUser=
```

```
DBKey=
```

```
APIkey=YOUR_API_KEY
```

```
user=YOUR_USER_KEY
```

```
databaseIP=127.0.0.1
```

```
ProviderName=aws
```

```
ServiceName=-ec2
```

Figure 36: 3ALib Configuration file

Values for APIkey and user are given following the authentication methods of Cloud providers. DatabaseIP is the IP of the server that contains the MongoDB instance for storing logging information and DBuser and DBkey the access credentials for this DB. Provider and service names include the valid names as used by the library. For the three supported providers this is summarized in the Table 6.

**Table 6: 3ALib accepted provider names**

Provider	Provider Name	Service Name
Amazon EC2	aws	-ec2
Microsoft Azure	microsoft	-azure
Google Compute Engine	google	-compute-engine

The configuration file can be edited via any standard editor or created via code, as in the following example:

```
public static void createPropertiesFile(){

    Properties prop = new Properties();

    OutputStream output = null;

    try {

        output = new FileOutputStream("3alibconfig.properties");

        // set the properties value

        //change them also in properties file

        //can be deleted from here

        prop.setProperty("databaseIP", "192.168.56.101");

        prop.setProperty("user", "YOUR_USER");

        prop.setProperty("APIkey", "YOUR_API_KEY");

        prop.setProperty("ProviderName", "aws");

        prop.setProperty("ServiceName", "-ec2");

        prop.setProperty("DBuser", "YOUR_DB_USER");

        prop.setProperty("DBKey", "YOUR_DB_KEY");

        // save properties to project root folder

        prop.store(output, null);

    } catch (IOException io) {

        io.printStackTrace();

    } finally {

        if (output != null) {
```

```
        try {  
            output.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}  
}  
}
```

### 7.3.2 Executions

3ALib can be executed either as an individual jar file or used in a per case base inside existing code. In order for the latter to happen the following piece of code may be used:

```
InputStream input = null;  
  
Properties prop2 = new Properties();  
  
try {  
    input = new FileInputStream("3alibconfig.properties");  
  
    // load a properties file  
  
    prop2.load(input);  
  
    //abstracted part-differences are hidden inside the implementation //of each Auditor  
  
    AvailabilityAuditor thisauditor=new AvailabilityAuditor();  
  
    thisauditor.setProviderName(prop2.getProperty("ProviderName"));  
  
    thisauditor.setServiceName(prop2.getProperty("ServiceName"));  
  
    thisauditor.setUser(prop2.getProperty("user"));  
  
    thisauditor.setApikey(prop2.getProperty("APIkey"));  
  
    thisauditor.setDbhost(prop2.getProperty("databaseIP"));  
  
    thisauditor.run();  
  
} catch (IOException ex) {
```

```
        ex.printStackTrace();

    } finally {

        if (input != null) {

            try {

                input.close();

            } catch (IOException e) {

                e.printStackTrace();

            }

        }

    }

}
```

For calculating the availability in a given interval, the user will be prompted to enter the dates in the following format (Figure 37):

```
Enter start year
2014
Enter start month
1
Enter start day of month
1
Enter stop year
2014
Enter stop month
2
Enter stop day of month
-
```

Figure 37: 3ALib Availability Calculation Interval Definition

## 7.4 Licensing information

3ALib is released under the Apache License v2.0.

## 7.5 Download

3ALib is available for download from the Artist Repository:

<https://github.com/artist-project/ARTIST-Tooling/tree/master/migration/target-environment-specification/ArtistAuditor>

## 8 Model updater Implementation

### 8.1 Functional description

Model Updater prototype is a supporting tool which is used in order to populate and update the performance profiles of CloudML@ARTIST framework stored in ARTIST repository, with the respective values from the benchmarking process. As already mentioned the overall benchmark measurement results are stored in the internal (WP7) Raw Data DB in order to avoid extra overloading the uml profiles and keep the model as simple as possible in the ARTIST repository.

The main purpose of the Model Updater is to run periodically and refresh the profiles with the latest average performance scores by keeping the CloudML@ARTIST updated at regular intervals. In order to do this, the Model Updater is responsible for establishing a database connection, executing specific MySQL queries in order to calculate the average scores for each generic benchmark category. The previous performance information is transformed in an appropriate data structure (java objects) in order to be inserted in the included placeholders of a cloud provider profile which is retrieved from the ARTIST repository. After having the updated cloud provider profile, Model Updater restores it in the ARTIST repository. The Model Updater process appears in Figure 38. Finally, it is worth mentioning that this prototype facilitates the aforementioned procedures by providing an automated way and relieving the user from the manual population of the profiles avoiding the possibility of inserting incorrect score values.

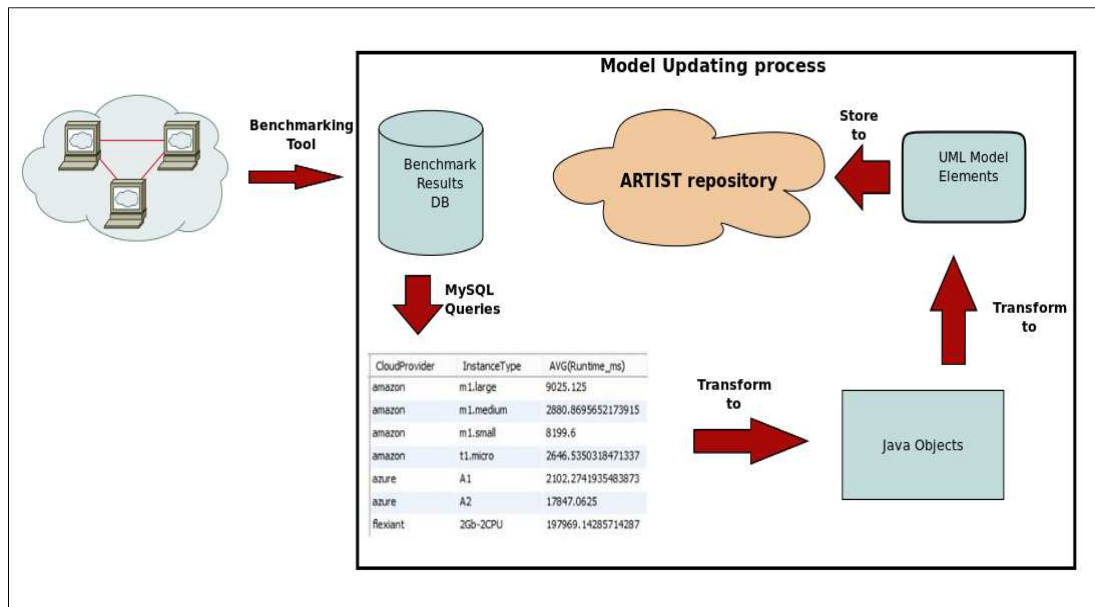


Figure 38: Model Updater process

## 8.2 Technical description

Model Updater prototype consists of three components, the DB Agent which is responsible for establishing database connection and executing MySQL queries, the Benchmark results data model which provides the appropriate data structure (java objects) of performance scores and the Model parser which is responsible for the interaction with the UML profiles. A detailed analysis for each component is presented in the following subsections.

### DB Agent

DB Agent is responsible for accessing the database and performing the required MySQL queries in order to obtain the average values coming from benchmark results. So, for every instance type of each provider, there is a set of benchmark tests performed each of which comes with a set of measured properties. This component connects to the database, performs the queries and stores the results in memory following the data structure provided by the results data model component.

### Benchmark results data model

This component provides the data structure used for data exchange between the dbAgent and the model parser. The abstract class BenchmarkResults is defined as shown in Figure 39.

```
public abstract class BenchmarkResults{

    protected String provider;
    protected String instanceType;
    protected ArrayList<String> results;

    public abstract List<String> getPropertyNames();
    public abstract String getBenchmarkType();

    public boolean setInstanceType(String instanceType) {
    }

    public boolean setProvider(String provider) {
    }

    public void addResult (Double result){
    }

    public String getProvider() {
    }
```

Figure 39: Abstract class BenchmarkResults

Other less important methods are also included. A set of classes extend this abstract class and concretize the abstract methods in such way so that:

1. getPropertyNames() returns the names of the properties for each benchmark test type exactly as these names appear on the Benchmark profile.
2. getBenchmarkType() returns the stereotype corresponding to this specific benchmark.

Two important methods for information exchange between the database and the models are the setInstanceType and setProvider methods. These two methods, get the string value as it appears in the database, and sets the corresponding Java class attribute with the string value as it appears in the model. In other words, these methods actually perform a transformation

### Model Parser

This component is designed in order to undertake the task of interacting with the UML profiles of CloudML@ARTIST. More precisely, it provides the methods for accessing the corresponding models, detecting the model elements and stereotypes which are to be updated and finally changing and storing the values of properties following the updates coming from the database. The whole task is about updating the CloudML@ARTIST metamodels from the point of view of benchmark profile applied on the providers' profiles. So, upon receiving the requests from the dbagent, the model parser is triggered and performs the requested changes.

The type of information given to the parser is:

- name of provider
- name of benchmark type
- type of instance type upon the benchmark test has been performed
- set of properties measured for this instance and this benchmark type
- set of values corresponding to these properties

The parser apprehends this information in modelling language terms:

- the name of the UML file to be accessed
- the name of the benchmark stereotype applied on an "InstanceType" stereotype of the above provider's profile
- the name of this instance stereotype
- the set of properties of the benchmark stereotype to be accessed
- the set of values to be stored

#### 8.2.1 Technical Specification

The Model Updater is based on open source software and it has been developed using Eclipse environment. In particular we are using the following java libraries for releasing the current components:

- UML2
- mysql-connector-java-5.1.34-bin.jar (GPL-2.0 License)
- guava-18.0.jar (Apache 2.0 License)

## 9 Model Updater Delivery and Usage

### 9.1 Package information

Model Updater is distributed as a jar file, containing the source code and dependencies. The file structure of the delivery package is shown in Figure 40.

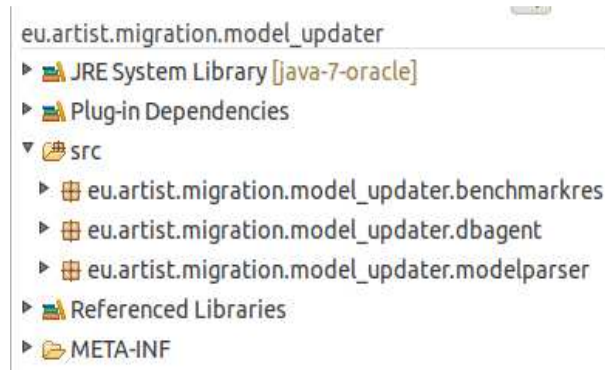


Figure 40: Model Updater packages

### 9.2 Installation instructions

#### Installation Instructions

- Install Java 7v (or later), if not already installed in your system.
- Download the runnable jar file from the ARTIST public repository

### 9.3 User manual

#### Execution Instructions

Prerequisites: Before executing the tool, the users should make sure that they have:

1. created a txt file for the name mappings between the database and the models. The file must contain one section for provider names mapping and one section for instance types mapping. Each section must have a title, either “providers” or “instances”. A txt file sample is shown in Figure 41. After each section's title the names are written (first the name for the database, and afterwards the name of the provider following a space character).

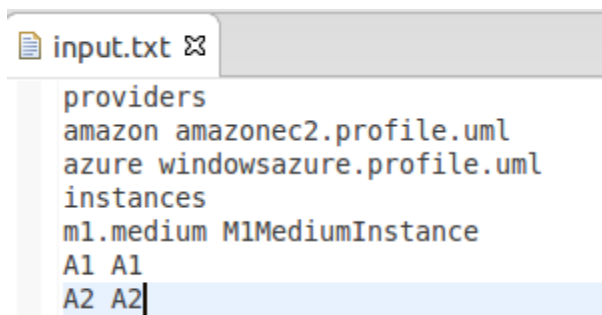


Figure 41: txt file sample



The two last lines stand for two different instance types of Microsoft Azure cloud platform. It is important to point out that, although these instance types have the same names for the database and the meta-models, they must be included in the input txt file, or else, they will be ignored during the model update. Bottomline, this file should include only the providers and the instances that are to be updated with the database values.

2. found the local directory of the CloudML@ARTIST metamodel.
3. the ip address, the username and the password for accessing the benchmark results database.

#### Execution

For executing the runnable jar the user should open a command line and type:

```
java -jar <runnable.jar.file> -i <ip address> -u <databaseusername> -p <database password> -f  
<input file with name mappings> -m <CloudML@ARTIST metamodel directory>
```

After the execution the new values should be stored in the model files.

## 9.4 Licensing information

Model Updater is offered under the Eclipse Public License (EPL).

## 9.5 Download

Model Updater is available for download from the Artist Repository at:

```
https://github.com/artist-project/ARTIST-Tooling/tree/master/migration/target-environment-  
specification/eu.artist.migration.model_updater
```

## 10 Conclusions

This document is delivered along with the final release of four prototypes in Work Package 7: a) the CloudML@ARTIST meta-model and b) the Benchmarking Suite c) the 3Alib library d) the Model updater

CloudML@ARTIST is an extension of the CloudML meta-model with several new stereotypes and relationships to describe aspects of cloud providers offering not taken into consideration by CloudML. In particular we added stereotypes to capture a) IaaS, PaaS and SaaS level aspects, b) performance aspect and c) pricing, security and availability aspects. CloudML@ARTIST is available as an UML meta-model ready to be imported in Eclipse Papyrus plug-in to be used as profile for cloud models. The concepts of the meta-model and of the supporting profiles have been used in practice to model provider features and capabilities. This process was continued in Y3 in order to identify improvement aspects, enrichment of information and extension of the produced models. In this work, input from WP9 from the usage of WP7 models has acted as a valuable feedback for features that should be reformatted, extended or removed. Furthermore, specific addons have been incorporated, including aspects of provider benchmarking and inclusion of the relevant information in the models.

Of particular interests are performance aspects since they will be a key criteria both in the target environment selection and migration evaluation processes. For this reason, we decided to release the ARTIST Benchmarking Suite: a tool-kit for managing the entire benchmarking process in ARTIST. Initial tests have been executed using these tools and feedback from the users has been used in order to update aspects of automatic execution and test setup. In this version, the ARTIST Benchmarking Suite, used to execute the tests, has been integrated with the Benchmarking GUI, for viewing the results in a dynamic selection manner.

Availability aspects are covered by 3Alib library; in particular, it is an abstracted way to measure the availability of services, regardless of the supported provider on which they are deployed. Such library will be able to calculate the actual levels of availability as defined in the respective providers SLAs, based on the latter's definition of the term. This tool is expected to be of great help for individual users in order to validate their SLAs. In this version we have extended the tool to include Google Compute Engine and we have contributed in the investigation of the incorporation of GAE services, and mainly the Google Datastore. Furthermore, we applied the 3aLib on a 1.5 month experiment on Amazon AWS monitoring, in order to investigate its robustness and the production of results.

Artefacts produced using the three prototypes are published in the ARTIST Repository and will be used as inputs by the others processes of a migration procedure. In particular, the business and technical feasibility (Work Package 5), forward engineering process (Work Package 9) and the testing and validation process (Work Package 11) will use the data. Furthermore, we have separated the average information (contained in the provider profiles) from the overall raw data, in order to keep the former lightweight and usable. The raw data are exposed through the benchmarking GUI to an interested party that is more focused in performance, and can make dynamic and configurable queries towards the raw data DB.

## References

- [1] G. Kousiouris and A. Evangelinou, "Artist Project - D7.2 - PaaS/IaaS Metamodelling Requirements and SOTA".
- [2] F. Sören, F. Fittkau and W. Hasselbring, Search-based genetic optimization for deployment and reconfiguration of software in the cloud, ICSE 2013, pp. 512-521.
- [3] D. Cosmo, Lienhardt, Treinen, Zacchiroli, Zwolakowski, Aeiche and Agahi, Automated Synthesis and Deployment of Cloud Applications, ASE2014.
- [4] H. Page, "CloudML," [Online]. Available: <http://cloudml.org/>.
- [5] "Electrical Engineering and Computer SciencesUniversity of California at Berkeley; Above the Clouds: A Berkeley View of CloudComputing," 2009.
- [6] Y. GitHub, "YCSB Getting Started," [Online]. Available: <https://github.com/brianfrankcooper/YCSB/wiki/Getting-Started>.
- [7] "Etics Home Page," [Online]. Available: <http://etics-archive.web.cern.ch/etics-archive/index.htm>.
- [8] "Bonfire Project Home Page," [Online]. Available: <http://www.bonfire-project.eu/>.
- [9] "CloudSuite Home Page," [Online]. Available: <http://parsa.epfl.ch/cloudsuite/cloudsuite.html>.
- [10] "Filebench Home Page," [Online]. Available: <http://sourceforge.net/apps/mediawiki/filebench/index.php?title=Filebench>.
- [11] "Filebench WML," [Online]. Available: [http://sourceforge.net/apps/mediawiki/filebench/index.php?title=Workload\\_Model\\_Language](http://sourceforge.net/apps/mediawiki/filebench/index.php?title=Workload_Model_Language).
- [12] "ARTIST Consortium, Deliverable D7.1 Definition and extension of performance stereotypes, ICCS/NTUA and other partners," March 2014.
- [13] "Eclipse Documentation for project import," [Online]. Available: <http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.platform.doc.user%2Ftasks%2Ftasks-importproject.htm>.
- [14] "JCloud Documentation," [Online]. Available: <http://jclouds.apache.org/documentation/devguides/using-eclipse/>.
- [15] "MongoDB Installation instructions," [Online]. Available: <http://docs.mongodb.org/manual/installation/>.
- [16] "MongoVUE Desktop application," [Online]. Available: <http://www.mongovue.com/>.
- [17] "Cloud4SOA Home Page," [Online]. Available: <http://www.cloud4soa.eu>.
- [18] "W3 Mathematical Modelling Language MathML," [Online]. Available: <http://www.w3.org/TR/MathML3/>.
- [19] T. Eilam, M. Kalantar, A. Konstantinou, G. Pacifici, J. Pershing and Agrawal, "Managing the configuration complexity of distributed applications in Internet data centers," *Communications Magazine, IEEE*, 2006.

- [20] "DeltaCloud Drivers," [Online]. Available: <http://deltacloud.apache.org/drivers.html>.
- [21] "JClouds What-is-Jclouds," [Online]. Available: <http://jclouds.incubator.apache.org/documentation/gettingstarted/what-is-jclouds/>.
- [22] "LibCloud Getting-Started," [Online]. Available: <http://libcloud.apache.org/getting-started.html>.
- [23] "Cobiacomm How to pick an ESB? An Enterprise Service Bus Evaluation Framework," [Online]. Available: <http://blog.cobia.net/cobiacomm/2012/04/18/how-to-pick-an-esb-an-enterprise-service-bus-evaluation-framework/>.
- [24] «Chris Haddad electing an enterprise service bus,» [Online]. Available: <http://de.slideshare.net/wso2.org/enterprise-use-case-selecting-an-enterprise-service-bus>.
- [25] «Fiorano,» [Online]. Available: [http://www.fiorano.com/docs/4\\_Comparision\\_Fiorano\\_BEAAquaLogic.pdf](http://www.fiorano.com/docs/4_Comparision_Fiorano_BEAAquaLogic.pdf).
- [26] "OpenLogic," [Online]. Available: <http://de.slideshare.net/OpenLogic/comparison-of-open-source-esb-solutions>.
- [27] "Manrivo," [Online]. Available: <http://www.manrivo.com/data/esbCompare061018.pdf>.
- [28] "MPhasis," [Online]. Available: <http://www.mphasis.com/pdfs/enterpriseservicebus.pdf>.
- [29] Y. Yahoo!, "Overview and Result," 2010.
- [30] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan and R. Sears, "Benchmarking Cloud Serving Systems with YCSB".
- [31] "Papyrus User's Guide," [Online]. Available: [http://wiki.eclipse.org/Papyrus\\_User\\_Guide](http://wiki.eclipse.org/Papyrus_User_Guide).
- [32] "Eclipse Indigo," [Online]. Available: <http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/indigosr2>.
- [33] "Amazon Amazon S3 Service Level Agreement," [Online]. Available: <http://aws.amazon.com/s3-sla/>.
- [34] "Yahoo! Home Page," [Online]. Available: <https://github.com/brianfrankcooper/YCSB/wiki>.
- [35] "A view from Berkeley," [Online]. Available: <http://view.eecs.berkeley.edu/wiki/Dwarfs>.
- [36] "CloudSuite Data Caching Home Page," [Online]. Available: <http://parsa.epfl.ch/cloudsuite/memcached.html>.
- [37] "Enterprise Integration Patterns Home Page," [Online]. Available: <http://www.eaipatterns.com/>.
- [38] "Cloud Vertical costs comparison and characteristics," [Online]. Available: [https://cloudvertical.com/cloud-costs#cloud\\_costs/index](https://cloudvertical.com/cloud-costs#cloud_costs/index).
- [39] Evangelinou, Galante, Kousiouris, Giammatteo and others, "Experimenting with application-based benchmarks on different Cloud providers via a multi-cloud execution and modeling framework," *Springer*, 2014.

## Appendix A – CloudML@Artist core profile

This section introduces the main extensions introduced in the CloudML@ARTIST meta-model (compared to the original CloudML meta-model). Extensions are described, grouped by functionalities, in separate sub-sections that present the context of the cloud characteristic analysed by the extensions and a rationale of the stereotypes' design.

Apart from the extensions, that can be applied in a selective way depending on the specific modelling needs, there is a small set of stereotypes at a higher level containing generic entities and properties that can be applied to any cloud provider, independently of the provider's nature (IaaS, PaaS or SaaS).

Next a picture of this set of stereotypes and a description of each one:

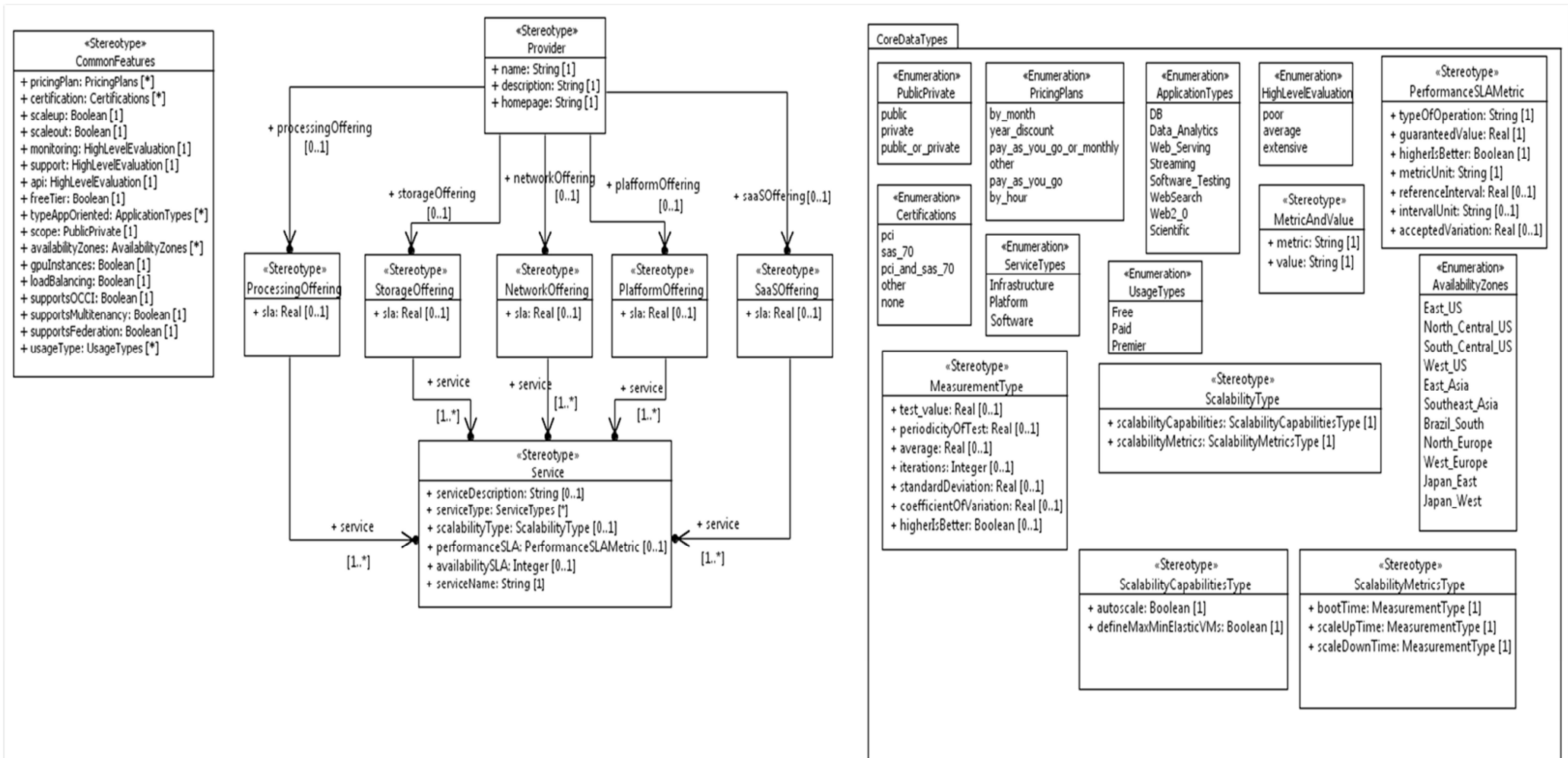
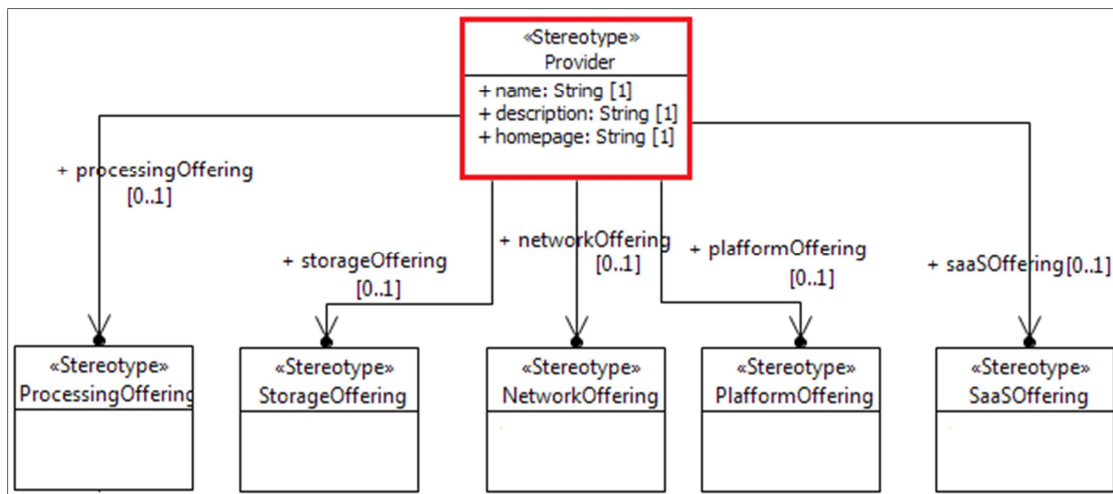


Figure 42: CloudML@Artist's high level stereotypes

## Provider



This stereotype is used to specify general information of the provider to be modelled, such as the name and homepage.

It is extended from stereotypes representing PaaS, IaaS and SaaS providers contained in the correspondent UML profiles (see next sections), so when modelling a provider, it is possible to reuse these properties specified at this higher level and to include some others related to the nature of the provider.

Next are described the properties contained in the stereotype.

Name	Type	Card.	Description
name	String	1	Unique identifying name for the provider
description	String	1	Provider description
homepage	URL	1	Provider homepage URL

### ProcessingOffering

This stereotype is used to categorize those services representing the processing capacities offered by the provider, such as number and speed of processors, processors architecture and so on.

### StorageOffering

This stereotype is used to categorize those services related with the storage capacities offered by the provider. As can be seen on sections related to IaaS and PaaS profiles (below in this appendix), these storage services can be described by different sets of properties depending mainly on whether they are infrastructure or platform related services.

### NetworkOffering

A network offering is a named set of network services a cloud provider can offer, such as:

- DHCP

- DNS
- Source NAT
- Static NAT
- Port Forwarding
- Load Balancing
- Firewall
- VPN

### PlatformOffering

This stereotype is used to categorize those services specifically offered by a paas provider, such as preinstalled application servers, application frameworks, programming languages and so on.

### SaaSOffering

This stereotype is used to categorize those services offered by a saas provider.

Next, on the section related to the SaaS profile, can be found a description of the services (SoftwareService entity) a SaaS provider can offer.

### CommonFeatures

«Stereotype» CommonFeatures
+ pricingPlan: PricingPlans [*] + certification: Certifications [*] + scaleup: Boolean [1] + scaleout: Boolean [1] + monitoring: HighLevelEvaluation [1] + support: HighLevelEvaluation [1] + api: HighLevelEvaluation [1] + freeTier: Boolean [1] + typeAppOriented: ApplicationTypes [*] + scope: PublicPrivate [1] + availabilityZones: AvailabilityZones [*] + gpuInstances: Boolean [1] + loadBalancing: Boolean [1] + supportsOCCI: Boolean [1] + supportsMultitenancy: Boolean [1] + supportsFederation: Boolean [1] + usageType: UsageTypes [*]

This stereotype offers the possibility to specify common features that can be applied in general to any cloud provider. Apart from this stereotype specified at a high level, there are also equivalent stereotypes at iaas, paas and saas profiles (see in next sections) extending from this one and allowing setting specific properties at those levels.

Next are described the properties contained in the stereotype.

Name	Type	Card.	Description
------	------	-------	-------------

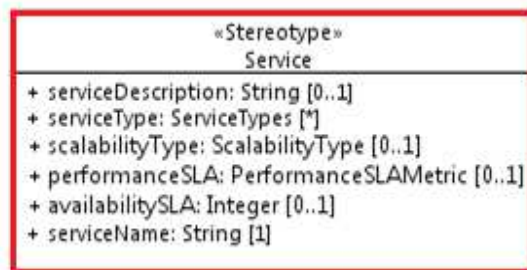


pricingPlan	PricingPlans	*	Type of pricing plans offered by a cloud provider. Possible values can be found on PricingPlans enumeration data type (see on data types section). In case it is necessary more detailed information regarding prices and discounts can be set by applying the cost uml profile (see on Cost profile section) when creating a specific provider's model.
certification	Certifications	*	Types of certifications owned by a provider. Possible values can be found on Certifications enumeration type.
scaleup	Boolean	1	Indicates if the provider offers the capacity to scale up (increasing resources on the same server)
scaleout	Boolean	1	Indicates if the provider offers the capacity to scale out (increasing resources by balancing the load among servers).
monitoring	HighLevelEvaluation	1	Monitoring capabilities offered by the provider. Values can be selected from HighLevelEvaluation enumeration type.
support	HighLevelEvaluation	1	This property gives an indication of the supporting level offered by the provider.  Values can be selected

			from HighLevelEvaluation enumeration type.
api	HighLevelEvaluation	1	Indicates in general terms if the api offered by the provider to access its infrastructure permits to perform a lot of operations or offers quite limited access.
freeTier	Boolean	1	Indicates if the provider offers a free tier or not.
typeAppOriented	ApplicationTypes	*	Type o applications to which it is oriented the provider. Values given by ApplicationTypes enumeration data type.
Scope	PublicPrivate	1	Indicates if the provider is public or private.
availabilityZones	AvailabilityZones	*	Geographical zones where the provider offers its infrastructure.
gpuInstances	Boolean	1	Indicates whether the provider offers gpu instances or not.
loadBalancing	Boolean	1	Indicates whether the provider offers load balancing capabilities or not.
supportsOCCI	Boolean	1	Indicates whether the provider supports occi (Open Cloud Computing Interfac) or not.
supportsMultitenancy	Boolean	1	Indicates whether the provider supports

			multitenancy or not.
supportsFederation	Boolean	1	Indicates whether the provider supports federation or not.
usageType	UsageTypes	*	Usage types offered by the provider. Possible values can be selected by using UsageTypes enumeration data type.

## Service



This stereotype is used to describe the main information related to a service that can be offered by a cloud provider. Every service included in the profiles representing paas, iaas and saas providers extends from this stereotype and includes (if necessary) some features specific to that type of provider.

Next are described the properties contained in the stereotype.

Name	Type	Card.	Description
serviceDescription	String	0..1	Brief description of the offered service.
serviceType	ServiceTypes	*	High level type of service. It could be iaas, paas or saas service
scalabilityType	ScalabilityType	0..1	Scalability related features that can be applied to a service.
performanceSLA	PerformanceSLAMetric	0..1	Service Level Agreement related to performance

			offered by the service.
availabilitySLA	Integer	0..1	Service Level Agreement related to availability offered by the service.
serviceName	String	1	Name of the service.

### Data types

Apart from this set of stereotypes described previously, there is also a set of high level data types that will be used extensively in CloudML@Artist whose main purpose is to specify lists of common “cloud world” related values that can be reused when creating models from CloudML@Artist.

Next, a view of the package containing these high level data types:

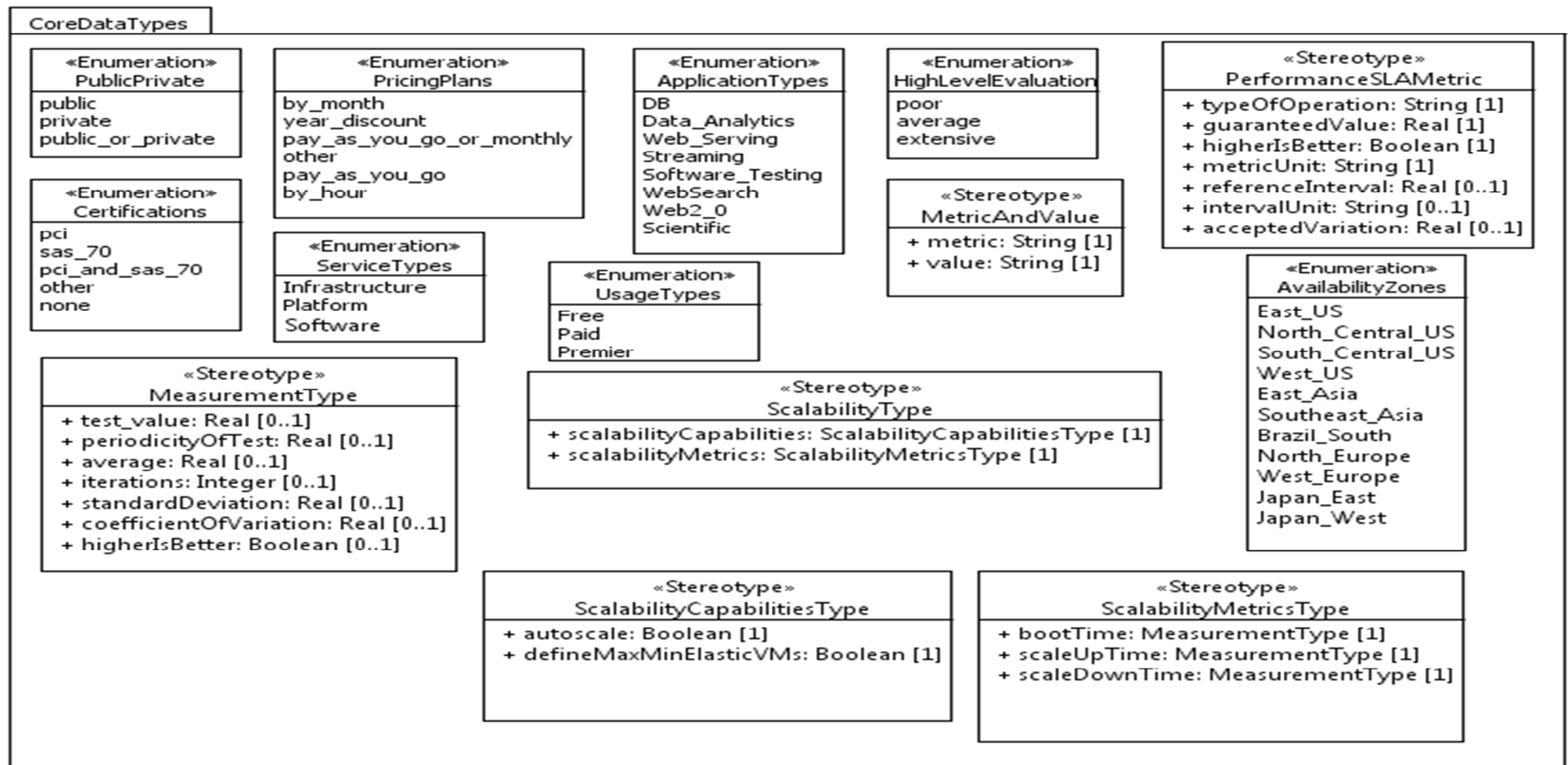


Figure 43: CloudML@Artist Core Data Types

## laaS

This section describes the laaS uml profile contained in CloudML@Artist meta-model.

This profile is composed of infrastructure related stereotypes and data types and allows (in conjunction with higher level stereotypes and data types already described) to model main characteristics of any laaS provider.

laaS providers' offering consists typically of a set of services classified in three categories: processing services, network services and storage services.

As can be seen in next picture, these services are represented by laaSProcessingService, laaSNetworkService and laaSStorageService stereotypes.

Following the picture can be found a complete description of the main entities contained in the profile.

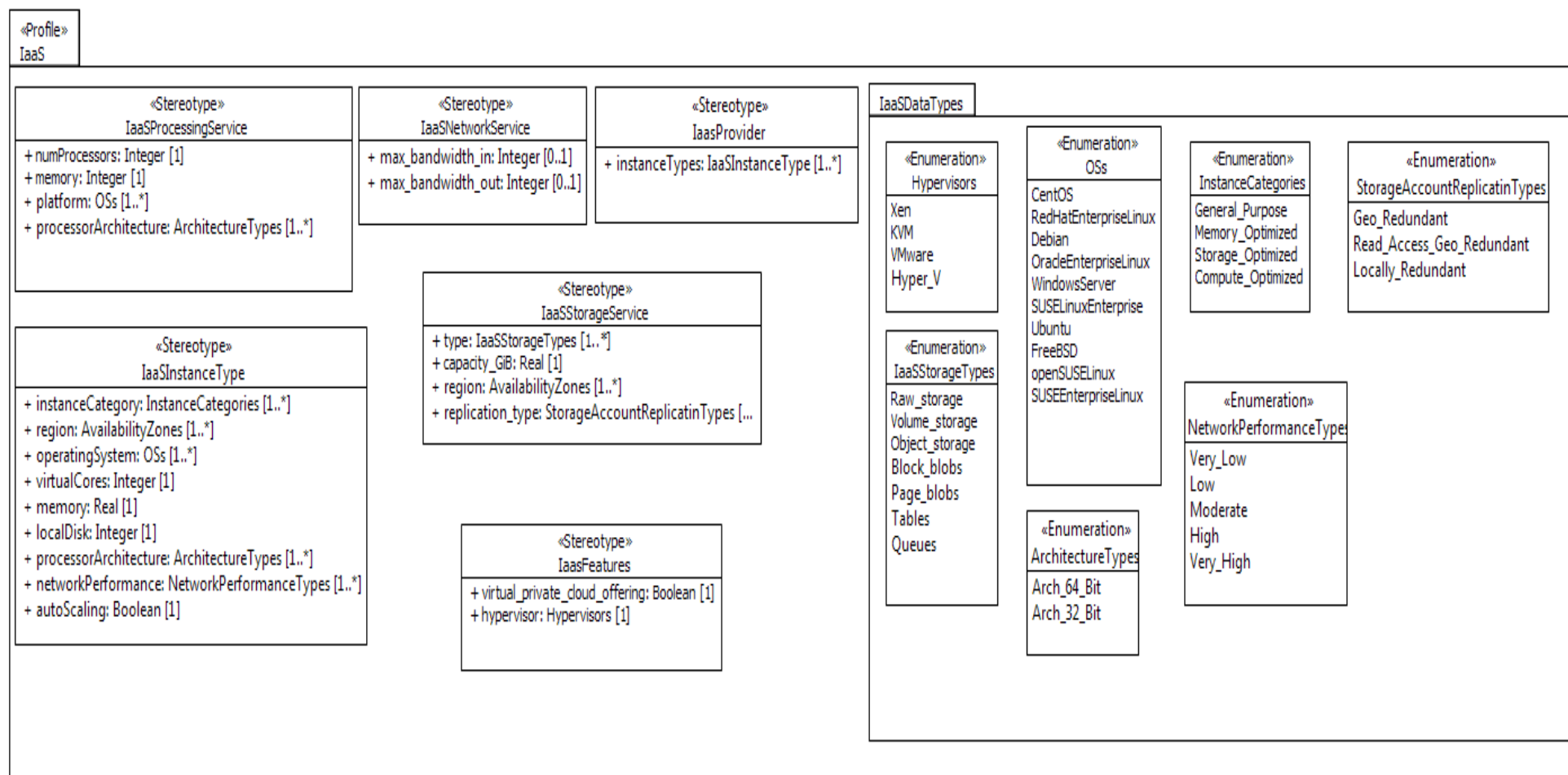
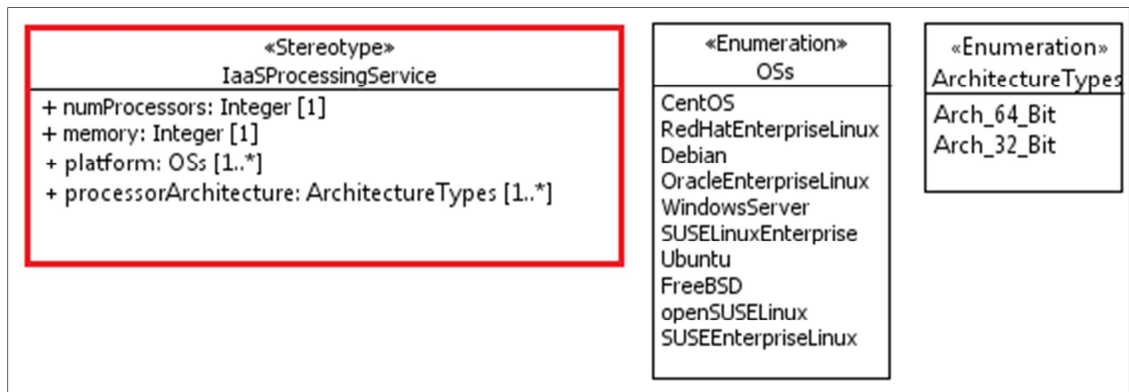


Figure 44: IaaS Stereotypes

## IaaSProcessingService



This stereotype allows modelling processing capabilities offered by an IaaS provider.

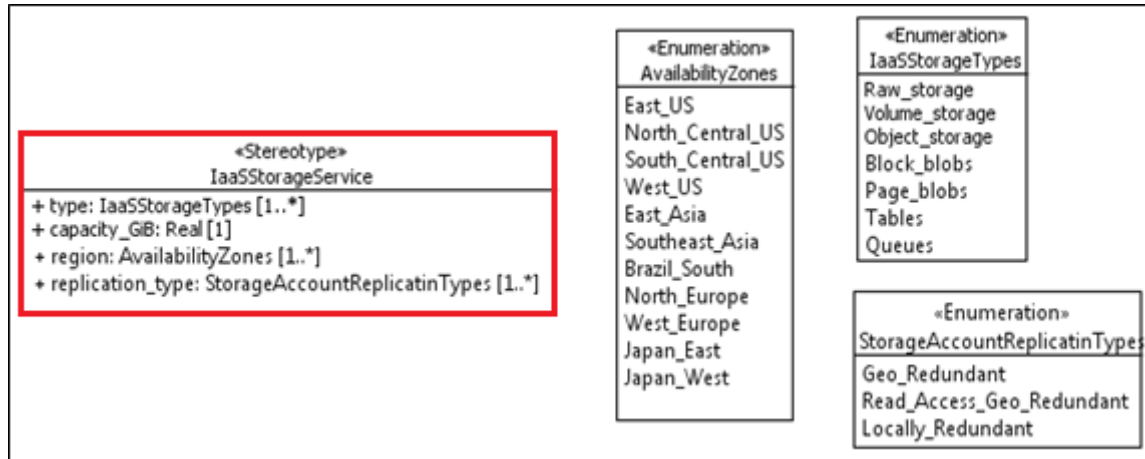
It extends from Service stereotype described in the beginning of this appendix.

Next are described the properties contained in the stereotype.

Name	Type	Card.	Description
numProcessors	Integer	1	Maximum number of processors the provider can offer.
memory	Integer	1	Maximum amount of RAM the provider can offer (GB)
platform	OSs	1..*	Relation of Operating Systems supported by the provider. Represented by the Enumeration type OSs (on the right side of the picture).
processorArchitecture	ArchitectureTypes	1..*	Property through which it is specified if the provider supports 64-bit architecture, 32-bit architecture or both. Uses ArchitectureTypes Enumeration type (on the right).



## IaaSStorageService

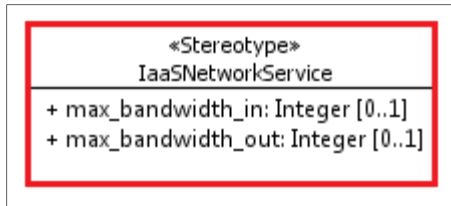


This stereotype allows modelling storage capabilities offered by an IaaS provider. As IaaSProcessingService it extends from Service stereotype.

Next are described the properties contained in the stereotype.

Name	Type	Card.	Description
type	IaaSStorageTypes	1..*	Relation of the different types of storage offered by the provider. Values can be selected by using IaaSStorageTypes Enumeration data type (on the right part of the picture).
capacity_GiB	Real	1	Maximum amount of storage the provider can offer (GB)
region	AvailabilityZones	1..*	List of regions where the data can be stored. Values can be selected by AvailabilityZones enumeration data type.
replication_type	StorageAccountReplicatinTypes	1..*	Types of replication of data the provider can offer. Values can be established by using StorageAccountReplicatinTypes entity.

## IaaSNetworkService

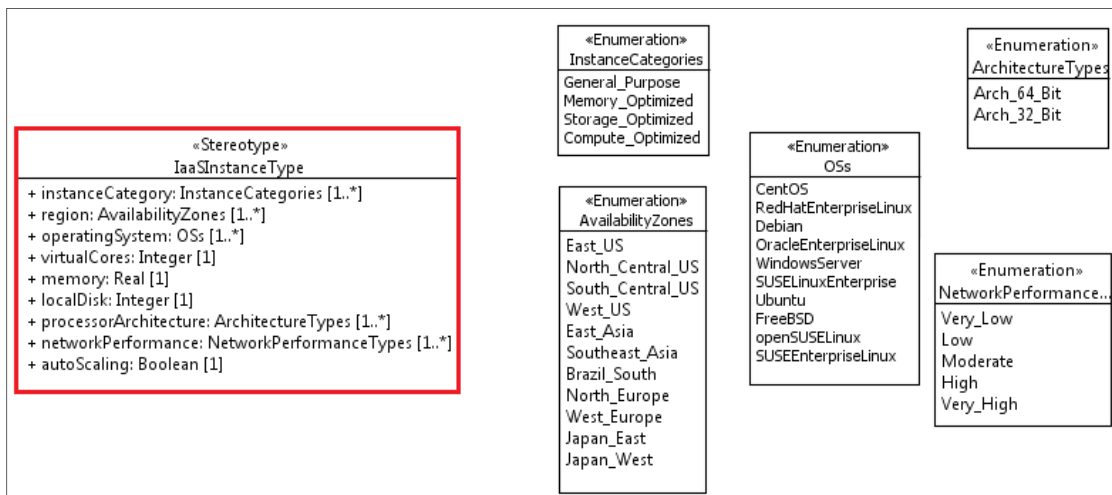


This stereotype allows modelling network services by an IaaS provider. It extends also from Service stereotype.

Next are described the properties contained in the stereotype.

Name	Type	Card.	Description
max_bandwidth_in	Integer	0..1	Maximum input bandwidth capacity offered by the provider (in Mbps)
max_bandwidth_out	Integer	0..1	Maximum output bandwidth capacity offered by the provider (in Mbps)

## IaaSInstanceType



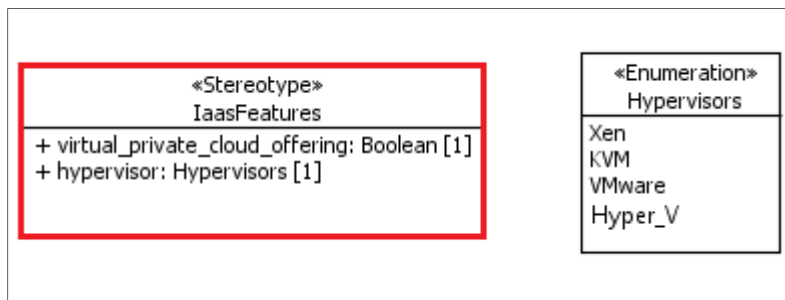
This stereotype allows modelling the characteristics of the instance types an IaaS provider can offer.

Next are described the properties contained in the stereotype.

Name	Type	Card.	Description
instanceCategory	InstanceCategories	1..*	Category of the instance, depending on the features offered. Possible values can be selected by using InstanceCategories enumeration type that can be found on the right part of the picture above.
region	AvailabilityZones	1..*	List of regions where the instance can be run. Values can be selected by AvailabilityZones enumeration data type
operatingSystem	OSs	1..*	Relation of Operating Systems supported by the instance. Represented by the Enumeration type OSs (on the right side of the picture).
virtualCores	Integer	1	Number of cores offered by the instance.
memory	Real	1	Maximum amount of RAM the instance can offer (GB).
localDisk	Integer	1	Capacity of the disk at instance level (in GB).
processorArchitecture	ArchitectureTypes	1..*	Property through which it is specified if the instance supports 64-bit architecture, 32-bit architecture or both. Uses ArchitectureTypes Enumeration type (on the right).
networkPerformance	NetworkPerformanceTypes	1..*	This property specify the type of network performance offered by the instance at high level. Possible values are

			detailed in NetworkPerformanceTypes enumeration type.
autoScaling	Boolean	1	Boolean value to specify if the provider offers autoscaling capabilities or not.

## IaaSFeatures



This stereotype extends from generic CommonFeatures stereotype and allows specifying specific IaaS features.

Next are described the properties contained in the stereotype.

Name	Type	Card.	Description
virtual_private_cloud_offering	Boolean	1	Property to specify if the IaaS provider can offer or not a virtual private cloud (VPC).
hypervisor	Hypervisors	1	Property used to specify the hypervisor used by the IaaS. An hypervisor is a piece of software, firmware or hardware that creates and runs virtual machines. The possible values can be obtained from Hypervisors enumeration type.

## PaaS

This section introduces a proposal for extension to CloudML@ARTIST, in order to incorporate

PaaS concepts into this meta-model, aligned with the IaaS description already available in CloudML. At this stage the PaaS extension, built as an UML profile describes the main PaaS stereotypes and their relationships, linking some of them to available CloudML stereotypes. By the use of this profile (in conjunction with the rest of stereotypes and entities defined at the higher level in CloudML@Artist described previously) it is possible to model the main characteristics of any PaaS provider such as Google App Engine for example. In the elaboration of this PaaS extension, we have also borrowed some ideas and concepts from Cloud4SOA Semantic Model for PaaS [8]. **¡Error! No se encuentra el origen de la referencia.** depicts the suggested PaaS extension. Key concept is the notion of CloudOffering, which generalizes the one borrowed from PaaS domain [8]. A CloudOffering is a package of services a CloudProvider offers to users for different purposes, depending on the layer of the Cloud stack.

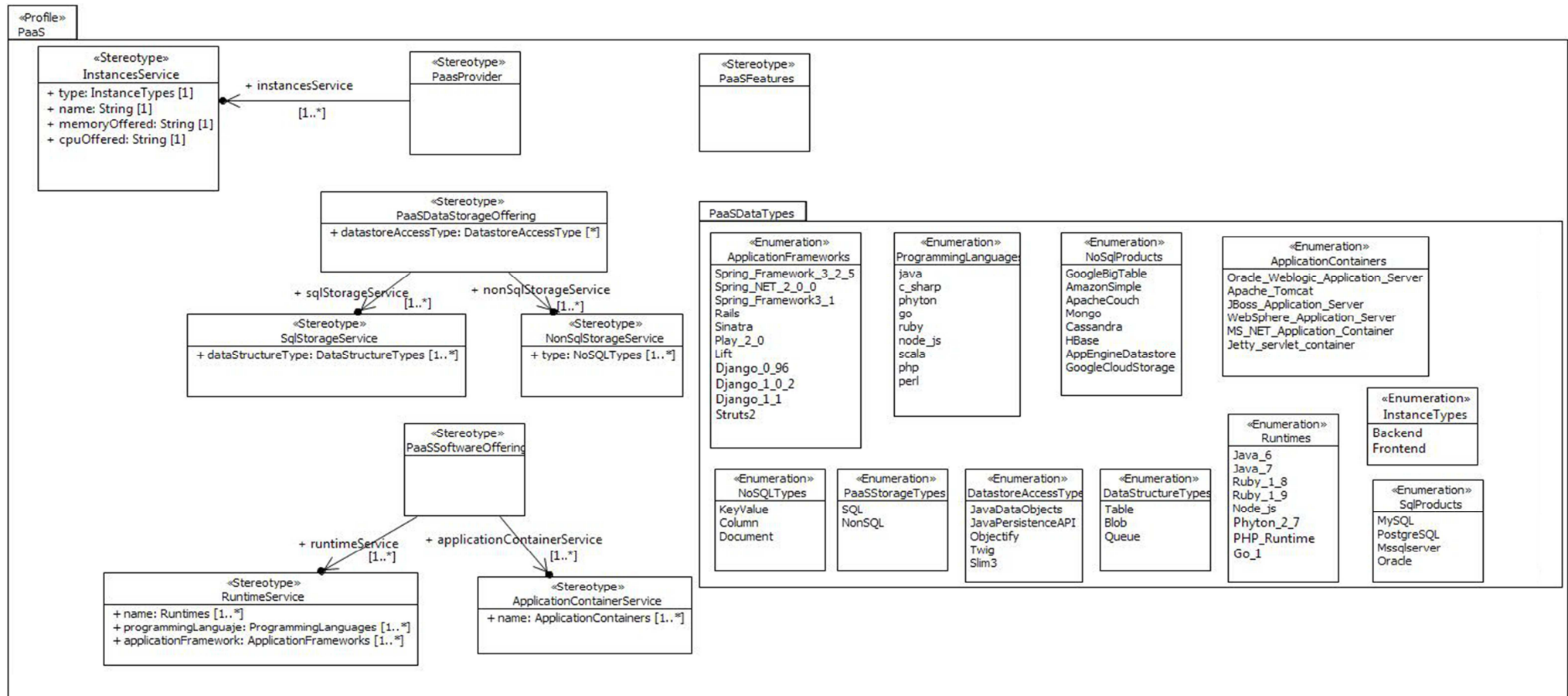
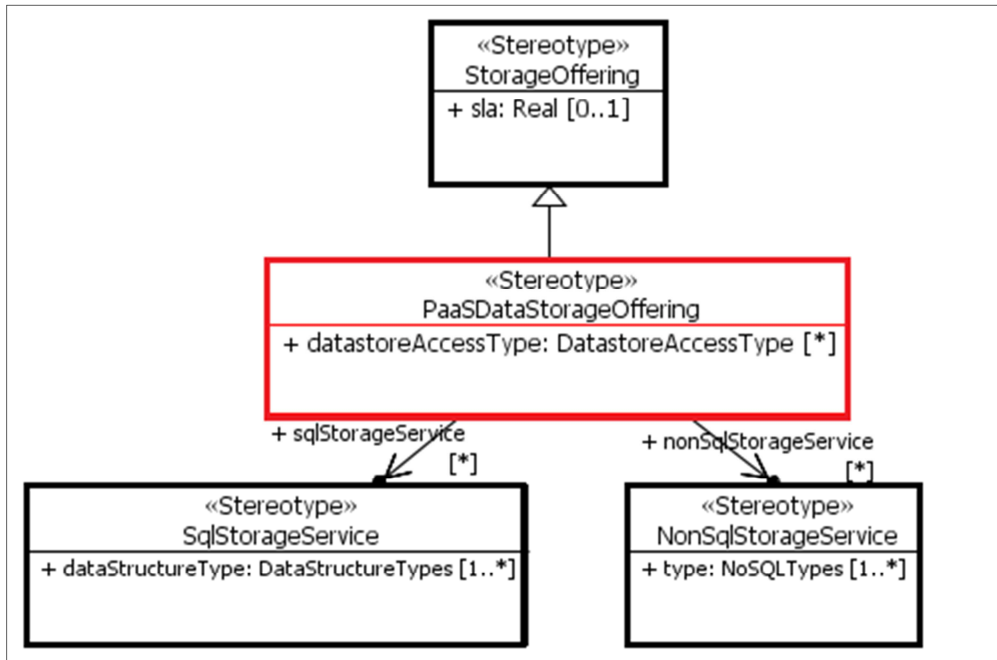


Figure 45: PaaS Stereotypes

Next, a detailed description of the most relevant stereotypes included in the PaaS profile. **PaaSDataStorageOffering**



This stereotype allows modelling the data storage offering related to a PaaS provider.

It extends from **StorageOffering** stereotype that is one of the common stereotypes that can be used in the context of PaaS and IaaS providers' models (it is included at a higher level).

The data storage offering consists of a set of services that can be offered by the provider.

These services are classified as sql storage services and non sql storage services, depending on the nature of the storage system provided.

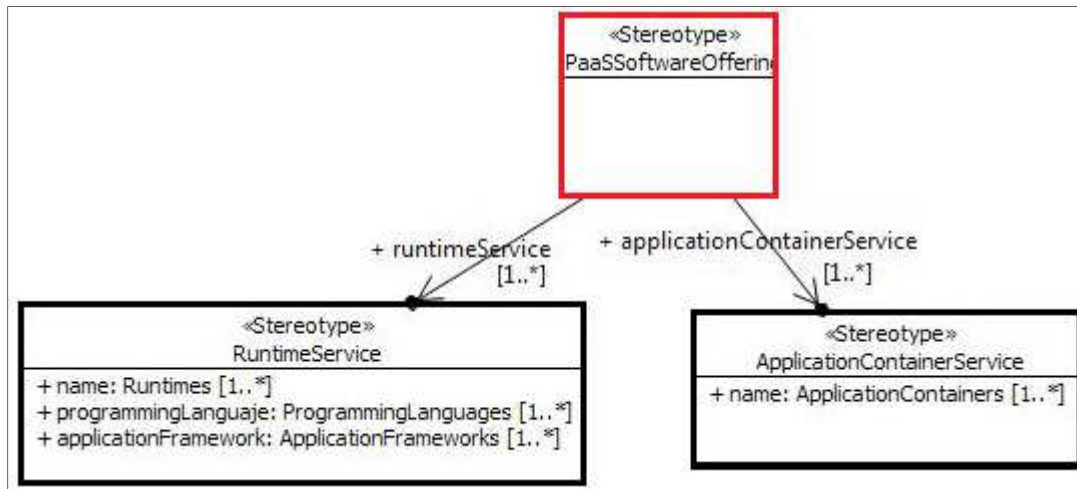
Next are described the properties contained in the stereotype.

Name	Type	Card.	Description
datastoreAccessType	DatastoreAccessType	1..*	Type of access to the data storage. Possible values given through DatastoreAccessType enumeration entity whose values can be seen in the previous picture.

## PaaSSoftwareOffering

Project Title: ARTIST

Contract No. FP7317859  
www.artist-project.eu



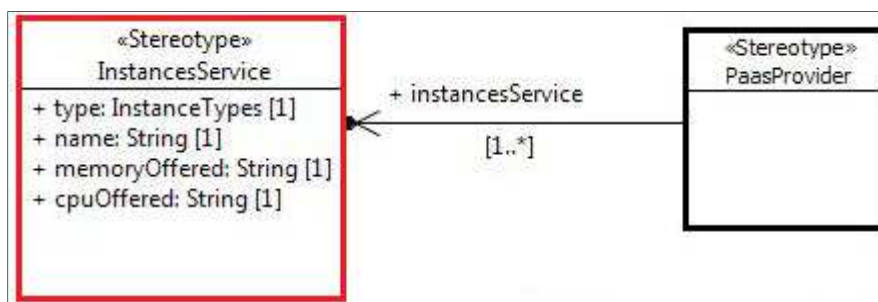
This stereotype allows modelling the software offering related to a PaaS provider.

It extends from PlatformOffering included in the core package.

The PaaSSoftwareOffering consists of a set of services that can be offered by a PaaS provider with the aim of making easier the deployment of applications on the servers. Through his offering, the providers offer preinstalled software environments such as frameworks and application servers, so the developer doesn't have to install them in the production environment, he/she can select those services fitting the needs of the application and use them directly.

As can be seen in the picture, for now we have included two types of software services that can be offered under this categorization: Runtime services, through which are offered different runtimes, programming languages and application frameworks and Application container services offering the possibility of using different preinstalled application servers. When modelling those services, specific values for the properties (i.e programming languages) can be selected from the enumeration entities offered in the scope of the PaaS profile (on the right side of the picture).

### InstancesService





Apart from providing platform related services, PaaS providers also offer infrastructure resources, so the applications can be deployed on them.

With the aim of modelling this feature, the InstancesServices stereotype allows to model the different instance types a PaaS provider can offer.

By now, and after analysing the offerings of most relevant PaaS providers, the instances can be classified as being of “Backend” or “Frontend” types.

Each instance offers a specific amount of memory and cpu available, so the developer/deployer can choose which one is the more appropriate according to the application's needs.

It extends from Service stereotype included in the core package.

Next are described the properties contained in the stereotype.

Name	Type	Card.	Description
type	InstanceTypes	1	Type of instances a PaaS provider can offer. Possible values given through InstanceTypes enumeration entity whose values can be seen in the previous picture.
name	String	1	Name of the instance used for identification purposes.
memoryOffered	String	1	Memory offered by the instance in MB.
cpuOffered	String	1	CPU offered by the instance in GHZ.

## SaaS

By this extension, implemented through a UML profile as the other extensions of CloudML, it is intended to provide the possibility to model software as a service (SaaS) features, thus covering along with IaaS and PaaS profiles, the needs of the three flavours of cloud computing.

Taking into account its flexibility to be extended if new needs arise, the current version of the profile allows to model different software services by specifying authentication methods to access the services, invocation urls, interfaces through which the service can be invoked and also configuration aspects.

Next a detailed description of the most relevant stereotypes included in the SaaS profile:

## SoftwareService

In order to integrate 3rd party services or client libraries, both could be seen as an API, we basically need 3 things: an Interface, Configuration, and Security.

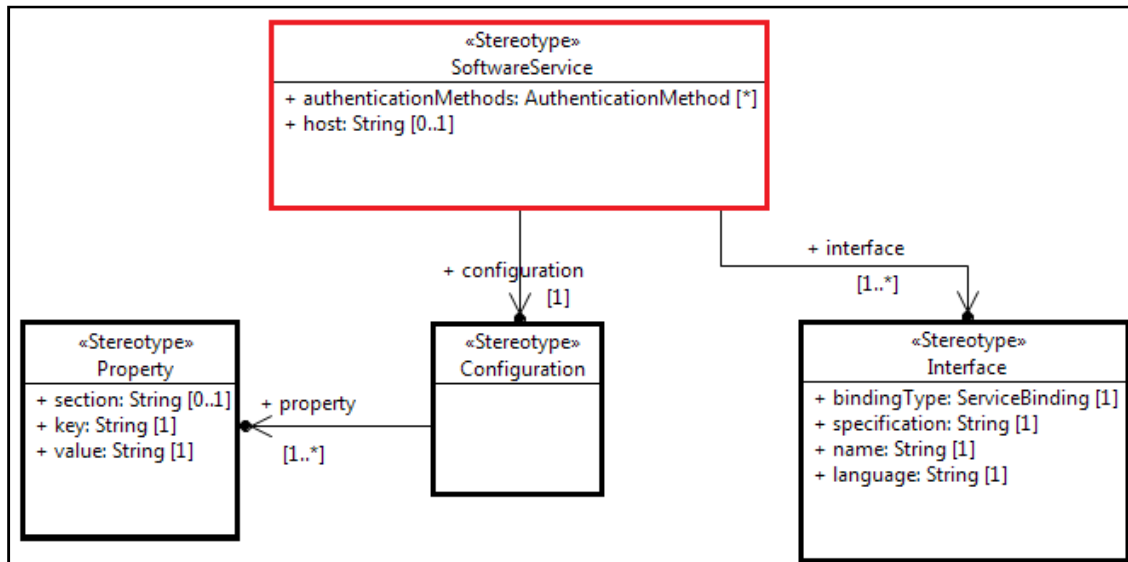


Figure 46: SoftwareService Stereotype

This stereotype allows modelling the software services a SaaS provider can offer.

It extends from Service stereotype defined at a higher level, whose description can be found on the beginning of this appendix. It contains a name and optionally a host URI pointing towards the actual service (in client libraries, this would not be necessary).

In the following table the properties contained in the stereotype are described.

Name	Type	Card.	Description
authenticationMethods	AuthenticationMethod	*	Type of authentication needed to access the service.
host	String	0..1	Host where is allocated the service.
category	SoftwareServiceCategory	0..1	A category describing the functionality provided by the service

Every SoftwareService supports a number of authentication methods that can be specified through a list of elements of type AuthenticationMethod. Currently, OAuth, SAML, HMAC, and Basic are provided as the ones most used nowadays. However, many more can be easily added in the future.

An additional category field is added which is related to the functionality provided by the SaaS/service. Here the same categorization of services is used as described in D7.3 defined by an enumeration type SoftwareServiceCategory. These categories are identity/multi-tenancy, security, monitoring, logging, billing, caching and messaging,

Every service exposes itself through a number of Interfaces. These interfaces are of a certain type, named BindingType. For web services, most of them are either HTTP/REST or SOAP services. In order to support client libraries as a kind of API, the binding type can be specified as being a Language. In this case the language needs to be specified as well through the corresponding property.

The actual API (client library or web service) description will be captured through the specification property which points to a URI. There are numerous interface description languages already available:

- RSDL<sup>6</sup>
- WSDL<sup>7</sup>

These specifications are eventually can be used to generate stub code / glue code to either use and/or access the API.

The following table gives an overview of the different properties belonging to the Interface stereotype:

Name	Type	Card.	Description
bindingType	ServiceBinding	1	The type of binding, either, how the communication happens with the service.
specification	String	1	A pointer towards a file containing the actual service description.
name	String	1	A short name for identifying the service.

<sup>6</sup> <http://en.wikipedia.org/wiki/RSDL>

<sup>7</sup> [http://en.wikipedia.org/wiki/Web\\_Services\\_Description\\_Language](http://en.wikipedia.org/wiki/Web_Services_Description_Language)

language	String	1	The language in which the service is being used (important for the use of software libraries).
----------	--------	---	--

A Configuration description needs to be specified for all services and libraries used. Every service however requires different configuration properties so they preferably need to be generic. The typical properties are:

- Application/User Id
- Secret Key
- ...

The table below gives a short overview of the different properties belonging to the Property stereotype:

Name	Type	Card.	Description
section	String	0..1	A category of the properties in order to further group the different properties.
key	String	1	The key identifying the property
value	String	1	The value belonging to the particular key.

## Appendix B – Availability profile

In order to define measured availability in CloudML@ARTIST the insertion of both fields GenericAvailability and ProviderSpecificAvailability is needed. The first one is related to the generic definition followed by Cloudsleuth, based on which we can compare different services on actual availability statistics. Generally this definition is based on the number of samples acquired, which can be summarized as follows:

$$Availability = \frac{TotalSamples - UnavailableSamples}{TotalSamples} \quad \text{equation 1}$$

For the population of this metric we will utilize the information provided by Cloudsleuth. The second one is related to the provider and service specific definition and measurement of availability, as stated in their respective SLAs. Moreover for the description of each provider's specific availability the addition of a new field in the availability template is required. This new field is ProviderSpecificAvailabilityStatsType and can be attached to each service offering that is bounded by an availability-related SLA. The overall Availability measurement stereotypes appear in Figure 47.

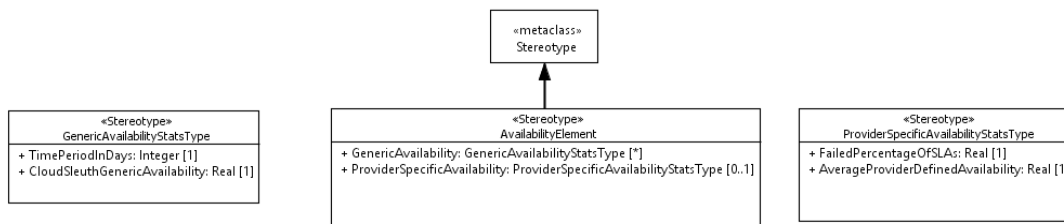


Figure 47: Availability measurement stereotypes

AvailabilityElement parameters are shown in the following table:

Name	Type	Card.	Description
GenericAvailability	GenericAvailabilityStatsType	*	Generic availability of service offering
ProviderSpecificAvailability	ProviderSpecificAvailabilityStatsType	0..1	Provider's specific availability based on each provider's definition

GenericAvailabilityStatsType subtype expresses availability statistics that are measured by CloudSleuth.

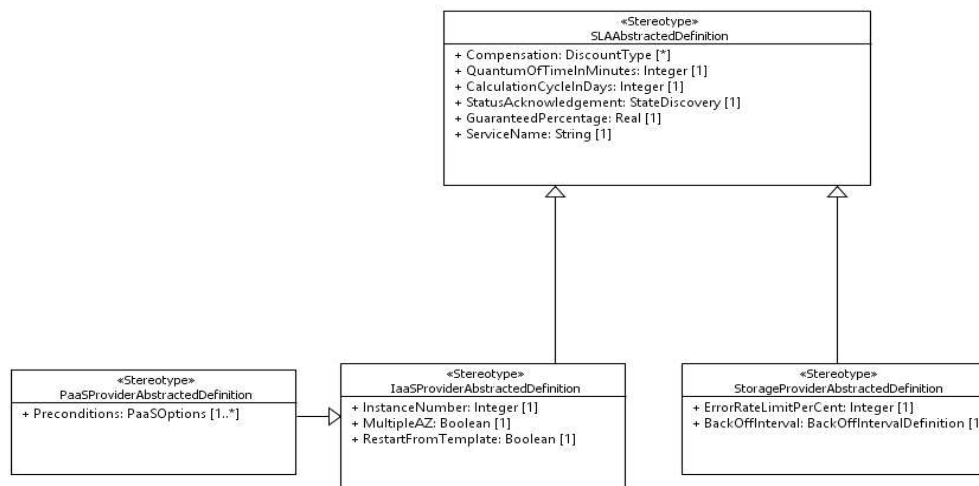
Name	Type	Card.	Description
TimePeriodInDays	Integer	1	Time period of statistics' collection

			chosen through official site of Cloudsleuth
CloudSleuthGenericA availability	Real	1	Generic availability based on Cloudsleuth equation 1

In the following table the properties contained in the ProviderSpecificAvailabilityStatsType stereotype are described. The concrete value for this field can be acquired through the use of the Abstracted Availability Auditor Library (3ALib) that we have developed. Further information regarding this library can be found in the content of D7.1.

Name	Type	Card.	Description
FailedPercentageOfSL As	Real	1	The percentage of failures due to an unavailable VM
AverageProviderDefin edAvailability	Real	1	The average of availability for each provider measured by 3ALib

Both Figure 48 and Figure 49 illustrate the total description of the overall availability SLA definition profile. The UML profile describes SLA definitions that come from compute, storage and platform level application frameworks layers and can be used to describe the specific SLA instances that are offered by the examined providers (Google Amazon and Azure). The rationale behind the abstraction of the SLA definitions is the identification of common concepts in the way the SLAs are defined for the respective providers.



**Figure 48: General information of SLA definition regarding PaaS, IaaS and Storage providers**

The SLAAbstractedDefinition which is described below contains the general information of SLA definition which is specialized in PaaSProviderAbstractedDefinition, IaaSProviderAbstractedDefinition and StorageProviderAbstractedDefinition subelements.

Name	Type	Card.	Description
Compensation	DiscountType	*	Compensation that customers claim after proving the violation
QuantumOfTimeInMinutes	Integer	1	Quantum of downtime period
CalculationCycleInDays	Integer	1	Calculation cycle of a month
StatusAcknowledgement	StateDiscovery	1	Representation of status availability
GuaranteedPercentage	Real	1	Availability guaranteed percentage provided by SLAs
ServiceName	String	1	Unique name of service

Next is described the only property contained in the PaaSProviderAbstractedDefinition stereotype which indicates in an abstracted way the necessary preconditions must typically apply for an SLA to be applicable regarding PaaS providers.

Name	Type	Card.	Description
Preconditions	PaaSOptions	1..*	necessary preconditions must typically apply for an SLA to be applicable

IaaSProviderAbstractedDefinition field includes parameters that result from comparing different IaaS provider's SLA definitions and identifying common concepts and differences. Information regarding the fields appears in the following table.

Name	Type	Card.	Description
InstanceNumber	Integer	1	The number of VM instances
MultipleAZ	Boolean	1	The need for different Availability Zones
RestartFromTemplate	Boolean	1	The need of restart template before validating an unavailability sample

Regarding the storage layer the properties that have been identifying after examining the three SLAs and have been included in StorageProviderAbstractedDefinition are ErrorRateLimitPerCent and BackOffInterval. The first one is related to the metric Error Rate which is defined as follows:

$$\text{Error Rate} = \frac{\text{FailedSamplesInInterval}}{\text{TotalSamplesInInterval}}$$

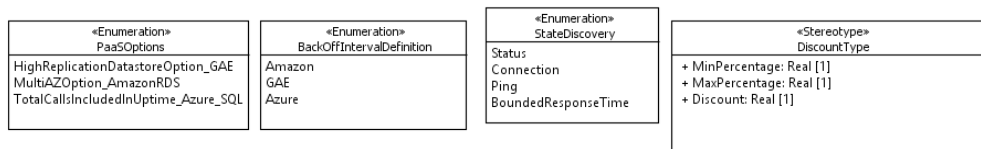
While the definition for the three providers is the same, some details may differ such as the duration of the interval (Amazon 5 minutes, Google 10 minutes and Azure 1 hour).and the fact that Azure defines a failed sample not only in terms of success or failure, but also in terms of the delay of the response. The second one, BackOffInterval, is necessary only for Azure and Google and is defined as the requirement not to enquire the service status constantly in case of failure, but with a predefined level of delay between the requests (different in definition for each provider).

Name	Type	Card.	Description
ErrorRateLimitPerCent	Integer	1	Percentage of Error Rate metric



BackOffInterval	Real	1	Identifying failure with a predefined level of delay between the requests
-----------------	------	---	---

In order to simplify the modelling framework of the Availability SLA definition, we have defined different enumeration fields that include information for supporting the three aforementioned fields namely PaaSProviderAbstractedDefinition, IaaSProviderAbstractedDefinition and StorageProviderAbstractedDefinition in which SLAAbstractedDefinition is specialized. Apart from enumeration fields (Figure 49) also DiscountType stereotype is illustrated. In this field compensation can be linked in order to describe what is promised as compensation by cloud providers in the case of failures.



**Figure 49: Illustration of discount type and different enumeration fields that complete the SLA definition model**

The properties contained in the PaaSOptions enumeration are described in the following table. Each of these options is unique and is related to a specific cloud provider.

Name	Description
HighReplicationDatastoreOption_GAE	high replication datastore option enabled used by GAE
MultiAZOption_AmazonRDS	For Amazon multi-AZ deployment is automatic following the enablement of the according option by the user
TotalCallsIncludedInUptime_Azure_SQL	Factor of total calls used in formula for defining uptime

BackOffIntervalDefinition enumeration properties are shown in table. This field includes the different logic and values defined by cloud providers regarding back off interval definition. For Amazon this metric is not required (thus we can set it at 0) while for Azure and GAE follows a dynamic calculation. At the moment BackOffIntervalDefinition includes enumeration literals

(named values), however in the future we are going to extend this field in order to be linked with subtypes describing extensively the specific calculation of back off interval for each cloud provider.

Name	Description
Amazon	Not necessary(set it at 0)
GAE	starting at 1 second and increasing exponentially up to 32 seconds
Azure	Azure after the first timeout/server busy wait 3 seconds, then 30 seconds, and then 90 seconds before retrying after each subsequent timeout/server busy

StateDiscovery enumeration parameters are shown in the following table. Each cloud provider, provides in a different way, information regarding the state of availability.

Name	Description
Status	Status message
Connection	Simple connection ability to the VM)
Ping	Ping utility to verify availability
BoundedResponseTime	Determination of a response time limit which characterizes if the service is available or not

DiscountType parameters are shown in the following table. This includes the availability levels between which a discount applies, along with the percentage of discount.

Name	Type	Card.	Description
MinPercentage	Real	1	Minimum percentage of actual/observed availability value for which the discount applies
MaxPercentage	Real	1	Maximum percentage of actual/observed availability value for which the discount applies
Discount	Real	1	The amount of discount offered by cloud providers in case of not meeting SLA agreements regarding availability

## Appendix C – Performance Profile

As mentioned in D7.1 in order to identify and check the performance rating across different application areas, we have defined application types that correspond to various common applications and can be met in real life environment. These application types were linked with specific benchmarks that provide application level workloads characterization and raise the abstraction for the end user. More information regarding the analysis of these applications and the description of benchmarks can be found in the content of D7.2 and D7.1 respectively.

In order to incorporate the information regarding performance aspects in CloudML@ARTIST profile, we have created a sub-profile that can be attached to the generic description of core profile that is used to describe a virtual offering. This subprofile includes a number of different benchmarks covering the most prominent application types and providing the ability for acquiring performance score results.

Regarding performance profile analysis, the basic stereotype is BenchmarkedElement which includes BenchmarkResult (Figure 50) stereotype, as an attribute. The latter includes only one property and is related to the potential benchmark workloads. Moreover in the definition of our performance model an OCL constraint has been generated. According to this constraint, a BenchmarkElement can be only applied to an InstanceType element. In the same way, different constraints may be defined throughout the profiles, to link them with the core profiles.

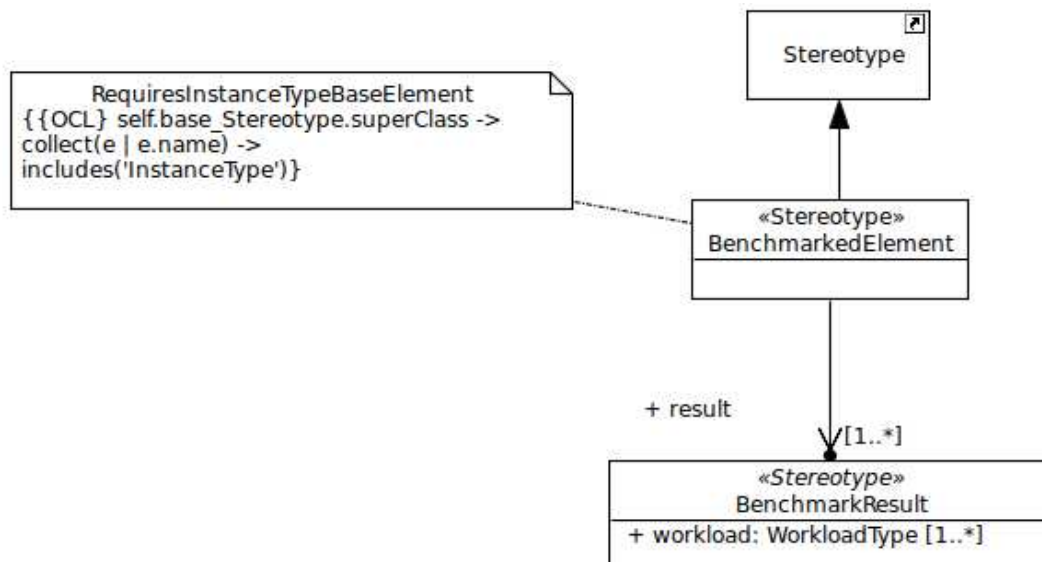


Figure 50: Illustration of the basic part of performance model and the OCL constraint

Next is described the property contained in BenchmarkResult stereotype.

Name	Type	Card.	Description
workload	WorkloadType	1..*	specific workload patterns that can be mapped to concrete applications

DacapoResult properties are described in the following table. The DaCapo benchmarks reflect performance time and are used in order to evaluate Java-based applications. Cardinality in all results can be also 0, in case no tests have been performed for this specific benchmark.

Name	Type	Card.	Description
PerformanceTime	Real	0..1	Response time for test completion

With regard to YCSBResult stereotype, YCSB reports back a number of metrics such as runtime, throughput, number of operations, average, minimum and maximum latency. These are included in the performance profile in order to describe overall results of an offering.

Name	Type	Card.	Description
runtime	Real	0..1	Response time for test completion
throughput	Real	0..1	Execution time needed for workload completion
operations	Real	0..1	Operations/sec
averageLatency	Real	0..1	Update operations completed
minLatency	Real	0..1	Average time per operations(the Client measures the end to end latency of executing a particular operation against the database)
MaxLatency	Real	0..1	Minimum latency
			Maximum latency

DwarfsResult stereotype includes the problem size which is set as a real parameter and the execution time for test completion. In the following table the aforementioned parameters are summarized.

Name	Type	Card.	Description
score	Real	0..1	Runtime benchmark result
size	Real	0..1	Problem size

FilebenchResult stereotype has been defined in order to capture the typology of results, including the various statistics that are returned. In the following table the aforementioned parameters are summarized.

Name	Type	Card.	Description
ops	Real	0..1	The number of operations
throughputOpsSec	Real	0..1	Operations per second
rw	Real	0..1	Reads/writes to get a feeling for maximum performance
bandwidthMbSec	Real	0..1	Megabytes/second
cpuOp	Real	1	Number of cpu operations
Latency	Real	0..1	Latency

Regarding the Cloudsuite case, which offers a benchmark suite for emerging scale-out applications(eight application types) we have kept only the generic average score to be included in the model instances in order to simplify the descriptions (each of the applications reports a large number of statistics, that are case specific). Next the CloudSuiteResult stereotype is described.

Name	Type	Card.	Description
Average_score	Real	0..1	Average score related to specific

			benchmark application type
--	--	--	----------------------------

All of the above benchmark results, included in the performance model, are represented in Figure 51.

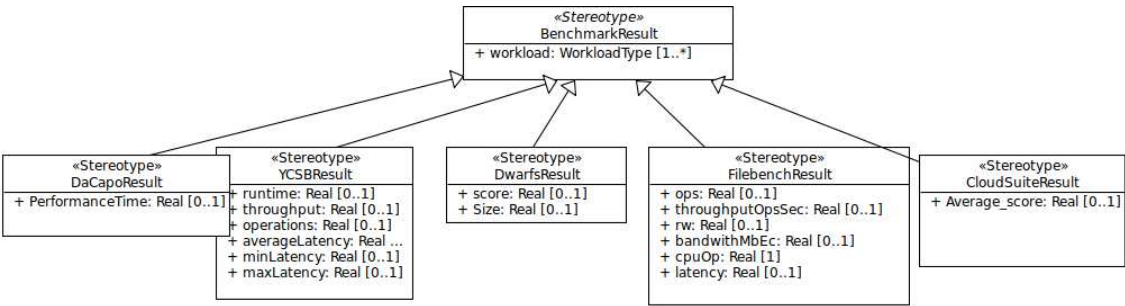


Figure 51: Benchmark results included in performance profile

In order to simplify the benchmark profile we have defined one universal enumeration (Figure 52) that includes the default workloads from the aforementioned benchmark categories. These workloads are static in order to be able to compare the performance of different services on the same examined workload.

«Enumeration» WorkloadType
YCSB_Update_Heavy
YCSB_Read_Heavy
YCSB_Read_Only
YCSB_Read_Latest
YCSB_Short_Ranges
Filebench_Webserver
Filebench_Fileserver
Filebench_Varmail
Filebench_Videoserver
Filebench_Webproxy
Filebench_OLTP
DaCapo_Avrora
DaCapo_Batik
DaCapo_Jython
DaCapo_LuIndex
DaCapo_Xalan
CloudSuite_Datacaching_Twitter
Dwarf_StructuredGrid_3DCurl
Dwarf_GraphTraversal_Quicksort
CloudSuite_MediaStreaming_GetShortHi

Figure 52: Illustration of the universal enumeration with the default workloads

In the following table a detailed description of the workloads included in the WorkloadType enumeration is given.

Name	Description
YCSB_Update_Heavy	a mix of 50/50 reads and writes
YCSB_Read_Heavy	a 95/5 reads/write mix
YCSB_Read_Only	100% read
YCSB_Read_Latest	new records are inserted
YCSB_Short_Ranges	short ranges of records are queried, instead of individual records
Filebench_Webserver	Emulates simple web-server I/O activity
Filebench_Fileserver	Emulates simple file-server I/O activity
Filebench_Varmail	Emulates I/O activity of a simple mail server that stores each e-mail in a separate file (/var/mail/ server)
Filebench_Videoserv er	emulates a video server
Filebench_Webproxy	Emulates I/O activity of a simple web proxy server
Filebench_OLTP	A database emulator
DaCapo_Avrora	simulates a number of programs running on a grid of AVR micro-controllers
DaCapo_Batik	produces a number of Scalable Vector Graphics (SVG) images based on the unit tests in Apache Batik
DaCapo_Jython	interprets pybench Python benchmark
DaCapo_Luindex	Uses lucene to indexes a set of documents; the works of Shakespeare and the King James Bible
DaCapo_Xalan	transforms XML documents into HTML ones
CloudSuite_Datacachi ng_Twitter	a simulation of Twitter-type workload for in cache memory data
CloudSuite_MediaStr eaming_GetShortHig h	It consists of two main components a client and a server: the client component emulates real world clients; sending requests to stress a streaming server.
Dwarf_StructuredGri d_3DCurl	Regular grids, can be automatically refined
Dwarf_GraphTraversa l_Quicksort	Decision Tree, searching, quicksort



## Appendix D – Cost profile

This section describes the “Cost profile” included in CloudMI@Artist, that it is used mainly to specify information related to prices and discounts that can be applied to the different services offered by the cloud providers.

A variety of pricing schemes can exist in modern Cloud infrastructures, based on different offering characteristics, means of payment or acquisition methods. ServiceCost is a generic type which includes two significant subtypes Acquisition and Discount.

Next, a picture showing the stereotypes and data types included in this profile and a detailed description of each one.

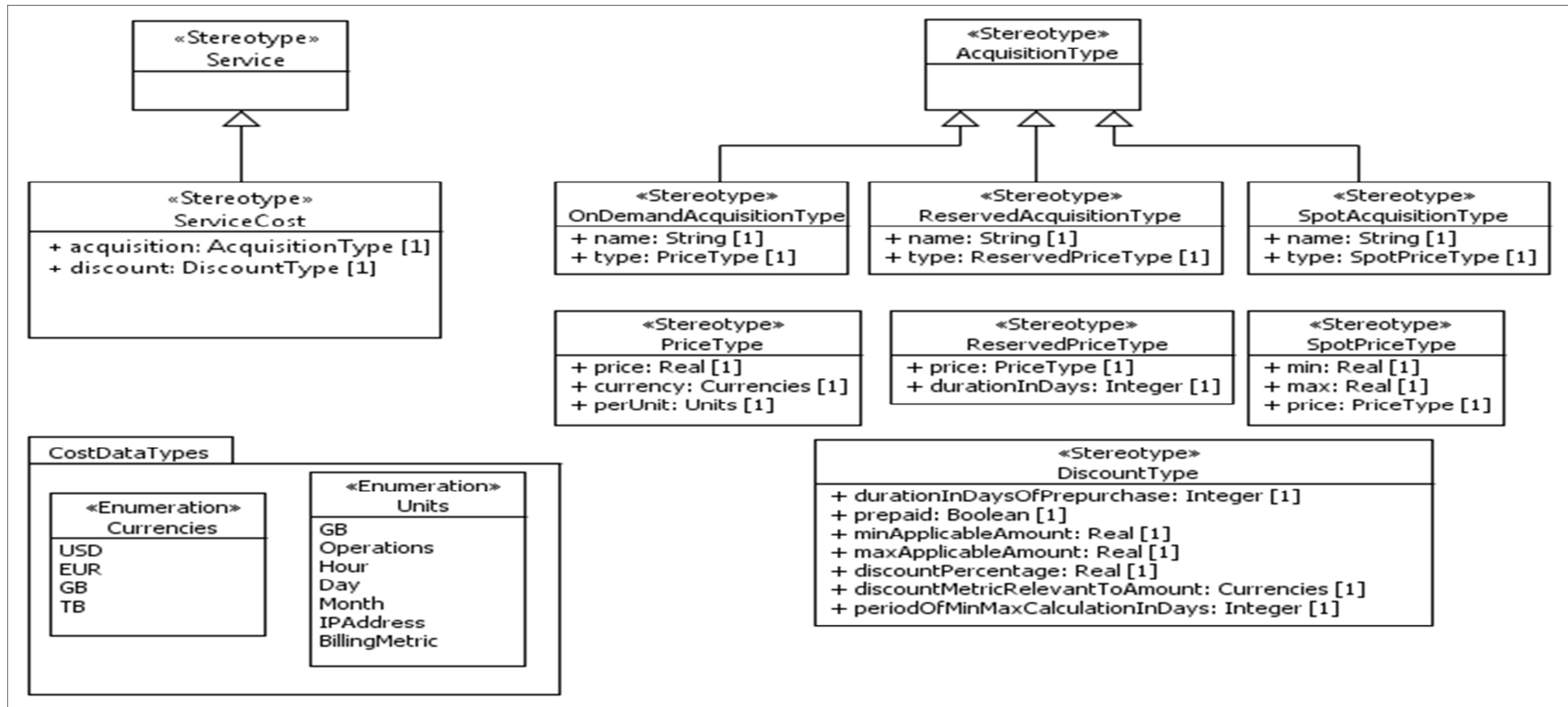
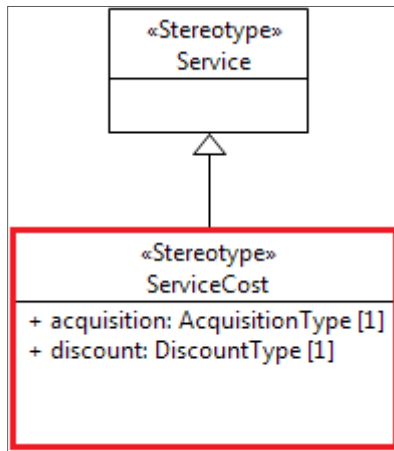


Figure 53: Cost Profile Stereotypes

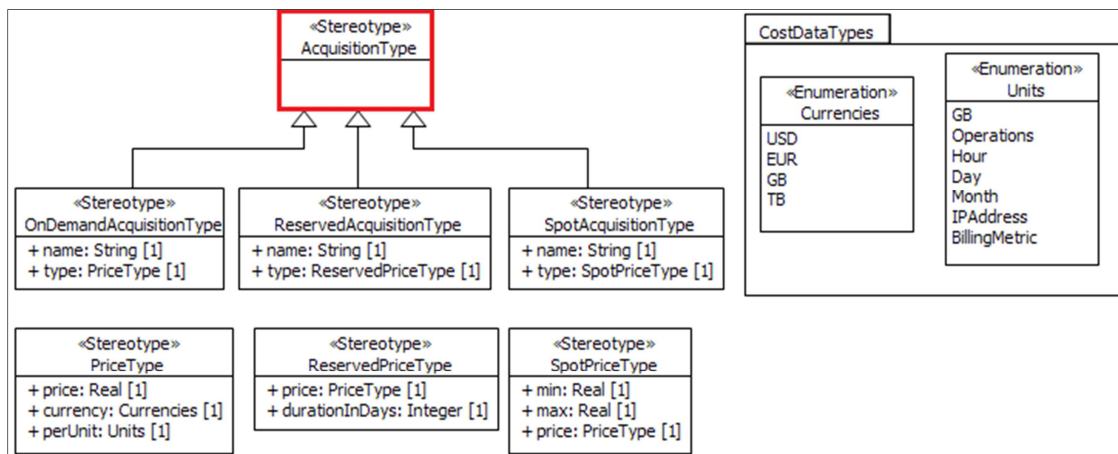
## ServiceCost



Parameters for this type include the acquisition type and potential discounts that may be applicable per case.

Name	Type	Card.	Description
acquisition	AcquisitionType	1	Variety of options (on demand, spot or reserved pricing)
discount	DiscountType	1	Potential discounts

## AcquisitionType



Furthermore, for a service acquisition, there may be a variety of options (for example for Amazon services there may be on demand, spot or reserved pricing). For this reason we have created the AcquisitionType (as shown in the figure above), to indicate the various options along with their specifics. The final price element is of the basic PriceType, which aims to capture cost values, currencies and cost item. This PriceType can be included to indicate the price of a variety of offerings such as storage and CPU, or as a subtype in more complex types such as the networking costs.

Name	Type	Card.	Description
onDemand	PriceType	1	Pay per use, pay as you go
reserved	ReservedPriceType	1	e.g Amazon Reserved Instances
spot	SpotPriceType	1	Run instances as long as long as their bid exceeds the spot price

### PriceType

PriceType parameters are shown in the following table:

Name	Type	Card.	Description
price	Float	1	Price of the offering
currency	String	1	Currency on which it is established the price.
perUnit	Units	1	Price unit, whose possible values can be selected from "Units" enumeration data type.

### ReservedPriceType

This type is related to the cost of reserved instances which gives customer the option to make a low, one-time payment for each instance wants to reserve and in turn receive a significant discount on the hourly charge for that instance.

Name	Type	Card.	Description
------	------	-------	-------------

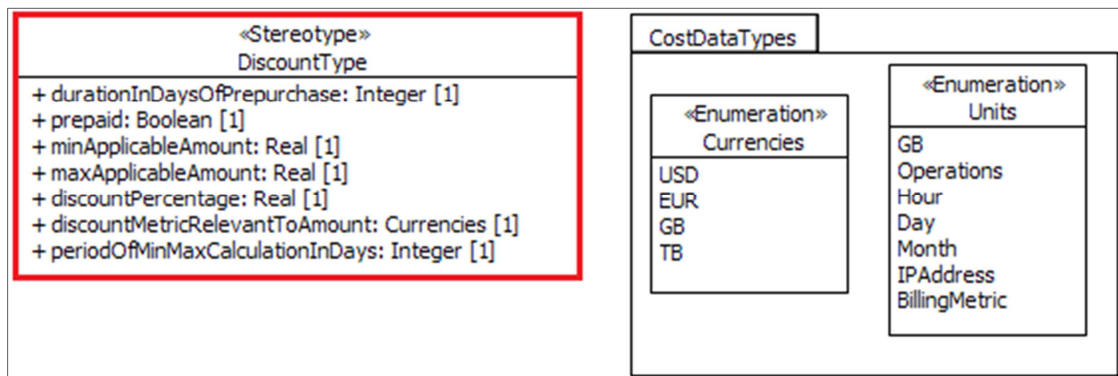
price	PriceType	1	Cost of the reserved price
durationInDays	Integer	1	Duration of reserved price

### SpotPriceType

This type is related to the cost of Spot Price. For instance Amazon EC2 charges the Spot Price, which fluctuates periodically depending on the supply of and demand for Spot Instance capacity.

Name	Type	Card.	Description
min	Float	1	Min value of spot price
max	Float	1	Maximum value of spot price
price	PriceType	1	Cost of spot price

### DiscountType



This type is defined in order to specify various discounts that different providers offer to customers, in relation for example to usage size.

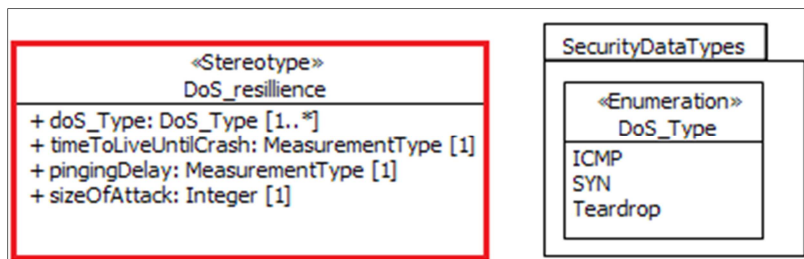
Name	Type	Card.	Description
durationInDaysOfPrepurchase	Integer	1	Duration of discount
prepaid	Boolean	1	Get discount when prepay
minApplicableAmount	Float	1	Minimum discount limit

maxApplicableAmount	Float	1	Maximum discount limit
discountPercentage	Float	1	Percentage of discount
discountMetricRelevant ToAmount	Currencies	1	Currency on which it is applied the discount.
periodOfMinMaxCalcul ationInDays	Integer	1	Period of time while is valid the discount.

## Appendix E – Security profile

Another crucial aspect of Cloud environments is their ability to scale for meeting increased demand. However this scaling does not come cheap and in the event of a Denial of Service attack the resulting costs could be significant for a service offering. Thus the ability of a Cloud provider to successfully meet this type of attack on e.g. a network level with the usage of suitable filtering mechanisms etc. is critical. But even if the issue of raising costs is not applicable, the performance of the offering itself (and indirectly the performance of the provider countermeasures) should be tested when it consists a target for a DoS attack.

Next a picture containing stereotypes and data types related to this profile.



Name	Type	Card.	Description
dos_Type	Dos_TypeType	1	DoS attacks(ICMP, SYN, Teardrop)
timeToLiveUntilCrash	Double	1	Time takes until crash
pingingDelay	Double	1	Delay because of pinging (e.g ping flood)
sizeofAttack	Float	1	Attacks size (Packet Per Second, average Bits Per Second)

## **Appendix F – Provider model example**

In the next two pictures a Windows Azure model is portrayed, created by using CloudML@Artist.

For doing this, an uml model has been created that makes use of the previously described profiles. Even it is an indivisible element, the model has been split in two images for a better visualization. The first one contains the main stereotypes describing the provider and the second one contains the stereotypes representing the instance types Windows Azure can offer.



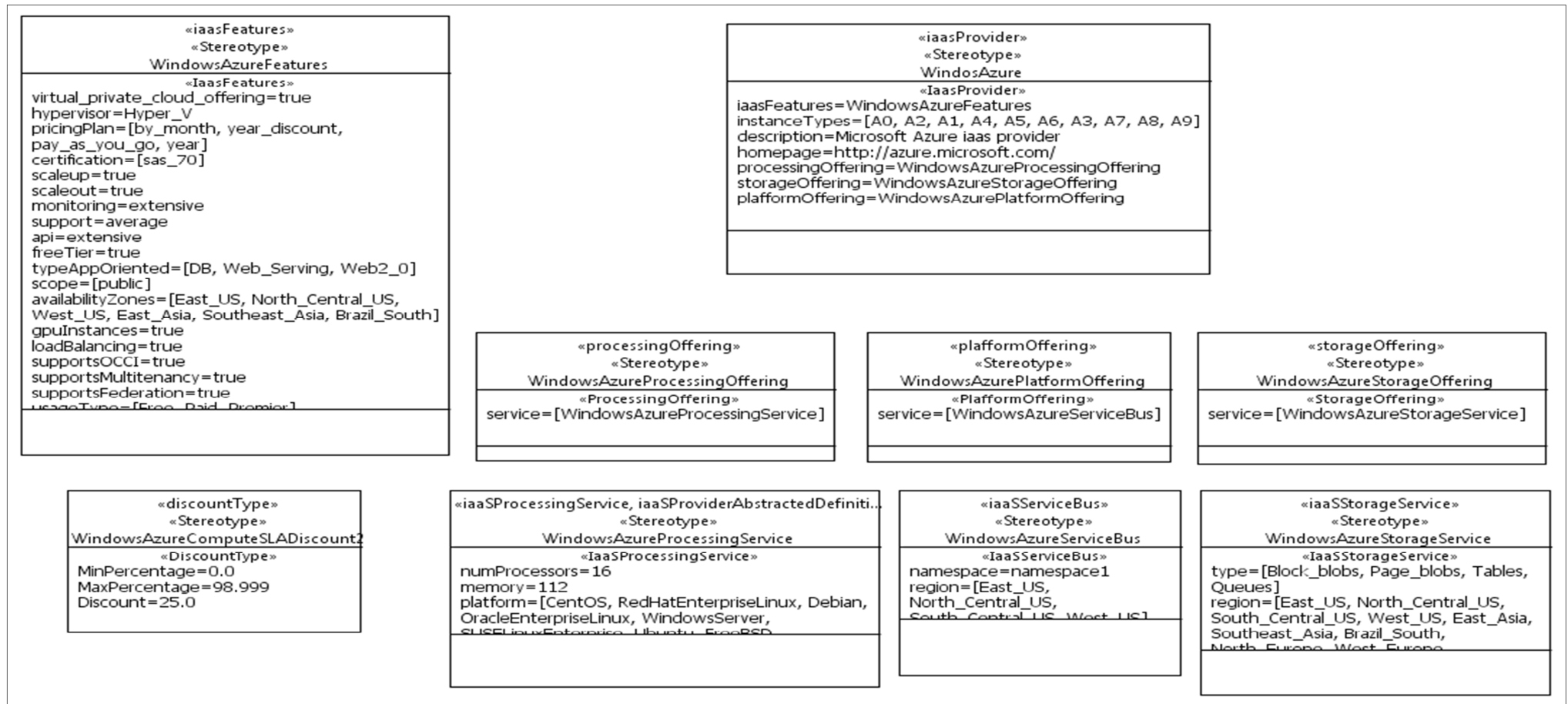


Figure 54: Windows Azure model main stereotypes

As stated out before, in the previous image can be seen the main stereotypes contained in Windows Azure model. Next a brief description of each one.

- WindowsAzure: this entity implements IaaSProvider stereotype. It contains properties describing the provider in general terms, such as name, description and home page, and other properties referencing entities in the model specifying Windows Azure's features (WindowsAzureFeatures) and the different offerings given by the provider (WindowsAzureProcessingOffering, WindowsAzurePlatformOffering, WindowsAzureStorageOffering).
- WindowsAzureFeatures: IaaSFeatures (whose description can be found on Appendix A) is applied to this entity. It contains specific values related to Windows Azure for properties containing information such as generic pricing plans offered, availability zones, if the provider offers scaling up and scaling out services and so on.
- WindowsAzureProcessingOffering: ProcessingOffering stereotype defined at CloudML@Artist's core level is applied to this entity. It contains the enumeration of services related with the processing capacity offered by the provider and the calculated sla for this kind of offering.
- WindowsAzurePlatformOffering: PlatformOfferingstereotype defined at CloudML@Artist's core level is applied to this entity. It contains the enumeration of platform related service and the calculated sla for this kind of offering.
- WindowsAzureStorageOffering: StorageOffering stereotype defined at CloudML@Artist's core level is applied to this entity. It contains the enumeration of services related with the storage capacity offered by the provider and the calculated sla for this kind of offering.
- WindowsAzureProcessingService: IaaSProccesingService and IaaSProviderAbstractedDefinition stereotypes are applied to this entity. It contains information related to the processing capacity offered, such as maximum number of processors, ram and processor architectures available. It also contains availability information expressed by means of the application of the IaaSProviderAbstractedDefinition sterotype, contained in the Availability uml profile (please see a complete description of the entity at Appendix B).
- WindowsAzureServiceBus: Entity created explicitly for Windows Azure provider. Even though the service bus is offered also by other cloud providers, for now it is not so extended to make it necessary to define it at a higher level in CloudML@Artist. This is why it has been included as a standalone stereotype in the model. The stereotype contains information related to the namespaces where the service can be applied and also the regions where it is applicable.
- WindowsAzureStorageService: IaaSStorageService is applied to this entity. It contains information related to the storage capacity offered, such as type of storage offered by the provider, regions (there may exist different data privacy constraints depending on the country where it is stored), maximum capacity in gigabytes and replication types.
- WindowsAzureComputeSLADiscount1: DiscountType stereotype from Availability

profile (see a detailed description on Appendix B) is applied to this entity. Main purpose of this stereotype is to represent the amount of discount offered by cloud providers in case of not meeting SLA agreements regarding availability.

«IaaSInstanceType» «Stereotype» A0 «IaaSInstanceType» instanceCategory=[General_Purpose] region=[] operatingSystem=[CentOS, RedHatEnterpriseLinux, Debian, OracleEnterpriseLinux, WindowsServer, SUSELinuxEnterprise, Ubuntu, FreeBSD, openSUSELinux, SUSEEnterpriseLinux] virtualCores=0 memory=0.768	«IaaSInstanceType» «Stereotype» A1 «IaaSInstanceType» instanceCategory=[General_Purpose] region=[] operatingSystem=[CentOS, RedHatEnterpriseLinux, Debian, OracleEnterpriseLinux, WindowsServer, SUSELinuxEnterprise, Ubuntu, FreeBSD, openSUSELinux, SUSEEnterpriseLinux] virtualCores=1 memory=1.75	«IaaSInstanceType» «Stereotype» A2 «IaaSInstanceType» instanceCategory=[General_Purpose] region=[] operatingSystem=[CentOS, RedHatEnterpriseLinux, Debian, OracleEnterpriseLinux, WindowsServer, SUSELinuxEnterprise, Ubuntu, FreeBSD, openSUSELinux, SUSEEnterpriseLinux] virtualCores=2 memory=2.5	«IaaSInstanceType» «Stereotype» A3 «IaaSInstanceType» instanceCategory=[General_Purpose] region=[] operatingSystem=[CentOS, RedHatEnterpriseLinux, Debian, OracleEnterpriseLinux, WindowsServer, SUSELinuxEnterprise, Ubuntu, FreeBSD, openSUSELinux, SUSEEnterpriseLinux] virtualCores=4 memory=7.0	«IaaSInstanceType» «Stereotype» A4 «IaaSInstanceType» instanceCategory=[General_Purpose] region=[] operatingSystem=[CentOS, RedHatEnterpriseLinux, Debian, OracleEnterpriseLinux, WindowsServer, SUSELinuxEnterprise, Ubuntu, FreeBSD, openSUSELinux, SUSEEnterpriseLinux] virtualCores=8 memory=14.0
«IaaSInstanceType» «Stereotype» A5 «IaaSInstanceType» instanceCategory=[Memory_Optimized] region=[] operatingSystem=[CentOS, RedHatEnterpriseLinux, Debian, OracleEnterpriseLinux, WindowsServer, SUSELinuxEnterprise, Ubuntu, FreeBSD, openSUSELinux, SUSEEnterpriseLinux] virtualCores=2 memory=14.0 localDisk=127	«IaaSInstanceType» «Stereotype» A6 «IaaSInstanceType» instanceCategory=[Memory_Optimized] region=[] operatingSystem=[CentOS, RedHatEnterpriseLinux, Debian, OracleEnterpriseLinux, WindowsServer, SUSELinuxEnterprise, Ubuntu, FreeBSD, openSUSELinux, SUSEEnterpriseLinux] virtualCores=4 memory=28.0 localDisk=127	«IaaSInstanceType» «Stereotype» A7 «IaaSInstanceType» instanceCategory=[Memory_Optimized] region=[] operatingSystem=[CentOS, RedHatEnterpriseLinux, Debian, OracleEnterpriseLinux, WindowsServer, SUSELinuxEnterprise, Ubuntu, FreeBSD, openSUSELinux, SUSEEnterpriseLinux] virtualCores=8 memory=56.0 localDisk=127	«IaaSInstanceType» «Stereotype» A8 «IaaSInstanceType» instanceCategory=[Compute_Optimized] region=[] operatingSystem=[CentOS, RedHatEnterpriseLinux, Debian, OracleEnterpriseLinux, WindowsServer, SUSELinuxEnterprise, Ubuntu, openSUSELinux, SUSEEnterpriseLinux] virtualCores=8 memory=56.0 localDisk=127 processorArchitecture=[Arch64]	«IaaSInstanceType» «Stereotype» A9 «IaaSInstanceType» instanceCategory=[Compute_Optimized] region=[] operatingSystem=[CentOS, RedHatEnterpriseLinux, Debian, OracleEnterpriseLinux, WindowsServer, SUSELinuxEnterprise, Ubuntu, openSUSELinux, SUSEEnterpriseLinux] virtualCores=16 memory=112.0 localDisk=127 processorArchitecture=[Arch64]

Figure 55: Windows Azure instance types

In the previous image can be seen the set of stereotypes describing the types of instances offered by Windows Azure provider.

The `IaaSInstanceType` stereotype defined in the IaaS profile (see Appendix A) is applied to all of them.

The properties contained in the `InstanceType` permit to specify information such as physical resources offered (virtual cores, memory, local disks), supported operating systems and regions where can be run. This information is very useful in order to be able to automatically check if a concrete instance could cover deployment needs of a specific application.