Title:	*Monitoring Platform Specification*
Authors:	*Danilo Ardagna (Politecnico di Milano), Marco Balduini (Politecnico di Milano), Ciprian Dorin Craciun (IEAT), Nicola Calcavecchia (Politecnico di Milano), Giuliano Casale (Imperial College London), Emanuele Della Valle (Politecnico di Milano), Elisabetta Di Nitto (Politecnico di Milano), Juan Fernando Perez (Imperial College London), Craig Sheridan (Flexiant), Weikun Wang (Imperial College London)*
Editor:	*Giuliano Casale (Imperial College London)*
Reviewers:	*Septimiu Cosmin-Nechifor (Siemens), Dana Petcu (IEAT)*
Identifier:	*D6.2*
Nature:	*Report*
Version:	*1*
Date:	*27/03/2013*
Status:	*Final*
Diss. level:	*Public*

Executive Summary

This deliverable presents the specifications of the MODAClouds *Monitoring Platform*. The objective of this platform is to collect metrics on the status and offered quality of service (QoS) of a MODAClouds application and to analyse the collected monitoring data. The deliverable proposes an architecture and identifies relevant technologies and metrics that will define the *Monitoring Platform*. A validation plan is included in the deliverable.

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 318484 (MODAClouds).

Members of the MODAClouds consortium:

Politecnico di Milano	Italy
StiftelsenSintef	Norway
Institutul E-Austria Timisoara	Romania
Imperial College of Science, Technology and Medicine	United Kingdom
SOFTEAM	France
Siemens Program and System Engineering	Romania
BOC Information Systems GMBH	Austria
Flexiant Limited	United Kingdom
ATOS Spain S.A.	Spain
CA Technologies Development Spain S.A.	Spain

Published MODAClouds documents
These documents are all available from the project website located at http://www.modaclouds.eu/

Contents

ACRONYMS

AOP	Aspect Oriented Programming
API	Application Programming Interface
ARMA	Auto-Regressive Moving Average model
C-SPARQL	Continuous SPARQL
DA	Data Analyser
DC	Data Collector
DTD	Document Type Definition
HDFS	Hadoop Distributed File System
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IRI	Internationalized Resource Identifiers
JMX	Java Measurement Extensions
JPA	Java Persistence API
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MAPE-K	Monitor, Analyze, Plan, Execute, Knowledge components
NTP	Network Time Protocol
PaaS	Platform as a Service
QoS	Quality of Service
RDF	Resource Description Framework
RFC	Request For Comment
SaaS	Software as a Service
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SPARQL	SPARQL Protocol and RDF Query Language (*recursive*)
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Machine
WP	Work Package
XML	Extensible Markup Language
XSD	XML Schema Definition
YAML	YAML Ain't Markup Language (*recursive*)

TERMINOLOGY

Application Cloud	Cloud platform where Cloud App is deployed
Cloud App	Cloud application developed with the *MODAClouds IDE*
Cloud App Admin	Cloud application human administrator
Execution Platform	*Runtime Environment* services for the Cloud app deployment and execution
MODAClouds IDE	MODAClouds Integrated Development Environment
Monitoring Platform	*Runtime Environment* services for monitoring data collection and analysis
Runtime Environment	MODAClouds Runtime services and Cloud App
QoS Engineer	Engineer who specifies at design time the application QoS constraints
Self-Adaptation Platform	*Runtime Environment* services responsible for the Cloud App self-adaptivity
Service Cloud	Cloud platform where the services of the *Runtime Environment* run

1. Introduction

1.1 Context

The QoS management of applications that can execution on multiple Clouds requires the ability to collect metrics on the application state and the processed workload, and use these metrics to decide at runtime adaptations to ensure high QoS. As described in deliverable D6.1, the MODAClouds *Runtime Environment* will implement a MAPE-K loop (monitor, analysis, planning, execution, knowledge). MAPE-K is a reference blueprint for the implementation of self-adaptive applications. The *Monitoring Platform* will be the system, inside the *Runtime Environment*, responsible for delivering the main monitoring and analysis capabilities of the MAPE-K loop. As explained in deliverable D6.1, these two capabilities may be summarised as follows:

- *Monitoring***.** The monitoring component is responsible for managing the different sensors that provide information regarding the performance of the system. In MODAClouds, sensors can capture the current consumption of critical node resources (such CPU and memory) but also other performance metrics (e.g., number of processed requests in a time window and the request process latency). The monitoring granularity is specified by rules. Sensors can also raise notifications when changes to the system configuration happen.
- *Analysis***.** The analysis component is responsible for processing the information captured by the monitoring component and to generate high level events or high-level aggregate information. For instance, it may combine the values of CPU and memory utilization to signal an overload condition in the platform.

Figure 1.1 highlights the *Monitoring Platform* role in the *Runtime Environment* envisioned in the MODAClouds project. Details about the conceptual architecture can be found in deliverable D6.1. The *Monitoring Platform* will receive a set of monitoring rules from the design-time environment (*MODAClouds IDE*) that will specify *what* to measure and will allow to customise *how* to measure it (e.g., monitoring resolution). The *Monitoring Platform* will be then responsible for acquiring information from the running application from its cloud environment and analyse this data with two main aims: 1) generate triggers and data streams for the *Self-Adaptation Platform* of the *Runtime Environment*; 2) feedback relevant monitoring data to the design-time IDE, with the aim of providing to the application developer and the application *QoS Engineer*, decision support and insights on application behaviour at runtime. The last scenario also includes the case where the application has been deployed on a sandbox, i.e. a controlled test environment, with the aim of calibrating the design-time tools and models based on initial runtime information.

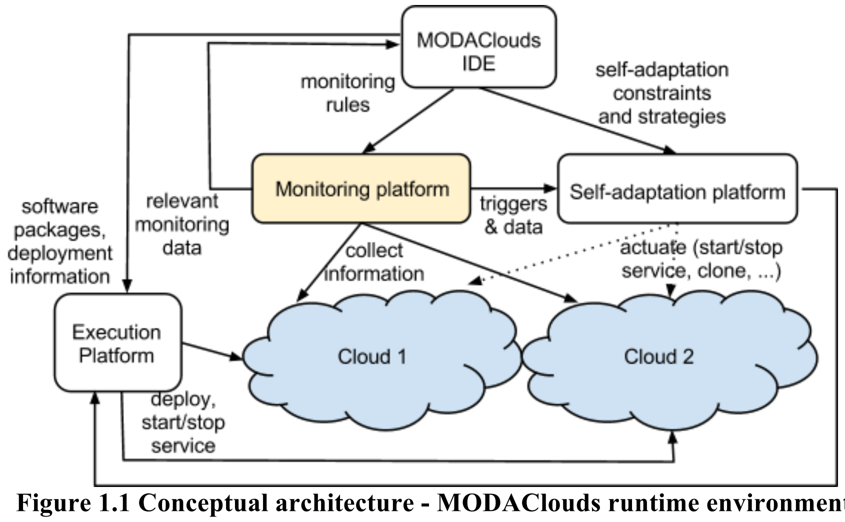


Figure 1.1 Conceptual architecture - MODAClouds runtime environment

The present deliverable focuses in particular on the specification of:

1. The general approach to monitoring (Section 2)
2. The definition of a subset of reference metrics at infrastructure-level (Section 3)
3. The definition of a subset of reference metrics at application and container level (Section 4)
4. The general approach to monitoring data analysis (Section 5)
5. The relevant technology stack (Section 6)

Finally, Section 7 includes an implementation plan and a validation that will drive the work activities throughout the upcoming months.

2. MODAClouds Monitoring Platform

2.1 Aims

In a multi-cloud environment, a *Monitoring Platform* has to face challenges due to the intrinsic complexity of the target clouds including, among others:

- Data and cloud provider heterogeneity. For example, monitoring data collected on IaaS and PaaS may have different formats and may be collected using different tools.
- Different level of abstractions. For example, data may be provided at host level, virtual machine (VM) level, container level, application level, etc.
- Time granularity. Due to overhead concerns, data may be collected at coarse granularity (hours) or finer resolutions (minutes, seconds).
- Differences in comparing nominal measurement units (e.g., GHz vs. Amazon ECU).

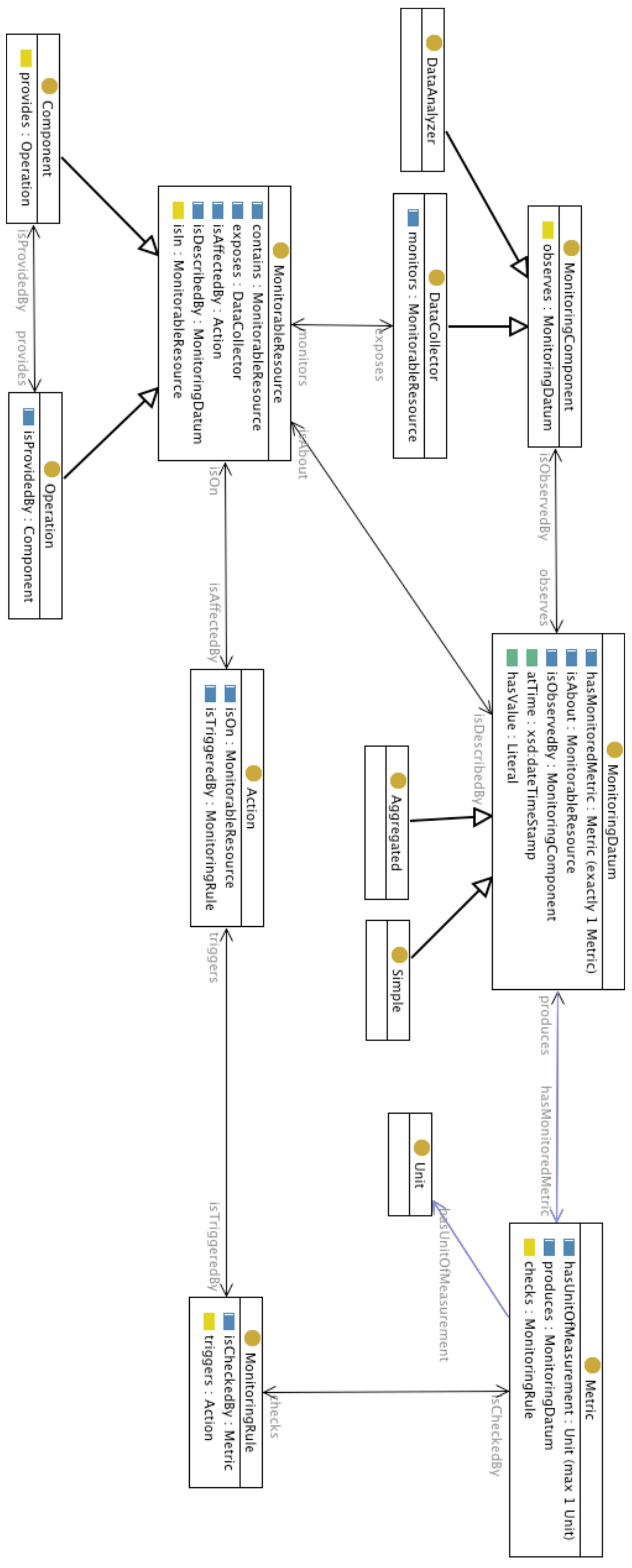
To be effective and facilitate cloud application management, a *Monitoring Platform* should provide a *unified view* of the application state and its offered quality-of-service (QoS). MODAClouds will tackle this challenge by providing a flexible and extensible *Monitoring Platform* architecture. The tasks of the *Monitoring Platform* will not be confined to the mere collection of infrastructure-level metrics from different sources (e.g., Host, VMs, etc.) or application-level metrics (e.g., response time, availability, etc.). Rather, the *Monitoring Platform* will include specialised data collectors and data analysis algorithms that will leverage on design-time knowledge about the application internals. This *white-box* approach will represent a major innovation of MODAClouds with respect to the other cloud monitoring solutions, which are normally *black-box*, and it will be a distinguishing feature of the project thanks to its MDE approach to cloud application development.

2.2 Main concepts relevant to the monitoring platform

Figure 2.1 shows the main concepts that are relevant for the *Monitoring Platform*. The purpose of the monitoring activity is defined by a set of Monitoring Rules that checks a set of Metrics observed by Monitoring Components in the form of Monitoring Datum. Each Monitoring Datum represents a time stamped measurement of a specific Metric in a given Unit. A Monitoring Datum can be Simple or Aggregated (i.e., a single point versus an aggregation of multiple points using some aggregation function). Monitoring Components can be either Data Collectors, which monitors one or more Monitorable Resources, or more abstract Data Analyzers that analyze the observed data.

Entire Virtual Machines (VM), Components in container or single Operations are examples of Monitorable Resources. A Monitorable Resource can contain other Monitorable Resources, e.g., a monitored VM can contain a monitored component. When a Monitoring Rule (e.g., the average number of served Web pages for a HTTP server has to be above 50 pages per minute) fails, it triggers one or more Actions on the Monitorable Resource it is about.

Figure 2.1: MODAClouds *Monitoring Platform* main concepts.



2.3 Monitoring platform interface

The interface between the *Monitoring Platform* and the other components of the MODAClouds architecture is defined according to the use cases described in deliverable D6.1 and shown in Figure 2.2. In particular such interface contains the following operations:

- *InstallMonitoringRule*: Monitoring rules produced by the *MODAClouds IDE* define the object to be monitored, which metric should be gathered, and which action should be executed in case the produced data are out of the expected range. This operation allows the installation of a new monitoring rule in the *Monitoring Platform*. The installation process involves various checks to ensure that the rule can be actually executed (i.e., associated data can be gathered and the corresponding computations executed).
- *ActivateMonitoringRule*: Once a monitoring rule is installed in the *Monitoring Platform*, the *ActivateMonitoringRule* operation effectively activates the rule so that it will receive monitoring data and perform evaluations on it.
- *DeactivateMonitoringRule*: this operation is the dual with respect to the previous one, i.e., it deactivates a monitoring rule. Activation and deactivation operations for monitoring rules provide a convenient level of flexibility at runtime allowing a quick change of the monitoring rules (and thus self-adaptations).
- *AddObserver*: An observer represents any software component that needs to receive information from the *Monitoring Platform*. The *Monitoring Platform* maintains a list of observers and updates them according to their interests. The *AddObserver* operation adds a new observer to the list of existing observers.
- *RemoveObserver*: this operation is the dual of the previous one. Its effect is to remove a given observer (specified by a specific reference) from the *Monitoring Platform*.

Collect Monitoring Data and *Distribute Data* are the main operations executed by the *Monitoring Platform*. The *Collect Monitoring Data* is executed by the Data Collectors. It is either activated periodically or on a push basis and aims at gathering information from data sources.
Distribute Data is executed by Data Analyzers and aims at distributing to the observers data that comply with the monitoring rules active in the *Monitoring Platform*.

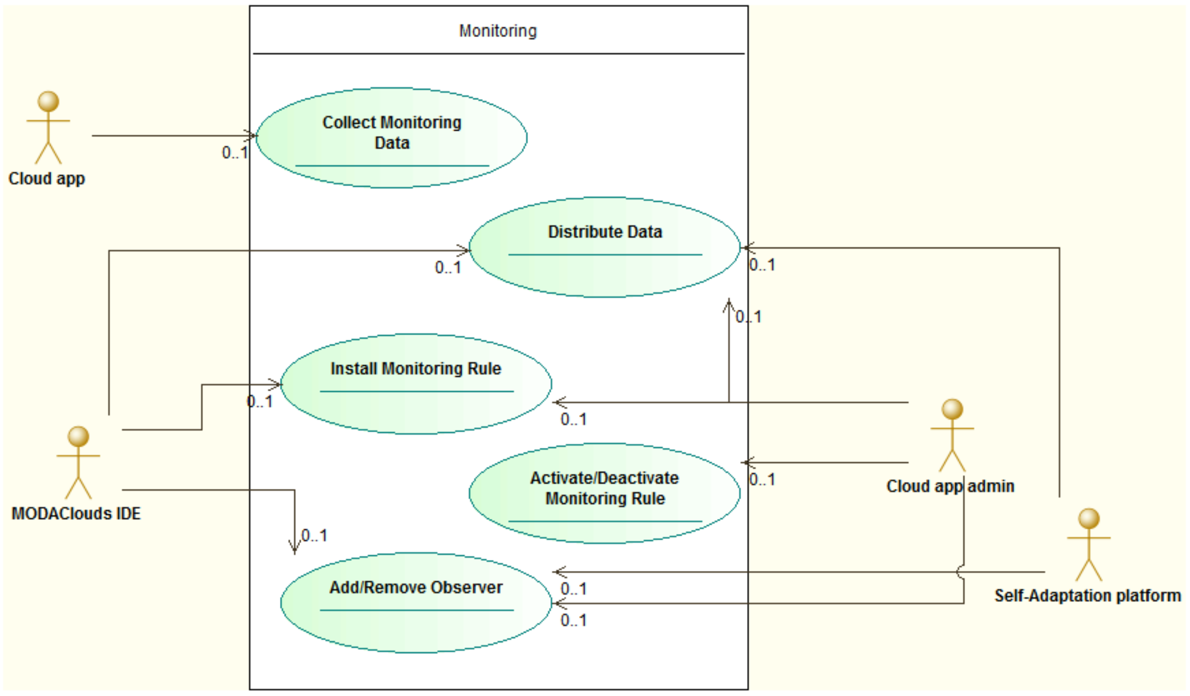


Figure 2.2: MODAClouds *Monitoring Platform* use cases.

2.4 Monitoring Architecture Overview

The monitoring system architecture is depicted in Figure 2.3 and its components are described in the following subsections.

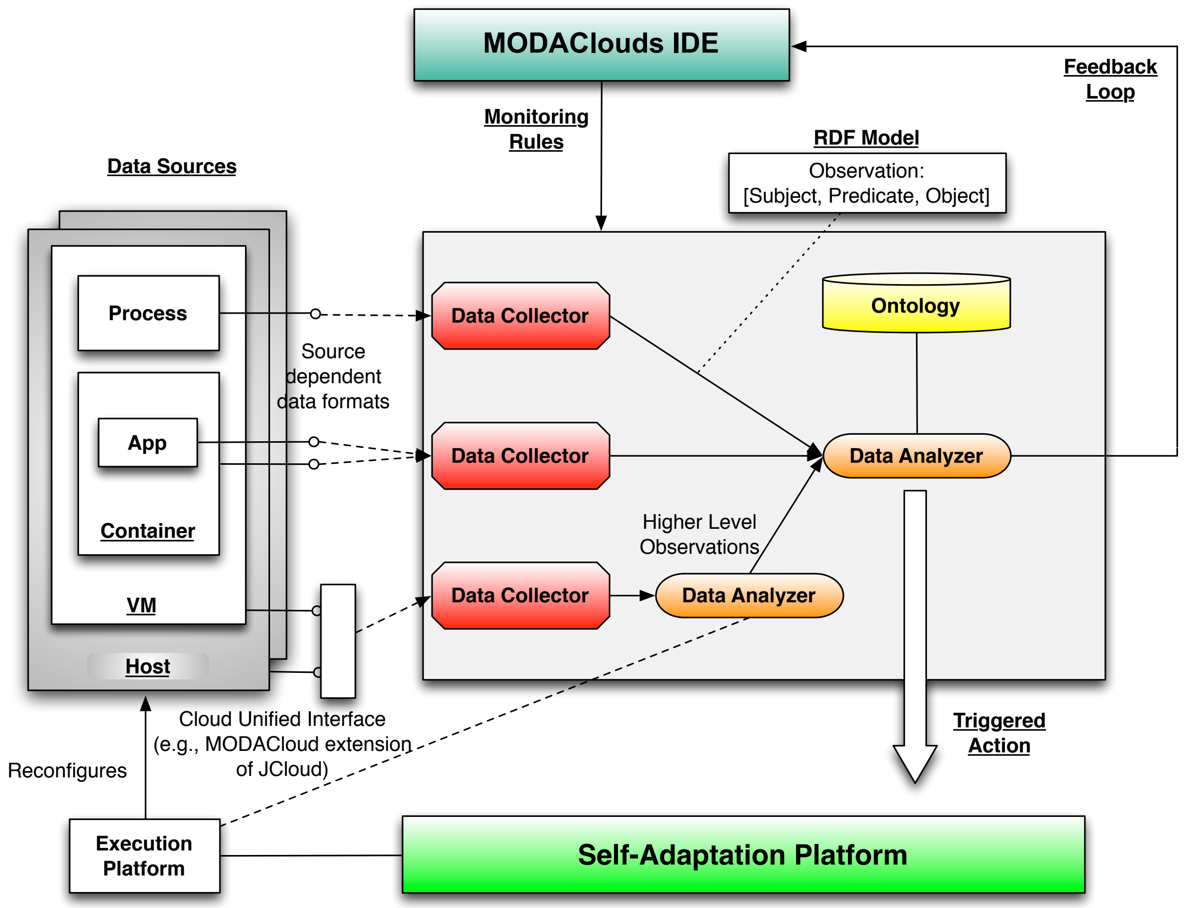


Figure 2.3 Monitoring Architecture

MODAClouds IDE and Monitoring Rules

Monitoring rules are defined at design time either manually by the Cloud app developer or administrator or automatically by the *MODACloudsIDE* that derives them from the definition of the non-functional requirements (for example constraints on response time, availability, etc.) specified for the cloud-based application. The specific language defined for monitoring rules as well as the mechanisms for their automatic generation are part of WP5 and will be reported in *Deliverable D5.2.1*. For the purpose of the *Monitoring Platform* defined in this deliverable we show some examples of monitoring rules. Their syntax is not to be considered final but they are useful to give an idea of the kinds of requirements the *Monitoring Platform* will be able to fulfil.
An example of monitoring rule involving a system composed by a web server and a database and four monitoring rules is reported in Figure 2.4. In particular, two rules concern the webserver: the first one is a hard constraint on the average value of the response time for the page *getAccountSummaryPage* while the second one is a soft constraint on the 90th percentile of the response time.
In the *Monitoring Platform* monitoring active monitoring rules continuously process the input monitoring data and evaluate the corresponding conditions. The activation of non-active rules can be triggered by the violation of some active rule, according to the activation dependencies specified by the developer at design time. In our example a dependency exists between *MonitoringRuleGetAccountInformationMR* rule and the *getAccount-SummaryPage.* In particular, whenever the response time constraints are not respected (either on average or on percentile) the *Monitoring Platform* automatically activates also the *MonitoringRuleGetAccountInformationMR* monitoring rule.

-

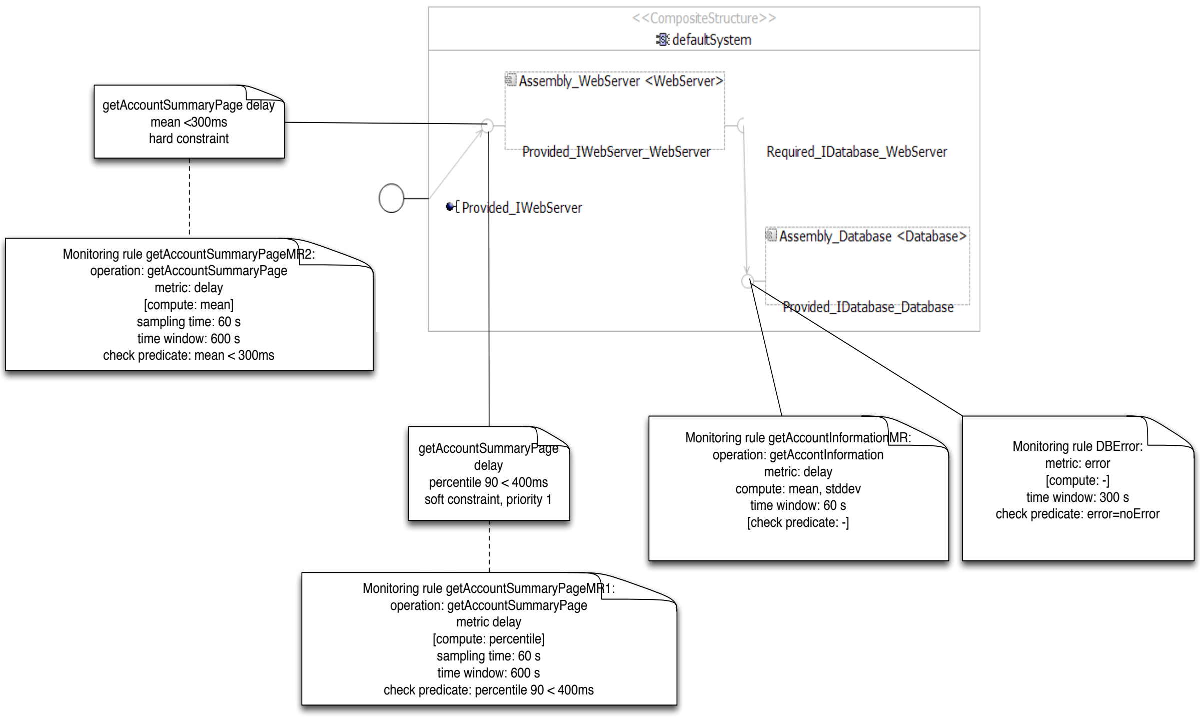

<<CompositeStructure>>
defaultSystem
Assembly_WebServer <WebServer>
Provided_IWebServer_WebServer
Required_IDatabase_WebServer
Provided_IWebServer
Assembly_Database <Database>
Provided_IDatabase_Database
getAccountSummaryPage delay
mean <300ms
hard constraint
Monitoring rule getAccountSummaryPageMR2:
operation: getAccountSummaryPage
metric: delay
[compute: mean]
sampling time: 60 s
time window: 600 s
check predicate: mean < 300ms
getAccountSummaryPage
delay
percentile 90 < 400ms
soft constraint, priority 1
Monitoring rule getAccountInformationMR:
operation: getAccontInformation
metric: delay
compute: mean, stddev
time window: 60 s
[check predicate: -]
Monitoring rule DBError:
metric: error
[compute: -]
time window: 300 s
check predicate: error=noError
Monitoring rule getAccountSummaryPageMR1:
operation: getAccountSummaryPage
metric delay
[compute: percentile]
sampling time: 60 s
time window: 600 s
check predicate: percentile 90 < 400ms

Figure 2.4 An example for a monitoring rule

Data Collectors

Data Collectors (DC) have two main objectives:

- They gather the monitoring data produced by the various data sources;
- They associate semantic information to the monitored data. This is possible thanks to the structured nature of the data format that will be used to transport the data (RDF).

Data collectors will deal with the heterogeneity of different data sources by producing data under the form of an RDF triple. For some data sources, the data collector may rely on other solutions providing a unified interface over some cloud aspects. For example, libraries such as jclouds are able to interface with multiple cloud providers using the same API. Cloud-proprietary APIs and services (e.g., Amazon CloudWatch API) may be directly invoked as an alternative. The decision of the best approach will be taken experimentally, by direct comparison of alternatives to understand the best trade-off between overhead and implementation complexity. However, in either case the implementation details will be hidden by the fact that each data collector will communicate with the rest of the system only via RDF.

To be as general as possible, the data monitored by the data sources will be retrieved following two strategies: *push* and *pull*. In the push strategy, data sources actively send (i.e., push) data to the data collectors, which passively wait for them. In order to perform a push data collection, data sources must know a data collector endpoint. In the pull strategy, data collectors periodically query (i.e., pull) the various data sources and retrieve data from them. While the push-based approach integrates more naturally with the stream-based architecture of the *Monitoring Platform*, it cannot be adopted in all cases as not all data sources support this strategy (consider for instance the case of a closed-source component offering a SNMP interface). In these cases, a pull-based approach will be adopted. The actual used strategy will then be identified case by case and the needed logic will be embedded in the data collectors. The other parts of the *Monitoring Platform* will see the data streamed out by the data collector in the RDF format and will not be aware of how these data have been retrieved.

Monitoring Data

Monitoring data refers to the actual data produced by data sources. The requirements for the monitoring data concern its representation in a general, interoperable, easily-understandable and easily-processable format. For this reasons, the data will be structured using the RDF model (Resource Description Framework). RDF organizes the information using a subject-predicate-object triple. In particular, the subject identifies the source of

the information (for example a specific application in a container), the predicate defines the aspect monitored (for example the response time) while the object specifies the value of the predicate (for example 300 milliseconds). This model offers a great deal of flexibility and allows for arbitrarily complex reifications. Subjects of the triples are generic URIs identifying resources over the web while the syntax used to describe RDF triples is XML (i.e., open and widely supported).
Figure 2.5 demonstrates an RDF stream example in which the "subject" is "InstanceId", the "predicate" is "CPUUtilization" and the "object" is "3.0".

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:j.0="http://mystream/" >
<rdf:Description rdf:about="http://mystream/InstanceId">
  <j.0:CPUUtilization
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">3.0</j.0:CPUUtilization>
</rdf:Description>
</rdf:RDF>
```

Figure 2.5: RDF Monitoring Data Example

Another example of monitoring data in RDF format is shown in Figure 2.6. The target of the monitoring is the home page of a web site hosted in a PaaS system. In the first part of the stream the "subject" is specified as "http://paas.cloudprovider.com/home.jsp", the "predicate" specified as "time" and the "object"s are the values registered in milliseconds for individual requests.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<rdf:Description rdf:about="http://paas.cloudprovider.com/home.jsp">
  <valueURI:time> 100 </valueURI:time>
  <valueURI:time> 302 </valueURI:time>
  <valueURI:time> 227 </valueURI:time>
  <valueURI:time> 120 </valueURI:time>
</rdf:Description>
</rdf:RDF>
```

Figure 2.6: RDF Monitoring Data Example

As the examples show, the source code is easily readable thanks to the text encoding and the simplicity of the schema itself (i.e., triple format). In particular, it is immediate to understand what is the subject of the triple (i.e., rdf:about attribute) and the value-predicates pair for that subject (i.e., children of the rdf:Description element). XML parsing is a standard task and typically all parsing techniques produce a DOM (Document Object Model) representing the content of the file using a tree-like data structure. Moreover, XML provides two largely known languages to define XML schemas (i.e., XSD and DTD). XML can also be used for transferring RDF data, but other serialization formats are gaining popularity (for example JSON and YAML) and could be adopted in MODAClouds without major impact on the design of the *Monitoring Platform*.

Data Analyzer

The *Monitoring Platform* will be populated by a set of data analyzers (DAs). A DA is a basic unit to processes monitoring data. At a high-level of abstraction, a DA may be seen as accepting in input a continuous stream of monitored data (in RDF format) and producing in output *one or more* streams of data (in RDF format). At a finer granularity, the main duties of a data analyzer include the generation of one or more output streams providing a higher level of abstraction with respect to the given input. For example data at different time granularities may be aggregated to produce a homogeneous view over the data at a longer time scale. Similarly, a DA may perform indirect estimation of unobservable or missing data and output this on new streams.
Since DAs will be heterogeneous and serve different purposes, we distinguish DAs into four categories:

- *Detection DAs* will be responsible to detect a violation in a QoS constraint (e.g. SLA) of a measured metric and raise a trigger to the *Self-Adaptation Platform*.

- *Correlation DAs* will establish a relationship between measurements collected on different components of the application, with the aim of generating measures that summarize component runtime execution correlations.

- *Estimation DAs* will estimate QoS metrics of the system within a time horizon specified by the *MODAClouds IDE* that are not directly observable by the data collectors, or that cannot be observed due to overhead concerns.

- *Forecasting DAs* will forecast, using statistical and stochastic methods, trends for some of the metrics needed by the *Self-Adaptation Platform* to manage the application QoS.

Within the above conceptual categorization, it is useful to remember that a single software component may serve in multiple roles. For example, detection and correlation may be done by the DAs.

A critical ingredient of the data analysis is the time correlation. Given a set of metrics recorded in a time window, the DAs may indicate correlation between these metrics based on their timestamps. This raises important *time synchronisation* issues for a distributed application. In virtualized systems the clocks of the different VMs can lose synchronization when deployed on heterogeneous and geographically separated hosts. An initial solution to deal with this issue in VMs will be to run the network time protocol (NTP), a protocol for clock synchronization over a network, specified by the IETF on RFCs 5905, 5906, 5907 and 5908 (in its latest version 4). When noticing that synchronization has been lost, a VM will attempt to slowly regain it by slowly correcting its clock. This is done to avoid jumps in the clock that may cause problems to the applications running on the VM. A PaaS application may be more constrained, thus we plan to investigate this in more details in upcoming experimental tests. In case the problem is found to be a serious issue in PaaS, we may consider the use of vector clocks.

The task of the data analyzers will be supported by the presence of a monitoring *ontology* that provides information about the various relationships among the gathered data. An ontology formally represents the knowledge of a specific domain. Ontologies define a set of common concepts and the relationships among them. From this knowledge it is then possible to apply automated reasoning techniques allowing for the extraction of further knowledge. In the context of the MODAClouds monitoring architecture, ontologies will support the tasks of monitoring rule writers and of rule execution. In particular, monitoring rules writers rely on the formal model represented by the ontology to take advantage of existing concepts and their relationships. The ontology models the concepts in Figure 2.1. The DAs will then exploit this information at runtime for data processing. Section 5 provides, through an example, some more explanation on the use of ontologies in the *Monitoring Platform*.

C-SPARQL
Several DAs will be implemented using C-SPARQL, a continuous query engine for RDF data. C-SPARQL provides an efficient, yet expressive tool to execute queries on data streams based on the system ontology. C-SPARQL queries correspond to our monitoring rules. They work on time windows i.e., the most recent triples of a given stream, observed while the data is continuously flowing. The support for RDF format allows for a great deal of flexibility and interoperability. Moreover, C-SPARQL reasoners can deal with knowledge which may evolve over time. The trade-off with flexibility comes with a small price with respect to performance costs, since tailored solutions can achieve better performance at runtime (but higher design and development costs). Given the multi-cloud context of MODAClouds project it becomes critical the adoption of an extensible yet customizable solution such as the one represented by C-SPARQL.
Since a data analyzer both consumes and produces RDF data, it is possible to derive complex topologies of DAs thus leaving maximum flexibility for the MODAClouds multi-cloud context.
A data analyzer consumes resources that are additional with respect to the actual work carried out in the cloud. For example, if monitoring data traverse different availability zones of the same cloud the provider typically charges for the traffic. In order to reduce such costs, data is only sent if it is necessary and specific conditions are met on the data collectors.

Feedback Loop

A monitoring feedback loop will carry information about the application execution to the *MODACloudsIDE* for further analysis and display to the *QoS engineer* or the *Cloud App* developer. This system will be developed as part of WP5 and therefore its specification is not included in the present document.

2.5 Metrics

Monitored data will characterize different aspects of MODAClouds applications and of the platform and infrastructure on which they will run. To describe the object of the measurement and its higher-level aggregates, throughout the deliverable we will adopt the following terminology:

- A *metric* will be defined by two components: and a unit of measurement (e.g., percentage) and a monitoring datum. A metric will be measured from a system and will normally refer to a tangible quantity, for example response time, throughput, uptime.
- A *MonitoringDatum* links the metric with different aspects of the *Monitoring Platform*. In particular, a *MonitoringDatum* contains a reference to a metric, refers to a *MonitorableResource* (i.e., a component able to generate information), is observed by a *MonitoringComponent* (i.e., a data collector) and contains a timestamp referencing the time at which the *MonitoringDatum* has been produced. *MonitoringDatum* can be *simple* (e.g. a single value generated by data collectors) or *aggregated* (e.g. an aggregate set of values produced by a DA).
- A *log* is a specific kind of aggregated Monitoring Datum. It provides textual information about the application execution. For example, stack trace details about a Java exception. The *Monitoring Platform* will support also the distribution of selected application logs, particularly pertaining to errors and incidents that need to be reported to the cloud application administrator.

A *high-level metric* is a quantitative or qualitative indicator that is obtained by processing of a set of one or more metrics and/or logs. For example, the average response time is a high level metric resulting from the processing of the response time punctual values. Pearson's correlation coefficient ρ(X,Y) is a high-level metric derived from any two input metrics X and Y. However, it cannot be measured directly from a system. This, in our terminology, is a high-level metric. Similar considerations apply to indexes, such as the service measurement index (SMI) (see D6.1). Other metrics such as the CPU Utilization instead can either be measured directly or derived from other metrics, depending on the interface offered by Monitorable Resource.

In the rest of this deliverable, metrics will be broadly classified at two different levels of the abstraction:

- *Infrastructure-level*: metrics refer to elements of the cloud infrastructure (e.g., hypervisor, VM)
- *Application-level*: metrics are application specific and characteristics of the application itself of its container (e.g., JVM).

For PaaS applications, the *Monitoring Platform* will produce only application-level metrics, unless the target cloud supports also infrastructure-level metrics. Monitoring data will be retrieved using different tools and techniques, as we describe in sections 3 and 4.

2.6 Timescales

The various services provided by the MODAClouds Monitoring Data Analyser, presented in the previous sections, expand different timescales. For instance, while anomaly detection will typically be performed continuously, with executions in the order of minutes, forecasting and estimation can provide useful information to the Cloud App Administrator for longer term planning. The MODAClouds Analyser identifies three main timescales:

- *Detection timescale*: This is the shortest timescale and aims at detecting anomalies and determining their possible causes, allowing the *Self-Adaptation Platform* to timely respond and take corrective actions when needed. This will typically be done in the order of minutes.
- *Capacity planning timescale*: At this timescale, all the services (detection, correlation, estimation and forecasting) can be used to identify, characterise and forecast changes in the workload mix. The result of these analyses will then be delivered to the *MODAClouds IDE*, and to the self-adaptive platform, to determine the appropriate number and allocation of resources (e.g. VMs or containers).These procedures will be done on an hourly or multi-hourly basis, as cloud resources such as VMs can currently be negotiated as a minimum on an hourly basis.
- *Decision support timescale*: This is longest timescale considered, and is expected to be performed on a daily, weekly or monthly basis. The emphasis in this case will be in providing an overview on the application non-functional behaviour, such that the Cloud App Administrator can take high-level decisions, beyond the reach of the Self-adaptive platform. At this timescale, the Analyser will provide the *MODAClouds IDE* with information to deliver reports that include the anomalies observed during the observation horizon, their causes, the overall performance and availability indices, the current and forecasted workload mix, among others.

Only the detection and capacity planning timescales will be relevant to the *Monitoring Platform* developed in WP6. The decision support timescale will be more relevant to WP5 and the *MODAClouds IDE*.

2.7 Interaction with Other Platforms

Interaction with the Self-Adaptation Platform

The *Self-Adaptation Platform* receives the aggregated information generated by the data analyzers and, possibly, determines a new configuration for the system updating the runtime model of the system used by the data collectors. As a matter of fact, it is the component in charge of planning complex reconfiguration of the clouds. Components of the *Self-Adaptation Platform* establish an interaction with the *Monitoring Platform* by registering themselves as observers of monitoring data. From the time they become observers, they start receiving the data that are relevant to the computation they have to perform. The *Self-Adaptation Platform* can also be proactively triggered by the *Monitoring Platform* whenever a monitoring rule appears to be violated.

Interaction with the Execution Platform

The *Execution Platform* represents the actuator of the system (i.e., the component in charge of effectively apply the adaptation). The *Execution Platform* can receive commands from the *Self-Adaptation Platform* (sequence of adaptation actions) or directly from the data analyzers (simple adaptation actions). Among the requirements of the *Execution Platform* is to support the creation of sandboxes, where the application could run for initial deployment, tests, debugging. The *Execution Platform* should inform the *Monitoring Platform* of this special deployment mode, so that high resolution measurements can be acquired without overhead concerns.

3. Infrastructure-Level Monitoring

In this section, we discuss the metrics that will be monitored at the infrastructure level to achieve the technical objectives of the MODAClouds *Monitoring Platform*. Explanations are given on why each metric is significant in general, and why it important to collect in the context of MODAClouds. We also identify a set of monitoring tools and probes that will be evaluated to acquire infrastructure-level metrics in the *Monitoring Platform*.

3.1 Approach

An important monitoring challenge that arises in MODAClouds is to tackle the different monitoring constraints imposed by different target clouds. IaaS and PaaS platforms offer very different metrics and therefore the *Monitoring Platform* has to cope with this heterogeneity. For example, in some hosted PaaS platforms (e.g., Google App Engine) it is not possible to execute code that accesses container-level statistics, for example code calling the Java Management Extensions (JMX) libraries. Also, on PaaS platforms, since the resource layer cannot be explicitly accessed by the application, the user is typically unable to collect CPU utilization and other statistics from VM, host, JVM, etc. On a IaaS platform, monitors face similar problems regarding the collection of host resource statistics, for example in establishing how many VMs are running on a host.

A possible approach to face these limitations consists in selecting a set of metrics that are common to all the targeted IaaS and PaaS clouds and require the *Monitoring Platform* to collect only these metrics. While this is a simplifying solution, it does not appear particularly appealing to achieve the QoS management and QoS design goals of MODAClouds. For example, ignoring CPU utilization data on IaaS platforms is unnecessary and could affect the quality of the adaptive decisions that the MODAClouds *Runtime Environment* will take to manage IaaS application QoS. On the other hand, support for metrics that are specific to a single cloud provider may make it difficult to develop general-purpose management algorithms independent of the target cloud.

To address these issues, the MODAClouds *Monitoring Platform* will explicitly distinguish between *infrastructure-level metrics*, that are exposed only on IaaS clouds, from *application-level metrics*, that will rely on monitoring probes injected in the application code and therefore can be collected on any target cloud. In particular, the focus at the infrastructure-level will be on *VM resources*, and if exposed by the provider on *host resources*. We will give lower priority to finer metrics related for example to virtual core counters (e.g., cache faults), low-level disk I/O information (e.g., merged requests), kernel counters. Albeit informative, low-level metrics can introduce performance overheads in the measurement. Furthermore, techniques that use these fine-grained metrics for QoS management have been suggested in the literature, but these methods are not popular yet. A special focus will be given in particular to VM resources, such as measures of utilization, memory, storage, or network, since these are widely available across today's cloud offering.

On top of the collection of VM resource metrics, the *Monitoring Platform* will also deliver advancements concerning the measurement of host resources. As of today, only few cloud platforms expose their monitoring data at the host level to the customers. Amazon CloudWatch is a noticeable example, since cloud customers can access host-level measurements of CPU, storage, and network under payment of a fee. These metrics are particularly useful to compensate inaccuracies in the monitoring data that are common in VMs. However, no information is given on the host environment in which the VM operates, for example the number of VM sharing the same host. Features of this kind are available only on experimental testbed (e.g., the EU project Bonfire). Furthermore, the data collection is supported only at coarse resolutions of minutes. Together, these issues increase uncertainty on the state of an application and, ultimately, reduce the ability of cloud applications to adapt their resource allocation at runtime in an optimal manner. Moreover, limited research has been conducted to investigate the impact of host-level cloud monitoring tools on QoS adaptation at the application level. As a proof of concept, the MODAClouds *Monitoring Platform* will be paired with new host resource monitoring tools provided by Flexiant on their infrastructure that will offer a finer grained control from application perspective of the host monitoring resolution. We will use this feature to explore the benefits in clouds of fine-grained host level monitoring. Thanks to the general architecture provided by the MODAClouds monitoring platform, this feature can be implemented in other cloud providers.

Similar to threads, the monitoring of processes can result in performance penalties. To cope with this, we will focus on process summary metrics, for example maximum number of processes in running state in a time window. If not too costly in terms of overhead, we plan to add to these only very metrics for threads (e.g. maximum number of threads observed in a period). However, we will try to minimize access to detailed process-specific information which is generally more costly and often difficult to exploit in general-purpose QoS management.

Infrastructure-level metrics from VM resources, host resources, and processes will be first acquired by probes and successively passed to data collectors that will translate them into the RDF format suitable for transportation and querying by C-SPARQL. In some cases, the C-SPARQL data collectors may directly acquire the data from the monitored system without relying on other tools. Several probes are available over the internet that can return measurements of QoS metrics, a survey of popular monitoring tools has been given in deliverable D6.1.

3.2 Infrastructure Metrics

A large number of infrastructure-level metrics can be collected and their utility in QoS management is different across application classes. Therefore, system administrators tend to prefer to focus on small sets of core metrics that are simple to interpret and do not cause excessive monitoring overheads. On the other hand, monitoring tools such as *collectl* can return many tens of metrics once interfaced to a *Monitoring Platform* without the need of defining complex interfaces for each metric. *collectl* [COL13] is a mature performance monitoring tool, used often in conjunction with Ganglia to monitoring large scale systems. We plan to use *collectl* as main tool for VM resource monitoring, however for specific metrics we could resort to other tools if they prove to offer specific advantages, for example *SIGAR* that integrates natively with the JVM. We will postpone the decision based on experimental data concerning overheads.

As a result of this, we outline here the main metrics that the *Monitoring Platform* will collect, but additional metrics may be added in the future depending on case study and Self-Adaptive Platform requirements. Furthermore, at the host level we consider Amazon EC2 and Flexiant as target platforms, additional platforms may be added level to this list if they support host resource measurement.

Table 3.2.1 shows metrics that are measurable for VM and host resources *both*. These metrics introduce a trade-off for the *Monitoring Platform* between quality of the measurement, which can be enhanced by comparison of VM and host resource measurements, and potential costs incurred, e.g. Amazon CloudWatch fees. Due to their general applicability, these metrics cover only the fundamental aspects of CPU, disk access and networking.

Table 3.2.1 Metrics measurable at both VM and Host

Resource	**Resources**	**Definition**	**Relevance to MODAClouds**	**Tools**	**Priority**
CPUUtilization	Host, VM	Total CPU utilization. Normalized in 0-1. At host level the metric refers to the application VM.	Manage application CPU scalability.	CloudWatch, Flexiant, collectl	Must Have
DiskWriteBytes	Host, VM	Number of bytes written to disk since last measurement. At host level the metric refers to the application VM.	Manage application disk I/O scalability.	CloudWatch, collectl	Must Have
DiskReadBytes	Host, VM	Number of reads written to disk since last measurement. At host level the metric refers to the application VM.	Manage application disk I/O scalability.	CloudWatch, Flexiant, collectl	Must Have
DiskWriteOps	Host, VM	Number of write operations to disk since last measurement. At host level the metric refers to the application VM.	Manage application disk I/O scalability.	CloudWatch, collectl	Must Have
DiskReadOps	Host, VM	Number of read operations to disk since last measurement. At host level the metric refers to the application VM.	Manage application disk I/O scalability.	CloudWatch, collectl	Must Have
NetworkInBytes	Host, VM	Number of bytes received over the network interfaces since last measurement. At host level the metric refers to the application VM.	Manage application network I/O scalability.	CloudWatch, Flexiant, collectl	Must Have
NetworkOutBytes	Host, VM	Number of bytes sent over the network interfaces since last measurement. At host level the metric refers to the application VM.	Manage application network I/O scalability.	CloudWatch, Flexiant, collectl	Must Have

Table 3.2.2 illustrates metrics that will be collected only at the level of the VM, either virtual resources or software resources of the operating system. Besides metrics related to disk space and memory usage, the list includes metrics related to the maximum number of running processes and threads observed in a time period, and low-level metrics such as interrupts and context switches which have been found in the literature to be useful for quantifying cloud CPU overheads in VMs [Dee11].

Table 3.2.2 VM-Only Metrics

Resource	**Resources**	**Definition**	**Relevance to MODAClouds**	**Tool**	**Priority**
MemUsed	VM	Number of bytes in memory at the measurement instant.	Establish presence of memory bottlenecks.	collectl	Must Have
CtxSw	VM	Total number of CPU context switches since last measurement.	Establish presence of CPU contention bottlenecks.	collectl	Should Have
MaxProcs	VM	Maximum number of processes running simultaneously since the last measurement	Model contention at resources.	collectl	Should Have
MaxThreads	VM	Maximum number of threads running simultaneously since the last measurement	Model contention at resources.	collectl	Should Have
DiskSpaceUsed	VM	Total used disk space at the measurement instant.	Establish presence of storage space bottlenecks.	collectl	Should Have
DiskSwapSpaceUsed	VM	Used disk swap space at the measurement instant.	Establish presence of swap storage space bottlenecks.	collectl	Should Have
Interrupts	VM	Total number of CPU interrupts since last measurement.	Establish presence of CPU contention bottlenecks.	collectl	Could Have

Finally, we list in Table 3.2.3 metrics for host resources that will be exposed by Flexiant experimental monitoring probes. Note that we do not plan to include disk metrics on Flexiant's infrastructure because it uses diskless nodes. While several metrics could be exposed by a cloud provider, one should take into account confidentiality and security constraints. For example, exposing low-level details about an infrastructure may be used to reverse engineer the infrastructure management practices of a provider to its competitors. To avoid this, we have limited the list of host metrics to normalized quantities and to the contention level expressed as VMs running on the same node. No VM-specific information is included in the list.

Table 3.2.3 Host-Only Metrics

Resource	**Resources**	**Definition**	**Relevance to MODAClouds**	**Tools**	**Priority**
HostRunningVMs	Host	Number of VMs running in the host.	Model host resource contention.	Flexiant	Should Have
HostCPUUtilization	Host	Total CPU utilization of the host. Normalized in 0-1.	Model host resource contention.	Flexiant	Should Have
HostNetworkInUtilization	Host	Total fraction of incoming bandwidth used by the host. Normalized in 0-1.	Model host resource contention.	Flexiant	Could Have
HostNetworkOutUtilization	Host	Total fraction of outgoing bandwidth used by the host. Normalized in 0-1.	Model host resource contention.	Flexiant	Could Have

4. Application-Level Monitoring

4.1 Application-Level Metrics

Differently from infrastructure level monitoring, monitoring at application level involves some knowledge of the application being monitored (i.e., components, software servers, etc.). Inside the application a number of *events* take place affecting the state of these two components. Metrics are defined upon these events to describe the state of the system and the historical evolution of it. Metrics may be divided according to different categories (as described in Section 2.5). Application level metrics may also be characterized into functional metrics, presenting proper working/functionality, and QoS metrics.
Table 4.1 presents a general set of application-level events. We distinguish between the monitored software entity (e.g., request, user session) from the scope in which it is monitored: application, software server (e.g., database server), individual requests and calls. Furthermore, textual log data is intended here as a special event class that the *MonitoringPlatform* should be able to distribute to the Observers (e.g., Cloud application administrator).

Table 4.1 Application-Level Events

Metric	**MonitorableResource**	**Resource**	**Definition**	**Priority**
Arrival	Session, Request	Cloud Application, Server, Component	MonitorableResource arrives at a resource	Must Have
SyncCall	Request	Server, Component	MonitorableResource is served by resource and makes a synchronous call to another resource	Could Have
AsyncCall	Request	Server, Component	MonitorableResource is served by resource and makes an asynchronous call to another resource	Could Have
Admission	Request	Server, Component	Resource has started serving MonitorableResource	Should Have
Completion	Session, Request	Cloud Application, Server, Component	Resource has finished serving MonitorableResource	Must Have
Busy	N/A	Server, Component	Time the resource has been busy since last measurement	Must Have
Error	Session, Request, Application, Component	N/A	MonitorableResource suffered an error (e.g., exception)	Must Have
Failure	Session, Request, Application, Component	N/A	MonitorableResource suffered a failure (e.g., timeout, QoS violation)	Should Have
Log	Arbitrary	N/A	Arbitrary textual information about an arbitrary MonitorableResource	Should Have

The DCs may distribute directly either information concerning these low-level events, or for computational efficiency provide directly higher level application metrics. For example, in one possible implementation of the platform, a DC may generate a *MonitoringDatum* for each observed request arrival and the definition of the metric left to the DAs using basic functions such as AVG, MIN, MAX, COUNT, SUM, if available also functions for variance and percentiles computation. On a different implementation, the DC may directly expose as *MonitoringDatum* the total number of request arrival in a time window. To discriminate on the best between these approaches, we will use empirical evaluation, since the answer depends on the specific event and target resource. A list of basic application-level metrics that may be readily computed from the above event list is given in Table 4.2. (The Completion and Arrival events in ResponseTime refer to the same MonitorableResource.)

Table 4.2 Application-Level Metrics (T = observation period)

Metric	**Definition**	**Relevance to MODAClouds**	**Priority**
ArrivalRate	COUNT Arrival / T	Characterize user workload.	Must Have
Throughput	COUNT Completion / T	Characterize application QoS.	Must Have
FailureRate	COUNT Failure / T	Characterize application QoS.	Must Have
ErrorRate	COUNT Error / T	Characterize application functional behaviour.	Must Have
ResponseTime	Completion-Arrival	Characterize application QoS.	Must Have
QueueLength	COUNT Arrivals – COUNT Completion	Characterize application QoS.	Must Have
WaitingQueue	COUNT Arrivals – COUNT Admitted	Characterize application QoS.	Could Have
Utilization	SUM Busy / T	Characterize application scalability.	Must Have

The collected/computed metrics will be used in order to get insights to the system (e.g., overloading conditions, SLA violations, functional misbehaviours and etc.), provide feedback to the design time and also come up with runtime adaptive decisions (e.g., to scale up the virtual resources if overloaded, to do load balancing in the case of long queuing/waiting and etc.). The customer, provider, or intended third parties could evaluate the retrieved metrics whether they meet/exceed/fall below defined thresholds.

4.2 Container-Level Metrics

Application-level monitoring deals also with providing monitoring data regarding the software platform on which the application runs. For example, a Java-based application will run inside a JVM with garbage collection algorithms that will affect its performance. Similarly, a web application served by Apache will be constrained by the threading pool performance. In these initial phases of the project, we have focused on the identification of container-level metrics that could characterize the JVM for applications deployed on IaaS clouds. The MXBeans offered by the Java Management Extension interface appears to be appropriate to this scope. Additional metrics will be included during the development of the *Monitoring Platform* depending on needs.

Table 4.3 JVM Container-Level Metrics

Metric	**MXBean**	**Definition**	**Relevance to MODAClouds**	**Priority**
NonHeapMemoryUsage	MemoryMXBean	Current memory usage of non-heap memory that is used by the Java virtual machine.	Establish JVM memory bottlenecks.	Should Have
HeapMemoryUsage	MemoryMXBean	Current memory usage of the heap that is used for object allocation	Establish JVM memory bottlenecks.	Should Have
PeakUsage	MemoryPoolMXBean	Peak memory usage of a memory pool since the JVM started or since the peak was reset.	Establish JVM memory bottlenecks.	Must Have
Usage	MemoryPoolMXBean	Estimate of the memory usage of a memory pool.	Establish JVM memory bottlenecks.	Must Have
Uptime	RuntimeMXBean	Uptime of the Java virtual machine in millisecond	Check JVM availability.	Could Have
PeakThreadCount	ThreadMXBean	Peak live thread count since the JVM started or peak was reset.	Establish JVM QoS.	Must Have

4.3 Data Collection Specifications

In this section we provide a description of the major tools or techniques which may be used in order to retrieve monitoring information at application level.

Aspect Oriented Programming

Aspect Oriented Programming (AOP) is one of the possibilities that can be used for monitoring of cloud computing services, especially for PaaS level. With AOP, modularity is improved by keeping separate the orthogonal functionality to the main modularization direction, such as logging persistence, and failure handling. Aspects are defined by different characteristics such as join points, pointcuts, and advices. A join point is a point identifying a point during the execution of a program (for example the execution of a method). Join points are specified by pointcuts which are defined as predicates on source code expressions, while advices contains the behavior of the aspect for the associated point cut. There are different AOP frameworks; among them we can name AspectJ, Spring AOP, and JBoss AOP. In MODAClouds, we use aspects as part of our data collector. For example considering servlets in Java EE environment, aspects could provide us the throughput of each servlet, while none of the servlet knows about the existence of them. At the same time we can also do monitoring the behavior of aspects and their pointcuts in run time, in the way of active or inactive them. In this case they produce the data when asked otherwise they are inactive. When all the required data in are ready, they are used to feed the C-SPARQL engine, for further analysis. Here is an example that shows how an aspect calculates the response time of all the servlet that are calling in the package of "Controllers" when the monitoring is enabled.

```
public aspect aspectProfile {
        pointcut startResponses() : execution(* Controllers.*.*(..));
        pointcut endresponses() : if (MonitorAspects.enable == true) && execution(*
Controllers.*.*(..));

        before() : startResponses() {
                start = System.currentTimeMillis();
        }

        after() returning() : endresponses(){
                end = System.currentTimeMillis();
                long elapse = System.currentTimeMillis() - start;
                System.out.println("elapsed time:" + elapse );
}
```

Figure 5: Example response time monitoring of a servlet using AspectJ

SpringSource Hyperic

SprigeSource Hyperic [Hyp13] is a monitoring tool for managing various aspects of web applications. In particular, it provides support for real-time monitoring and, by default, visibility into availability, performance, utilization, and throughput. Hyperic is structured as a set of installable plugins, each one providing various metrics for a single application type. Hyperic supports the most popular application servers (e.g., JBoss, Apache Tomcat, IBM WebSphere, Microsoft .NET, etc.), mail servers (e.g., sendmail, postfix, Microsoft Exchange, etc.) and messaging middleware (e.g., RabbitMQ, ActiveMQ, Hadoop, etc.).

JVM agents

Implemented using JVM Tool Interface (JVM TI) [JVM13], a native programming interface is used to inspect the state and to control the execution of applications running in the Java virtual machine (JVM). A JVM agent runs in the same process with and communicates directly with the virtual machine executing the application being monitored. JVM TI supports the tools that need access to JVM state, including monitoring. A client of JVM TI (i.e., agent), can be notified of interesting occurrences through events. JVM TI can query and control the application through functions, either in response to events or independently. The agents (written C, C compatible and C++) run in the same process and communicate directly through a native interface (JVM TI) with the virtual machine executing the application being examined.

Interceptors (natural probes)

Interceptors are responsible for message processing with the purpose of extracting usage habits from service requests and responses (e.g., Tomcat filters, AXIS handlers). They are independent from the system both at coding and runtime. For instance, by profiling applications with JProfiler [JFP13] it is possible to dynamically analyze the application and measure its memory, CPU and time usage. It also provides a GUI for VM load and activity analysis. Profiling could be carried out by instrumenting either the program source code or its binary executable form using a profiler. The Apache Axis handler processing SOAP messages can also be used as the monitoring mechanism.

JMX interface
JMX [JMX13] is a resource management standard which could be used for building web-based and dynamic solutions for managing and monitoring applications implemented by many kinds of middleware, such as Tomcat, JBoss. The resource to be monitored could be instrumented by objects, called Managed Beans (MBeans), registered in an object server (MBean server), that manages them. The MBean server and the services for handling the MBeans are called the JMX agent altogether. Connectors enable the access to these agents from remote management applications. JMX connectors could use different protocols and the management application can manage resources regardless of that protocol. Moreover JMX provides built in instrumentations in the Java virtual machine (JVM), enabling monitoring through JMX. JVM includes a platform MBean server and MXBeans to be used by management applications. JVM functionalities are encapsulated by platform MXBeans that can be interacted with by a JMX compliant monitoring tool (e.g., JConsole), hence enabling monitoring JVM functionalities. The platform MXBeans are registered in the platform MBean server.

Instrumentation service probes
This category refers to various code injection tools which could be differentiated based on their level of abstraction (e.g., ASM works at bytecode level, Javassist [JAS13] works at bytecode and source level or advices in Java in which the code to be injected could be expressed as syntax-checked and statically compiled Java such as AspectJ), the code injection stage (build time, load time and runtime) and functionalities (e.g., add code before/after methods, add new methods, add/remove methods, modify body of a method, etc.). Dynamic Java Proxy, the bytecode manipulation library ASM, JBoss Javassist, AspectJ, Spring AOP/proxies and Java EE interceptors are examples of well-known code injection tools which could be used in the monitoring data collection phase. Javassist is a class library in order to define (/modify) a class at runtime, providing source level and bytecode level of APIs.

Kieker
Kieker [KIE13] is an extensible framework offering monitoring, analysis and visualization of the behaviour of a system at runtime. In order to use Kieker, source code must be instrumented with *monitoring probes*. Kieker supports Java-based systems although recently adapters for other type of systems have been added (i.e., .NET and COM). Kieker already comes with a set of predefined monitoring probes for the major metrics (i.e., response times, user sessions, CPU utilization, memory usage, etc.). Kieker moreover supports the Palladio component model [BEC09] thus it takes advantage of the various analysis tools already available for this platform.
Its overhead has been shown to be minimal and it has been successfully used in production environments [KIE13]. Kieker is published as an open source project with contributions from both academy and industrial partners.

4.4 Application-level monitoring plan

The strategy concerning the implementation of the application-level monitoring aims at maximizing the re-utilization of existing monitoring tools (the ones defined in Section 4.2). In particular, thanks to the great generality of the monitoring architecture, it is easy to install new monitoring tools (i.e., the change is confined within the data collectors) without changing the existing architecture (i.e., other components are unaware of the specific monitoring tool adopted). It is important to point out that the specific monitoring tool adopted greatly depends on the case study to be implemented. For example if the case study is developed as a Java EE application, then approaches like AOP, JMX and Java agents represents a solid choice for the application level monitoring.

5. Data Analysis

5.1 Detection DAs

When analysing online data for QoS, the *Monitoring Platform* must be able to determine if the system is not currently complying with SLAs. This can be due to, for instance, insufficient capacity, a malfunctioning host, or a sudden change in the workload mix. The automatic detection of such violations becomes critical in large deployments, where manual detection can be too costly or infeasible [CHE09]. To this end, the detection DAs will perform window-based estimations to determine whether a certain metric is within the application SLAs. The analysis will be done on a time window, as punctual observations can be affected by several sources of noise, while a set of observations collected during a time window can absorb this noise and provide more accurate values for diagnosis. The detection DAs may range from basic threshold-based queries to identify QoS violations, to outlier-detection algorithms for performance anomaly detection. Detection of QoS constraint violations will trigger alarms to the Self-Adaptive Platform and may be further passed to the Correlation DAs in order to establish the possible causes for the anomaly.

Detection DAs work in the *Monitoring Platform* in the context of an ontology that models the concepts defined in Figure 2.1. This ontology has to be interpreted as the core of a modular network of ontologies that extends the core one depending on the specific characteristics of the cloud application to be monitored. When deploying a *Monitoring Platform*, the responsible for the deployment has to specify the concrete monitoring rules, metrics, monitorable resources, and control actions.
As an example of extended ontology to be deployed in a specific monitoring system, let's consider a HTTP server h_1, installed on a virtual machine vm_1, which hosts one HTML page p_1. vm_1 is an instance of VM. To type h_1 and p_1, we need to define two classes: HTTPServer and HTMLPage. The former is a subclass of Component, while the latter is a subclass of Operation. As the names suggest they represent respectively the HTTP servers and the HTML pages that can be published on it. p_1 is an instance of HTMLPage and h_1 is an instance of HTTPServer. The relations between the three elements are defined exploiting the 'contains' relation: all the three classes VM, HTMLPage and HTTPServer inherit this property from Monitorable Thing. So vm_1 contains h_1 and h_1 contains p_1. Summarizing, we defined the following RDF model:

```
:HTTPServer rdfs:subclassOf mo:Component
:WebPage rdfs:subclassOf mo:Component
:vm_1 rdf:type mo:VM
:h_1 rdf:type :HTTPServer
:p_1 rdf:type :WebPage
:vm_1 mo:contains :h_1
:h_1 mo:provides :p_1
```

The following C-SPARQL query models a monitoring rule that produces a new data on the output stream when the average number of served Web pages for a HTTP server is to below 50 pages per minute. In this case the new data is to be interpreted as an alert of a critical situation.

```
REGISTER QUERY ServedPagesController AS
PREFIX mc: <http://../2013/2/monitoring_0-2#>
CONSTRUCT { [] a mc:Action ; mc:isOn ?httpServer . }
FROM STREAM <http://…/monitoringStream> [RANGE 1m STEP 1m]
FROM <http://../modaclouds/ModaCloudsSK.rdf>
WHERE {
 ?httpserver a mc:HTTPServer ;
                  mc:isIn ?vm .
 ?vm mc:exposes ?dc .
 ?dc mc:observes ?md .
 ?md a mc:NumberOfServedPages ;
       mc:isAbout ? httpserver ;
       mc:hasValue ?v .
}
GROUP BY ?httpServer
HAVING(avg(?v) < 50)
```

The query is executed as follows: first, all HTTPServer Component (?httpserver) are extracted from the current tumbling time window of 1 minute over the stream, then using the mc:IsOn property we extract the Virtual Machine (?vm) that contains ?httpServer, ?vm mc:exposes a Data Collector ?dc that observes a specific Monitoring Datum (?md), in this case we are interested in NumberOfServedPages. We check if ?md is related to the extracted component ?httpServer and extract its value, finally we group the result by ?httpServer and check if the average NumberOfServedPages is less than 50 per minutes.

5.2 Correlation DAs

The detection of an SLA being violated must be complemented by an analysis of the possible causes of the anomaly. In this manner it should be possible to differentiate two broad types of anomalies: exogenous causes associated to a change in the workload, which are out of control of the application, and those associated to an internal problem, e.g., limited scalability of software or hardware components, errors, failures. Such a distinction is needed because a change in the workload can alter significantly the performance of an application, indicating that more resources are needed, so this would not be considered as a failure and care should be taken on when to trigger an alarm [CHE09].

Although SLAs will typically be set for the application as a whole (e.g. application response time), in order to take corrective actions, it is necessary to pinpoint the root cause of a QoS constraint violation. To this end a correlation analysis performed, such that a change in a certain metric in a given VM can be selected as the potential cause of the SLA violation. Furthermore, in absence of QoS violations, statistical correlations may still be characterized on the flowing data to learn the application behaviour at runtime and parameterise the predictive models used by the Self-Adaptive Platform reasoners. In this context, we envision as fundamental the input of white-box models defined at runtime to describe the application functional behaviour and its QoS, the work that the MODAClouds project will undertake in WP4 and WP5. For example, by knowing a priori that a given request type cannot hit the database, it is possible to exclude this request arrival rate as a cause of overloading at the database tier. The Correlation DAs will perform these analyses, and deliver the results to the *MODAClouds IDE* and the Self-Adaptive Platform, which will then take the appropriate actions, if necessary.

5.3 Forecasting DAs

In addition to detect an anomaly and determine its root cause, Forecasting DAs will forecast the value of some metrics registered on the *Monitoring Platform*. The aim of this is to give the opportunity to the Self-Adaptive Platform to pro-actively reconfigure the deployment when a trend in the parameters is identified. For instance, a forecasted decrease in the workload may be used by the self-adaptive platform to reduce the number of instances or VMs required by the application for the planning interval.

To this end the forecaster will rely on time-series tools, which are well suited to describe the relation among a stream of samples for a given parameter, and can therefore provide estimates on its future values. Autoregressive models, such as AR, ARMA or ARIMA models, and regression methods are good candidates to perform this analysis [JIA06].

5.4 Estimation DAs

In order to provide accurate performance and availability estimations, the Self-Adaptive Platform will rely heavily on a set of models that capture the inner structure and dynamics of a cloud application. For the models to be accurate, their parameters must be constantly updated and refined to stay in agreement with the current characteristics of the cloud environment. Furthermore, the estimation of such parameters must be based on the monitoring information, where direct measurements cannot always be taken, and the estimation procedure must rely on indirect measurements, such as request response times. This is a salient issue in PaaS environments, where relevant information, such as CPU utilization, is not available. Other relevant issues include limits on the sampling rate, as well as a significant background noise caused by other applications deployed in a virtualized environment.

The Estimation DAs will provide a set of algorithms to estimate such model parameters, considering different levels of available information. The estimator will focus on demand estimation methods, that enable the characterization of the user service demand, a key ingredient in a QoS model. These methods may rely on information such as the response time, and the outstanding number of requests at a resource, which are available for both PaaS and IaaS environments. Other information available in IaaS deployments only, such as the CPU utilization, will be considered in IaaS-oriented estimation DAs. An important constraint to consider when estimating model parameters in runtime is the need to provide timely results that are valid under the current system conditions. The Estimation DAs will feature different computational requirements in order to cope with different types of deadlines to return the results of their analyses.

6. Technology Stack

6.1 C-SPARQL

In 2008, Della Valle et al. in [DEL09] called the Semantic Web community for techniques able to reason upon rapidly changing information. When reasoning on massive data streams, such as those characterizing the monitoring of cloud base applications, well known artificial intelligence techniques have the right level of expressivity, but their throughput is not high enough to keep pace with the stream (e.g., belief revision [GAE03]). The only technological solutions with the right throughput are Data Stream Management Systems

(DSMS) [GAR07] and Complex Event Processing [LUC08], but, on the other hand, they are not expressive enough. A new type of inference engines is thus needed to reason on streams. [DEL09] named them *stream reasoners*.
In the following years, a number of stream reasoning approaches have been developed [BAR10,ANI11,TDO11,CAL10]. They share three main concepts:

1. they logically model the information flow as an RDF stream, i.e. a sequence of RDF triples annotated with one or more non-decreasing timestamps,
2. they process the RDF streams "on the fly", often by rewriting queries to the raw data streams, and
3. they exploit the temporal order of the streaming data to optimize the computation.

Within the monitoring systems of MODAClouds, we propose to use Continuous SPARQL (C-SPARQL) [BAR10] – an extension of SPARQL that continuously processes RDF streams observed through windows (as done in DSMS). The current version of the C-SPARQL engine[1] is an extension of SPARQL 1.1 and it includes support for all its constructs. The syntax and semantics of C-SPARQL were described in [BAR10b]. The C-SPARQL execution engine and its optimization techniques were illustrated in [BAR10c]. The optimization needed for high-throughput RDFS++[2] reasoning are described in [BAR10d]. Hereafter we provide a minimum guide to RDF streams and C-SPARQL syntax.

RDF Stream Data Type

C-SPARQL adds RDF streams to the data types supported by SPARQL1.1. An RDF stream is defined as an ordered sequence of pairs, where each pair is made of an RDF triple and its timestamp τ:

... (⟨subji, predi, obji⟩ , τi) (⟨subji+1, predi+1, obji+1⟩ , τi+1) …

Timestamps can be considered as annotations of RDF triples; they are monotonically non-decreasing in the stream (τi ≤ τi+1). They are not strictly increasing because timestamps are not required to be unique. Any (unbounded, though finite) number of consecutive triples can have the same timestamp, meaning that they "occur" at the same time, although sequenced in the stream according to some positional order.

Example. In our running example, taken from monitoring system scenario, data streams are associated with http-status. In the streams every triple corresponds to the page-url that return a specific http-status. The predicate of the triple (t:returnedBy) is fixed, while the subject (?httpstatus) and object (?requestedurl) parts of the triple are variable. Thus, a physical source for this stream has items consisting of pairs of values. This arrangement is coherent with RDF repositories whose predicates are taken from a small vocabulary constituting a sort of schema, but C-SPARQL makes no assumption on variable bindings of its stream triples. An example of stream with five cars passing through three different tollgates is given below (for the sake of simplicity in the following figure we are not using the ontology defined in 2.2).

Triple	**Timestamp**
c:403 t:returnedBy "http://www.modaclouds.eu/url1"	t100
c:404 t:returnedBy "http://www.modaclouds.eu/url2"	t101
c:404 t:returnedBy "http://www.modaclouds.eu/url3"	t102
c:403 t:returnedBy "http://www.modaclouds.eu/url4"	t103
c:401 t:returnedBy "http://www.modaclouds.eu/url5"	t104

Data Sources and Time Windows

The introduction of data streams in C-SPARQL requires the ability to identify such data sources and to specify selection criteria over them. As for identification, we assume that each data stream is associated with a distinct IRI, that is a locator of the actual data source of the stream. More specifically, the IRI represents an IP address and a port for accessing streaming data. As for selection, given that streams are intrinsically infinite, we

[1] http://streamreasoning.org/download

[2] RDFS++ is the Description Logic compatible subset of RDFS extended with owl:inverseOf, and owl:transitive.

introduce the notion of windows upon streams, whose types and characteristics are inspired by those defined for relational streaming data.
Identification and selection are expressed in C-SPARQL by means of the FROM STREAM clause. The syntax is as follows:

```
FromStrClause → 'FROM' ['NAMED'] 'STREAM' StreamIRI '[ RANGE' Window ']'
Window    → LogicalWindow | PhysicalWindow
LogicalWindow → Number TimeUnit WindowOverlap
TimeUnit    → 'ms' | 's' | 'm' | 'h' | 'd'
WindowOverlap → 'STEP' Number TimeUnit | 'TUMBLING'
PhysicalWindow → 'TRIPLES' Number
```

A window extracts the last data elements from the stream, which are the only part of the stream to be considered by one execution of the query. The extraction can be physical (a given number of triples) or logical (all triples occurring within a given time interval, whose number is variable over time).
Logical windows are sliding if they are progressively advanced by a given STEP (i.e., a time interval that is shorter than the window's time interval). They are non-overlapping (or TUMBLING) if they are advanced in each iteration by a time interval equal to their length. With tumbling windows every triple of the stream is included exactly into one window, whereas with sliding windows some triples can be included into several windows.
The optional NAMED keyword works exactly like when applied to the standard SPARQL FROM clause for tracking the provenance of triples. It binds the IRI of a stream to a variable which is later accessible through the GRAPH clause.

Example. A monitoring system query counts the number of pages returning http-statuses; the query considers the last 10 minutes, while the sliding window is modified every minute.

```
PREFIX mc: <http://www.modaclouds.eu/monitoring#>
SELECT DISTINCT ?httpstatus (COUNT(?requestedurl) AS ?pagecount)
FROM STREAM <http://stream.org/monitoredhttpstatus.trdf>[RANGE 10 MIN STEP 1 MIN]
WHERE { ?httpstatus mc:returnedBy ?requestedurl . }
GROUP BY ?httpstatus
```

The query is executed as follows: first, all pairs of status and url are extracted from the current window over the stream, then the total number of url for each status is counted into the new variable pagecount and every pair is extended into a triple, and finally the triple is projected as distinct pairs of status and number of url. The window considers all the stream triples in the last 10 minutes, and is advanced every minute. This means that at every new minute new triples enter into the window and old triples exit from the window. Note that the result of the aggregation does not change during the slide interval, therefore also the query result does not change during the slide interval; it changes instead at every slide change.

C-SPARQL Queries

All queries over RDF data streams are denoted as continuous queries, as they continuously produce output in the form of tables of variable bindings tables or RDF graphs. Each C-SPARQL query is registered through the following statement:

Registration → 'REGISTER QUERY' QueryName ['COMPUTED EVERY' Number TimeUnit] 'AS' Query

Only queries in the CONSTRUCT and DESCRIBE form can be registered as generators of RDF streams, as they produce RDF triples, associated with a timestamp as an effect of the query execution. The optional COMPUTED EVERY clause indicates the frequency at which the query should be computed. If no frequency is specified, the query is computed at a frequency that is automatically determined by the system.

Example. Assume that a (classic, static) RDF repository stores (a) the page hosted in a http-server, (b) the URL of each page, and (c) the virtual machine containing http-server. We now show a query that combines static knowledge (from the repository) and dynamic knowledge (from the streaming data) in order to periodically count how many http-statuses have returned from each VM in the last 30 minutes. In this example the window is sliding with a step of five minutes. From now on, the c: and t: prefixes are omitted for brevity.

```
REGISTER QUERY httpstatuspervm COMPUTED EVERY 5 MIN AS
SELECT DISTINCT ?vm (COUNT(?httpstatus) AS ?httpstatuscount)
FROM STREAM <http://stream.org/monitoredhttpstatus.trdf> [RANGE 30 MIN STEP 5 MIN]
FROM <http://www.modacloud.eu/monitoringsystemmap.rdf>
WHERE { ?httpStatus mc:returnedBy ?url .
?page mc:hasurl ?url .
?page mc:IsIn ?httpserver .
?vm mc:contains ?httpserver .
}
```

The query is executed as follows. As in the previous query, all pairs of bindings of http-status and url are extracted from the current window over the stream, and joined to a graph pattern used to extract from the RDF repository the pair of bindings of URL with their pages, the pages with their http-server and, finally, the http-server with their VM. Then, the number of http-statuses returned by the page in each VM is counted into the new variable passages. Finally, pairs of distinct VM and http-status count are projected.

Stream Registration

C-SPARQL allows the production of RDF streams, registered through the following statement:

Registration → 'REGISTER STREAM' QueryName ['COMPUTED EVERY' Number TimeUnit] 'AS' Query

Only CONSTRUCT and DESCRIBE queries can be registered as RDF streams, as they produce RDF triples that are associated with a timestamp as effect of the query execution. Every query execution produces from a minimum of one triple to a maximum of an entire RDF graph, depending on the query construction pattern. In the former case, a different timestamp is assigned to every triple. In the latter case, the same timestamp is assigned to all the triples of the constructed graph. Still, the system-generated timestamps are in monotonic non-decreasing order.

Example. The following example allows the construction of new RDF data streams, by registering CONSTRUCT queries. Consider the previous query in which the projected pairs of districts and passages are used to construct a triple. The corresponding C-SPARQL query is:

```
REGISTER STREAM httpstatuspervm COMPUTED EVERY 5 MIN AS
CONSTRUCT {?vm mc:returned ?httpstatuscount}
FROM STREAM <http://stream.org/citytollgates.trdf> [RANGE 30 MIN STEP 5 MIN] WHERE
{
   SELECT ?vm (COUNT(?httpstatus) AS ?httpstatuscount)
   WHERE {
          ?httpStatus mc:returnedBy ?url
          ?httpStatus a mc:clientErrors
          ?page mc:hasurl ?url
          ?page mc:IsIn ?httpserver
          ?vm mc:contains ?httpserver
         }
}
```

This query uses the same logical conditions as the previous one, but using the inference, extracts only the http-statuses of the client-error class (and its subclasses e.g. 400, 401, 402, 403, 404 etc.) and constructs the output in the format of a stream of RDF triples. Every query execution may produce from a minimum of one triple to a maximum of an entire graph. In the former case, a different timestamp is assigned to every triple. In the latter case, the same timestamp is assigned to all the triples of the graph. In both cases, timestamps are system-generated in monotonic order. Results of two evaluations of the previous query are presented in the table below.

triple	**Timestamp**
c:303 t:returnedBy "http://www.modaclouds.eu/url1"	t400
c:404 t:returnedBy "http://www.modaclouds.eu/url2"	t400
c:403 t:returnedBy "http://www.modaclouds.eu/url3"	t401
c:404 t:returnedBy "http://www.modaclouds.eu/url4"	t401
c:301 t:returnedBy "http://www.modaclouds.eu/url5"	t401

The first evaluation occurs at t400. Suppose that only data from two sources (i.e., c:303 and c:404) are present in the window. Then, the evaluation generates two triples with the same timestamp (i.e., t400).
The second evaluation occurs at t401. Suppose that part of the data elaborated by the previous query are still in the window and that new data related to uc:403 entered in the window. Then, the evaluation produces 3 triples; all of them have the same new timestamp (i.e., t401).

6.2 MATLAB Compiler Runtime

Estimation and forecasting will require advanced time series analysis and stochastic modelling tools. Algorithms and models to attack these problems have been reviewed in deliverable D6.1 and include, among others, maximum likelihood, ARMA processes, clustering, neural networks, optimization methods, and regression methods. In order to use these models as part of the DAs, we plan to use the MATLAB Compiler Runtime (MCR), which is a royalty-free container that allows the standalone execution of compiled scripts developed in MATLAB on machines of users without MATLAB licence. The development of many of the tools will therefore be done in the MATLAB and their deployment will be performed through the MCR. Since MATLAB is capable of running Java code natively, we could seamlessly blend standard Java code to consume services and streams with this numerical environment. One unknown at this stage is the ability of the MCR to operate operation fully in memory, an important feature to use for performance reasons. In case this turns out to be problematic, we may evaluate other numerical computation engines, such as GNU OCTAVE or GNU R.

6.3 Monitoring Data Archival and Graphing

In the context of MODAClouds, the monitoring data coming from either the application, its resources, and possibly the support services, are used mainly to support QoS monitoring, SLA monitoring and enforcement. Thus most of the employed algorithms and techniques use only "sufficiently recent" data --- for example a sliding window, or a ring buffer, fed from the monitoring data stream --- thus the value of "historical" monitoring data is less important. Moreover the from a technical point of view, the handling of monitoring data seems to move in the following direction: once the readings are collected they are quickly transferred to a C-SPARQL instance, which is the single data source to be used by all the tools requiring access to monitoring data. However there might be use cases where the operator of an application might want to store, and possibly access on-line, monitoring data spanning over a large period of time, for example more than a week, which in turn would require a technical solution adapted for such a situation.

Fortunately there are a few out-of-the box solutions built specifically for such a purpose, which we shall briefly describe in what follows. Thus, if during the course of the project we deem as necessary to provide monitoring data archival, we have out our disposal a few potential candidates that could be quickly integrated in the MODAClouds ecosystem. In particular, these solutions may be versatile for integration in a monitoring dashboard or for selective graphing inside the *MODAClouds IDE* of the runtime monitoring data.

OpenTSDB (http://opentsdb.net/)
Initially built at StumbleUpon, and released as open source, it is one of the most mature solutions, capable of handling huge amounts of data (as in TBi). OpenTSDB exploits features offered by Hadoop's HBase, possibly deployed on top of Hadoop's HDFS or another distributed file system. OpenTSDB provides both data storage, data access, and minimal graphing support.

Graphite (http://graphite.wikidot.com/)
Starting as a replacement of the venerable RRDtool, it aimed at being semantically close to it, but technically more efficient, by allowing fast writes of many metrics from many sources (in the thousands), which with earlier versions of RRDtool was sub-optimal. However as with RRDtool the original data is not retained past a particular point in time, but aggregated into increasingly coarse grained statistics. On the upside it also provides a useful dashboard-like interface, allowing the operator to browse through various graphs of the available metrics.

Cacti
This open source network graphing tool is the most well-known front end to a RRDTool and therefore uses standard MySQL to store and retrieve data. RRDtool is a high performance data logging and graphing system for time series data. Cacti's versatility comes with the ability to use external data gathering scripts as well as direct data sources. It also features a hierarchical user rights tree for management as well as a template for ease of

scaling graphs and data sources. By using Cacti, statistics of the chosen metrics over a time horizon can be logged without additional effort.

Cube (http://square.github.com/cube/)
Cube is another recent open-source project, which provides support for storage and query of monitoring data, by leveraging another NoSQL database, namely MongoDB. However it is still a young project, not deployed nearly as much as the previous solutions.

Generic time series databases
Another option would be to turn towards time-series databases (like for example IBM's Informix), which were specifically built for such purposes. Unfortunately, to our knowledge, there are no production ready open-source solutions, thus it would prove impractical for our project to integrate such a solution.

Generic SQL, NoSQL, or other embedded databases
If everything else fails, we can fallback to general purpose solutions, but we would need to provide a wrapping service, and the scalability would be highly impacted. However embedded databases like Berkeley DB (Oracle), or LevelDB (Google) might prove the best candidates to start with, as would provide the greatest flexibility, although they aren't themselves distributed. Regarding relational databases, we would expect that they would perform much more poorly due to their inherent features, such as transactions or constraints, which are not strictly required for our use case.

7. Implementation and validation plans

7.1 Implementation Plan

The implementation plan is organized according to two principal deadlines. The first deadline is scheduled for month 12 where a first version of the MODAClouds prototype will be produced.
The second deadline is scheduled six months after the previous one (month 18) and will include a fully working implementation of the MODAClouds middleware. In both deadlines the prototype will cover both IaaS and PaaS cloud models. However, in order to factor into the implementation the case study requirements when they are in more advanced release status, we will initially focus more on infrastructure-level metrics (IaaS) and application-level metrics (PaaS/IaaS) will be emphasized more in the second release of the *Monitoring Platform*.

7.2 Validation Plan

The validation plan is structured in two phases. In the first phase, early test proving the feasibility of the approach will be implemented. Since the early versions of the prototype might not be complete enough to allow an evaluation, initial tests will be performed on a simulation-based environment.
The second phase of the validation phase will focus on two case study applications: *MiC*, a simple social networking application with user recommendation capabilities primarily developed for PaaS, and *OFBiz*, a web application that runs on IaaS which we will stress with the OFBench framework. Details of these applications are provided below. Our validation will consist in extracting the infrastructure-level metrics from the OFBiz use case and the application-level metrics from both use cases. In this way we will be able to demonstrate the ability of the framework to cope with both IaaS and PaaS application. Since the first year of the project will focus mostly on IaaS, as explained in the deliverable D6.1 roadmap, we intend to initially focus on OFBiz and then extend the *Monitoring Platform* to include PaaS applications like MiC.

MiC – Meeting in the Cloud

MiC is a basic social networking application. Registered users fill a questionnaire expressing interest on specific topics. Then, users' similarity is computed. Finally, when a users access the system the messages posted on specific topics by its top-most three similar users are shown in his page. Users can also ask to update their similarity time by time. The workload is generated by jMeter specifying the concurrent number of users and their think time. The number of users can be changed during the test exploiting jMeter plugins. The application has been developed as a set of JSPs and a Java servlet for the similarity computation. JPA, AJAX. The MiC application can be deployed on Azure or Google App Engine (GAE). The application uses a wide set of cloud services, in particular:

- Blob service: for storing each user's profile picture
- SQL service: stores users'profile and best contacts

- NoSQL service: data access is performed through JPA. The NoSQL database stores data on user's topic preferences
- memcache: stores topics and best-contacts of the user for each topic
- Task queue: used to decouple the front-end and the application logic that computes the users similarity

The application can be deployed on a single VM (i.e. Azure worker role or GAE front-end VM), or on two VMs. If two VMs are adopted, the first runs the front-end and the second the user similarity servlet. In the Google deployment this is achieved by deploying the applications with two different versions. The deployment is fully automatic by means of an Eclipse plug-in.

Apache OFBiz

OFBench is realistic research benchmark for the demo e-commerce store distributed with the Apache Open For Business (OFBiz) ERP framework. OFBiz is a framework deployed in production environments, hence our benchmark allows to experiment with a software architecture and a technology stack that are representative of real business applications. For these applications, OFBench can stress a range of performance indicators such as response times, CPU utilization, memory consumption, number of active sessions over time, multi-threading levels, data caching. Real web browsers are used to drive user emulation, by means of the Selenium automation framework for Mozilla Firefox. Compared to an HTTP replayer, this approach allows for more realistic stress testing of the presentation tier of the application, for example by including in the response times also the browser rendering time components.

7.3 Conclusion

In this deliverable, we have provided specifications for the MODAClouds *Monitoring Platform*. The problem of monitoring application in IaaS and PaaS clouds is complex because the metrics that can be collected in these two environments are different. MODAClouds will tackle this approach by providing a set of application metrics that can be consistently gathered on all target clouds. In addition, to leverage the specificity of IaaS providers, infrastructure-level metrics, such as resource utilization, will be gathered when possible. The data will be acquired by data collectors and passed via the C-SPARQL stream engine to data analysers that will extract metrics of interest for the observers of the monitoring data. In particular, the Self-Adaptive Platform will consume this data to adapt application QoS at runtime. Implementation will focus initially on infrastructure-level metrics and, after the case study requirements are defined, we will refine also application-level metrics and start their implementation. At this stage, we plan to use the MiC and OFbiz applications as test applications to validate the effectiveness of the *Monitoring Platform* proposed in this deliverable.

8. References

[ANI11] D. Anicic, P. Fodor, S. Rudolph, N. Stojanovic, EP-SPARQL: a unified language for event processing and stream reasoning, in: S. Srinivasan, K. Ramamritham, A. Kumar, M. P. Ravindra, E. Bertino, R. Kumar (Eds.), WWW, ACM, 2011, pp. 635–644.
[BAR10] D.F. Barbieri, D. Braga, S.Ceri, E. Della Valle, M. Grossniklaus, Querying RDF streams with C-SPARQL, SIGMOD Record 39 (1) (2010) 20–26.
[BAR10b] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, M. Grossniklaus, C-SPARQL a continuous query language for rdf data streams, Int. J. Semantic Computing 4 (1) (2010) 3–25.
[BAR10c] D.F. Barbieri, D. Braga, S. Ceri, M. Grossniklaus, An execution environment for C-SPARQL queries, in: I. Manolescu, S. Spaccapietra, J. Teubner, M. Kitsuregawa, A. Léger, F. Naumann, A. Ailamaki, F. Özcan (Eds.), EDBT, Vol. 426 of ACM International Conference Proceeding Series, ACM, 2010, pp. 441–452.
[BAR10d] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, M. Grossniklaus, Incremental Reasoning on Streams and Rich Background Knowledge, in: Proc. of ESWC2010, 2010.
[Bec09] Becker S., Koziolek H., Reussner R., The Palladio component model for model-driven performance prediction, Journal of Systems and Software, Volume 82, Issue 1, January 2009, Pages 3-22, ISSN.
[CAL10] J.-P. Calbimonte, Ó. Corcho, A.J.G. Gray, Enabling ontology-based access to streaming data sources, in: P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Z. Pan, I. Horrocks, B. Glimm (Eds.), International Semantic Web Conference (1), Vol. 6496 of Lecture Notes in Computer Science, Springer, 2010, pp. 96–111.
[CHE09] Cherkasova, L, Ozonat, K, Mi, N, Symons, J, Smirni, E. Automated anomaly detection and performance modeling of enterprise applications. ACM Trans. Comput. Syst. 2009.
[COL13] http://collectl.sourceforge.net/ Last accessed March 26, 2013.
[Dee11] D. Jayasinghe, S. Malkowski, Q. Wang, J. Li, P. Xiong, C. Pu. Variations in Performance and Scalability When Migrating n-Tier Applications to Different Clouds. Proc. of IEEE Cloud 2011, 73-80.

[DEL09] E. Della Valle, S. Ceri, F. van Harmelen, D. Fensel, It's a Streaming World! Reasoning upon Rapidly Changing Information, IEEE Intelligent Systems 24 (6) (2009) 83–89.
[FID88] Fidge, C. J. Timestamps in Message-Passing Systems That Preserve the Partial Ordering. In K. Raymond (Ed.). Proc. of the 11th Australian Computer Science Conference (ACSC'88). pp. 56–66. 1988.
[GAE03] P. Gaerdenfors (Ed.), Belief Revision, Cambridge University Press, 2003.
[GAR-7] M. Garofalakis, J. Gehrke, R. Rastogi, Data Stream Management: Processing High-Speed Data Streams, Springer-Verlag New York, Inc., 2007.
[GUO06] Guo, Z, Jiang, G, Chen, H, Yoshihira, K. Tracking Probabilistic Correlation of Monitoring Data for Fault Detection in Complex Systems. Proceedings of the International Conference on Dependable Systems and Networks (DSN '06). 2006.
[HOL09] Holub, V, Parsons, T, O'Sullivan, P, Murphy, J. Run-time correlation engine for system monitoring and testing. Proceedings of the 6th international conference on Autonomic computing (ICAC '09). 2009.
[Hyp13] http://www.hyperic.com/ Last accessed March 21, 2013
[JAS13] http://www.csg.is.titech.ac.jp/~chiba/javassist/ Last accessed March 21, 2013
[JIA06] Jiang, J, Chen, H, Yoshihira, K. Discovering likely invariants of distributed transaction systems for autonomic system management. Cluster Computing. 2006.
[JMX13] http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html . Last accessed March 21, 2013
[JPR13] http://www.ej-technologies.com/products/jprofiler/overview.html . Last accessed March 21, 2013
[JVM13] http://docs.oracle.com/javase/6/docs/platform/jvmti/jvmti.html . Last accessed March 21, 2013
[KIE13] Van Hoorn, A., Waller, J. and Hasselbring, W. (2012) Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis In: 3rd joint ACM/SPEC International Conference on Performance Engineering (ICPE'12).
[LAM78] Lamport, L. Time, clocks, and the ordering of events in a distributed system. Comm. ACM. 1978.
[LUC08] D.Luckham,The power of events: An introduction to complex event processing in distributed enterprise systems, in: N. Bassiliades, G. Governatori, A. Paschke (Eds.), Rule Representation, Interchange and Reasoning on the Web, Vol. 5321 of Lecture Notes in Computer Science, Springer Berlin, 2008, pp. 3–3.
[TDO11] T. Do, S. Loke, F. Liu, Answer set programming for stream reasoning, in: C. Butz, P. Lingras (Eds.), Advances in Artificial Intelligence, Vol. 6657 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2011, pp. 104–109.