Agent Based Services for Negotiation, Monitoring and Reconfiguration of Cloud Resources

Salvatore Venticinque, Second University of Naples, Italy

http://www.mosaic-cloud.eu
Agenda

- Motivation
- Cloud Agency Vision
- Requirements and design
- Architecture
- Agent Based Services
  - Negotiation
  - Monitoring and Reconfiguration
Cloud as a computer market

Cloud business model: *Pay per use*

1. What buy?
2. Where buy?
3. How ask?
4. How get?
5. How check that you get what you buy?
Ask, buy and use?

Define requirements

broker a SLA agreement

Use cloud services

Resources, QoS parameters and patterns

Cloud agency

mOSAIC APIs
Objectives

The mOSAIC team plans to investigate Cloud-specific SLAs and QoS requirements.

- **Definition of QoS parameters:** time, cost, computer power, storage size, reliability, trust, security, or even location of resources due to business constraints.

- **SLA negotiation.** SLA negotiation with multiple Cloud providers by delegation to a third party, represented by a broker in a market based context.

- **Re-negotiation:** it can solve some inconsistencies between the SLA and the real user’s requirements which can change dynamically.

- **Monitoring:** of Cloud utilization can be delegated to a trusted third party 1) by providers who want maximize utilization of their resources in order to optimize profit, without violate the agreements; 2) by user, who has conflicting interests with providers.
Cloud resources

- **Applications**
  - Amazon
  - Facebook
  - Google
  - Salesforce

- **Platforms**
  - Microsoft Azure
  - Google App Engine
  - Rackspace

- **Infrastructures**
  - OpenNebula
  - Amazon Web Services

**What buy?**
Definition of QoS parameters

One of the preliminary requirements, which are relevant to support negotiation activities into the Cloud, is the definition of QoS parameters for existing service.

QoS parameters are necessary to:

- fill the services request in order to negotiate the Cloud resource
- describe the Cloud offer
- match the compliant services and build the best available solution
- define the SLA
- monitor the service levels

QoS parameter will be part of a common *Cloud ontology* to support interoperability and advanced semantic services.
The starting point will be Cloud taxonomy. The ontology will describe Cloud resources and services, Cloud actors, Quality of Service Parameters, the negotiation protocol and the SLA.

- Support of interoperability
- Semantic service discovery
- Mapping of different native ontologies
SLA negotiation

SLA negotiation with multiple Cloud providers will be delegated to a broker in a market based context that provide for:

- searching for available Cloud services, compliant with user needs;
- checking trustiness of providers;
- deciding with whom to negotiate, according to user requirements and past experiences;
- negotiating the best price for the same offer by different providers;
- negotiating of multiple SLAs, with different providers, to overcome the lack of one compliant offer by a single provider.
Brokering

- mOSAIC Agents and providers will interact to broker optimal resources
- FIPA interaction protocols will be extended
- SLA agreement will store results

Composition of different offers driven by:
- patterns
- resources
- QoS parameters
- all described in an Ontology
SLA agreement

- Digital signature on both sides
- Advanced model to support re-negotiation

How to get?

- Used
  - By client to ...
  - By the agency to ...
  - By the server to ...

---

**Involved Parties:**
- IDs and Interfaces of Signatory Parties
- IDs and Interfaces of Supporting Parties

**Service Characteristics & Parameters:**
- Operations offered by Service
- Transport encoding for Messages
- Agreed-upon SLA Parameters (Output)
- Metrics used as Input
- How/where to access Input Metrics
- Measurement Algorithm
- Measurement Duration, Sampling Rate

**Guarantees & Constraints:**
- When is SLA Parameter guaranteed?
- How to detect Violation (Formula)
- Corrective Actions to be carried out

---

SLA
- protect your business with the right service level
Providers monitor utilization of their resources for billing, to change bid prices in order to optimize profit, and to not exceed in resource allocation beyond the capability of respect the agreements.

Users have conflicting interests with providers, needs to trust a third party that can be delegated to monitor the satisfaction of the agreed service levels.

Monitoring process should provide information about:
● under-utilization of cloud resources, in order to negotiate cheaper agreements;
● saturation of resources, to not let the users' applications work under the QoS level granted to users’ clients;
● unbalanced utilization of Cloud resources, in order to check the correctness of negotiated parameters, or to tune the execution of applications in the Cloud;
● violation of SLA by providers.
SLA re-negotiation

Re-negotiation is another service that can be provided to solve some inconsistencies between the agreed SLA and the real user’s requirements which can change dynamically.

Dynamic SLA re-negotiation has actually limited support. Issues to be investigated are:

- withdraw of a SLA and negotiation of a second one;
- deletion of a SLA objective;
- addition of a SLA objective;
- redefinition of SLA parameter;
- negotiation of boundaries within which the SLA can be re-negotiated at the same price or with a pre-defined price adjustment.
A Cloud Agency

• **T1.4 Cloud agency design** is the task whose objective is the development of a Cloud Agency

• The mOSAIC Cloud agency will be conceived according a service-oriented architecture, where agents will implement state-full, eventually mobile, services

• Negotiation, monitoring, dynamic benchmarking and reconfiguration of cloud resources are some mandatory services to be implemented

Agent Services act on users' behalf
MAGDA

Mobile Agent based Grid Architecture

MAgDA is a prototypal toolset conceived to provide the programmer with facilities which could ease the development of Agent based Grid applications

• It is conceived according the layered model of the Grid (Y. Forest)
• It is based on the exploitation of mobility
• It extends Agents API to support distributed computing.
Overview

Application
- OMP integration
- Parallel Skeletons
- Collective communication primitives

Collective
- Service Discovery
- Load Balancing
- Server clustering
- Reliable protocols

Resources
- System Monitor
- Security policies
- Access Control and agent authentication

Connectivity
- Agent’s protocol (messaging, dispatching, ...)
- SIP based localization protocols
- Authorization protocols
- Remote access

Fabric
- JVM & platform dependent natives modules
Grid like SOA

X

Grid Computing

X

Grid Services
Grid and Web Services Standards

Technological Alignment to build Business and Computational services
Services Provisioning

http/html
ACL over http
SOAP over http

End-Point RPC
Tomcat

Agent Server

Access Manager
Authentication

Service 1

Service 2
MAgDA Interface

- API design will extend OCCI RESTful HTTP Rendering
- An agent proxy will translate this requests into FIPA ACL messages for Cloud Agency internal communication
High level architecture: server side
Developing cycle

In order to develop and deploy an agent service as a Grid service, programmers need to:

1) define the service interface;
2) generate necessary code to forward SOAP requests as ACL messages to target agent provider;
3) specialize construction of ACL requests;
4) specialize handling of ACL responses;
5) deploy the solution as a new Grid service into the Globus container;
6) provide users with stubs of the new service (this is optional because users can generate stubs by themselves using the WSDL description);
7) test the service.
Eclipse Plug-in

Service Interface

- Stub
- Client

MAGE - GDT
- WSDL
- Skeleton

WSAG
- Gateway Agent
- WS Wrapper

Globus GAR

Agent Service

Eclipse SDK
From the mOSAIC Dow

We will provide a framework to design, develop and deploy agents-based services.

- The mOSAIC Cloud agency will be conceived according a service-oriented architecture model, where agents will implement state-full, eventually mobile, services.

- Services will be stored in a repository and will be able to be deployed on a Cloud resource.

- Dynamic reconfiguration service will be supported by mobility.

The mOSAIC platform will be able to deploy and un-deploy services on resource.

- Furthermore the technology of mobile agents will be exploited to decouple service interface and service execution. [...] 

- A core set of functionalities to be designed and developed as agent based services will be defined.
Cloud Agency Architecture

Agent type
Client agent
Negotiator
Mediator
Vendor agent
Archiver
Automatic service deployer
Benchmarker
### Agents and roles

<table>
<thead>
<tr>
<th>Agent type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client Agent</td>
<td>Responsible for collecting users’ application requirements, for creating and updating the SLAs in order to grant always to best QoS</td>
</tr>
<tr>
<td>Negotiator</td>
<td>Manages SLAs and mediates between the user and the meta-broker, selects appropriate protocols for agreements, negotiates SLA creation, handles fulfillment and violation</td>
</tr>
<tr>
<td>Mediator</td>
<td>Select a vendor agent that is capable of deploying a service with the specified user requirements</td>
</tr>
<tr>
<td>Vendor Agent</td>
<td>Interacts with virtual or physical resources at provider side, and in case the required service needs to be deployed it interacts directly with the automatic service deployer</td>
</tr>
<tr>
<td>Archiver</td>
<td>Stores historical data about quality of services and resources offered by providers</td>
</tr>
<tr>
<td>Service deployer</td>
<td>Install the required service on the selected resource on demand</td>
</tr>
<tr>
<td>Benchmarker</td>
<td>Periodically build performance figures of used resources and notify the client agents about values of measured parameters</td>
</tr>
</tbody>
</table>
Needing for a Provisioning Subsystem

- The complexity of the business model related to a multi-Cloud environment imposes the automation of the offer selections.

Because of:
- A lot of providers
- Different requirements
- Dynamic changing requirements
- Service levels management:
we need a step forward in the Cloud computing evolution is the development of tools that allows the composition of services offered by different Cloud providers.

- An objective of the mOSAIC project (http://mosaic-cloud.eu) is to develop an Agency that provides advanced Cloud Services
CA Executes in the Cloud
Vision

- Cloud Agency is in charge of provisioning of Cloud Resource for mOSAIC applications.

- Provided Services:
  - Negotiation;
  - Monitoring;
  - Reconfiguration
    - .... at the Cloud Infrastructure Level.

- Before deployment Cloud Agency runs on the programmer desktop.

- After deployment Cloud agency is running in the Cloud.

- Cloud Agency can be composed of a number of containers, which are dynamically spawned on each new booked.
Scenarios

- Scenarios:

  - Cloud Agency as an independent component. As an independent component Cloud Agency can be used just to book Cloud Resources and eventually to monitor and reconfigure them also without programming with the mOSAIC SDK. Cloud Agency will be able to scale up, scale down resources or to change providers in an autonomic way. In this case users can access a graphical interface or can invoke Cloud Agency services from their applications using a REST-FULL interface.

  - Cloud Agency as component of the mOSAIC framework. As a component of the mOSAIC framework the Cloud Agency offers its services to the core components of the framework (E.G: reconfiguration at application level) and to mOSAIC application. In the second case a Connector will be developed to uses services from Cloudlets.

  - Internal work-flows: Being implemented as a MAS (Multi Agent System), the internal work-flows of the Agency are intrinsically asynchronous, because agents react on the occurrence of a message receive. At the same time they launch new events sending messages to other agents.

  - Cloud Agency APIs: Cloud Agency interface must an asynchronous APIs that is compliant with the event-driven mOSAIC programming model. To address this issue Use Case are designed in therm of Service Requests, Events and Callbacks. Access to Cloud Agency services will be enabled by HTTP REST interface. This is the model we will use to design Cloud Agency APIs for mOSAIC.
Stand Alone Scenario

Client

Accept the contract

<ContractID>83a930d9-3296-423d-be6f-4764b4a42a51</ContractID>

Negotiator

Change the ContractState to accepted

<resource>
  <type>ConnectionSpeed</type>
  <value>1Gbit</value>
</resource>

<resource>
  <type>quadrant</type>
  <value>3</value>
</resource>

Negotiator

Vendor1

Vendor2

Vendor3

Vendor4
MOSAIC Scenario

Users

mOSAIC-Based Application

Developers

mOSAIC API

mOSAIC Framework

mOSAIC Platform

Cloud Provider

VM

data
mOSAIC: Scenario
Task connections

- Task 1.2: Ontology
- Task 1.5: SLA and QoS
- Task 1.4: Cloud Agency
- Task 2.3: Semantic Engine
- Task 2.6: Negotiation Module
- Task 2.5: Provider Agent
Use Case

UC01: Install and start CA
UC02: Retrieve the list of supported vendors
UC03: Allocate new resources
UC04: Deallocate existing resources
UC05: Retrieve the list of monitoring parameters for a certain resource type
UC06: Manage monitoring metrics
UC07: Manage policies
UC08: Provide the Client with monitoring data
UC09: Evaluate the policies
UC10: Stop CA
UC11: Import SLA
UC12: Retrieve the list of resources
UC13: Resource Info
UC14: Retrieve the list of active transactions
UC15: Retrieve Transaction Info
UC16: Retrieve Resources/Vendor Info
Event Driven Design

- Actions can be distinguished into three classes:
  - **Requests.** They are used to ask the Cloud Agency for something to be executed. For example, to start a Negotiation, to accept to refuse a SLA, to change a Policy, etc. Requests, as any other action, are not blocking. It means that execution is started on remote, but the client can continue to run.
  - **Events.** They are generated events in the future by Requests and have to be handled by the requester.
  - **CallBack.** They are used to receive and handle events which follow previous Requests.
  - **Queries.** They are used to get information. For example, client asks for an SLA, for the status of a negotiation, to get the list of vendors, or the list of resources. Queries return immediately the response if it is available, an exception otherwise.

- A draft list of action for the Negotiation Use Case has been defined.
- Actors are the Cloud Agency Client and the Cloud Agency.
Event Driven Design
REST Rendering

POST / HTTP/1.1
User-Agent: curl/7.21.0 (x86_64-pc-linux-gnu) libcurl/7.21.0 OpenSSL/0.9.8o zlib/1.2.3.4
Host: localhost:8080
Accept: */*
Cookie: pyocci_user=Zm9v[1291753962|7811a4821179ff98ea96d4b44fade0512b1ff52
Content-Type: text/occi
Reply-To: localhost:8080/CallBackHAndler
Category: cfp; scheme="http://schemas.orgf.org/occi/provisioning#"; class="mixin";
<ws:Template ws:TemplateId="t1" xmlns:ws="http://schemas.orgf.org/graaap/2007/03/ws-agreement">
  <ws:Name>Test1</ws:Name>
  <ws:Context>
    <ws:ServiceProvider>ANY</ws:ServiceProvider>
    <ws:TemplateId>t1</ws:TemplateId>
    <ws:TemplateName>Test1</ws:TemplateName>
  </ws:Context>
  <ws:Terms>
    <ws:ServiceDescriptionTerm ws:Name="ANY" ws:ServiceName="Storage">
      <mod:Storage xmlns:mod="http://occi-wg.org/model">
        <title>ANY</title>
        <size>ANY</size>
        <summary>ANY</summary>
        <writetime>1000</writetime>
      </mod:Storage>
    </ws:ServiceDescriptionTerm>
    <ws:Terms>
      <ws:CreationConstraints/>
    </ws:Template>

Proposal received and negotiation started
HTTP/1.1 200 OK
Content-Length: 2
Content-Type: text/html; charset=UTF-8
Location: /users/fo0/cf0/b9ff813e-fee5-4a9d-b839-673f39746096
Server: CloudAgency OCCI/1.1
OCCI Resource Model
OCCI Resource Model
Architecture
Cloud Agency Design
Negotiation

- **Negoziazione delle risorse tra vari vendors**
- **Monitoring delle risorse in funzione dei parametri indicati negli SLA**
- **Riconfigurazione delle risorse**
mOSAIC SLA Architecture
SLA model

- The Agency will use a uniform SLA model
- SLA@SOI and **WS-Agreement** are our current reference
- **OCCI** is our current reference for description of Cloud resource
- Clients use this model to ask for Cloud Resources
- Vendor Agents translate SLA from Specific Provider Language into the internal model
<ws:Template ws:TemplateId="t1"
xmlns:ws="http://schemas.ggf.org/graap/2007/03/ws-agreement">
  <ws:Name>Test1</ws:Name>
  <ws:Context>
    <ws:ServiceProvider/>
    <ws:TemplateId>t1</ws:TemplateId>
    <ws:TemplateName>Test1</ws:TemplateName>
  </ws:Context>
  <ws:Terms>
    <ws:All>
      <ws:ServiceDescriptionTerm ws:Name="SAMPLE REQUEST0"
ws:ServiceName="SET VARIABLE">
        <mod:Storage xmlns:mod="http://occi-wg.org/model">
          <title>ANY</title>
          <size>ANY</size>
          <summary>ANY</summary>
        </mod:Storage>
      </ws:ServiceDescriptionTerm>
    </ws:All>
  </ws:Terms>
  <ws:CreationConstraints/>
</ws:Template>
FIPA Interaction protocols

• The cooperation among agents has been designed adopting well defined Agents Interaction Protocols of the FIPA standard.

• The Client Agent implements a standard FIPA Contract Net Interaction Protocol with the Negotiator Agent. It submits a call for proposal using an SLA Template with the application requirements and gets the SLA proposal.

• The Mediator implements the standard FIPA Brokering Interaction protocol. The sub-protocol used to talk with vendors is again the standard FIPA Contract Net interaction protocol.
A proxy message contains only 1 objective. The FIPA brokering interaction protocol continues until the end of objectives list.
Agents design: Negotiator

- Agents' behaviors are event-driven.
- Incoming messages generate event.
Scenari di utilizzo

Amazon VM: memoria RAM 2 GB, Larghezza di banda 250 Mbps
Rackspace storage: capacità 1 TB

Uso 512 MB di memoria RAM
Riduco la memoria RAM acquistata

Uso 950 GB di spazio nello storage
Aumento la capacità dello storage

Bandwidth minore rispetto a 250 Mbps
Cerco un nuovo provider
Jade Implementation

http://jade.tilab.com/

- Open Source
- Third Party add-ons
- JAVA
- Jade is a trademark of Telecom Italia SpA
Cloud Agency

- Negoziazione delle risorse tra vari vendors
- Monitoring delle risorse in funzione dei parametri indicati negli SLA
- Riconfigurazione delle risorse
Monitoring and Reconfiguration

- Two different Services
  - Monitoring of Cloud Resources (not applications)
  - Intelligent event-driven reconfiguration
- RESTful access
- Multi Agent implementation
- Multi model support for reasoning
- Policy based
Monitoring and Reconfiguration

Benchmarkers and monitors

Receive Data

Daemon \_1 \rightarrow Daemon \_K

Communication with users's applications

Store Data

Statistics DB

Read

Reasoners

Monitoring

Observers

Read/Write

Executors

Push event

Event Bus

Observers

Subscribe

External Application

Read/Write
Archiver

- Parses SLA
- Find available monitors for:
  - service levels
  - specific provider
- Starts monitoring

1: Read SLA
2: Find daemons
3: Install daemons
4: Start daemons
Data Collection

- Performance information are collected by the archiver.
- Data are collected, processed and stored.
- They are available to agents and to Cloud application.
Linker and Benchmark

- Linker provides access to Resource functionalities
- Benchmark uses linker to collect necessary information
- Linker is technology dependent
- The suited couple of linker-benchmark is embedded into the user application.
Reasoner is a MAS itself.
Observers subscribe to the Event bus to detect events.
Observers make decision about what actions to be taken.
Executors execute action according to observer requests.
Executors can raise new events.
Configuring interaction

- Reasoner configuration
- Observers and Executors are launched according to it
- Messages among agents are routed by an agent again
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="executorListType">
    <xsd:sequence>
      <xsd:element name="executor" type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="parameterType">
    <xsd:sequence>
      <xsd:element name="par_type" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="par_value" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="parameterListType">
    <xsd:sequence>
      <xsd:element name="parameter" type="parameterType" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="descriptionType">
    <xsd:sequence>
      <xsd:element name="type" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="parameters" type="parameterListType" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="eventType">
    <xsd:sequence>
      <xsd:element name="executors" type="executorListType" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="description" type="descriptionType" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="event" type="eventType"/>
</xsd:schema>
Information flow

Monitoring Results DB
1: Read Monitoring Results

Observer
2: Process Results
3: Generate Event

Routing Agent
4: Send Event

Executor
5: Forward Event
5: Push Event

Event Bus
Case Study

- A Cloud Agency has booked a Storage from GoogleAppEngine
- WriteTime of his application should be 1000ms/MB
- The user has built his application using a Cloudlet that uses the Storage by installing on Google a Servlet
- The Servlet embeds a benchmarker
- We aim at detecting when the average value of WriteTime crosses a certain threshold
Google example

User's application is a servlet
Each time the servlet is invoked the
Configuration Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <observers>
    <observer>
      <classname>MonitorAgent</classname>
      <bin_path>./reasoner/monitorobserver.jar</bin_path>
      <package>monitorobserver</package>
    </observer>
    <observer>
      <classname>ReconfigurationAgent</classname>
      <bin_path>./reasoner/reconfigurationobserver.jar</bin_path>
      <package>reconfigurationobserver</package>
    </observer>
  </observers>
  <executors>
    <executor>
      <classname>Logger</classname>
      <bin_path>./reasoner/loggerexecutor.jar</bin_path>
      <package>loggerexecutor</package>
    </executor>
    <executor>
      <classname>ReconfigurationExecutor</classname>
      <bin_path>./reasoner/reconfigurationexecutor.jar</bin_path>
      <package>reconfigurationexecutor</package>
    </executor>
  </executors>
</config>
```
SLA example

```xml
<ws:Template ws:TemplateId="t1" xmlns:ws="http://schemas.ggf.org/graap/2007/03/ws-agreement">
  <ws:Name>Test1</ws:Name>
  <ws:Context>
    <ws:ServiceProvider>google</ws:ServiceProvider>
    <ws:TemplateId>t1</ws:TemplateId>
    <ws:TemplateName>Test1</ws:TemplateName>
  </ws:Context>
  <ws:Terms>
    <ws:All>
      <ws:ServiceDescriptionTerm ws:Name="google_storage01" ws:ServiceName="SET VARIABLE">
        <mod:Storage xmlns:mod="http://occi-wg.org/model">
          <title>ANY</title>
          <size>ANY</size>
          <summary>ANY</summary>
          <writetime>1000</writetime>
        </mod:Storage>
      </ws:ServiceDescriptionTerm>
      <ws:Terms/>
    </ws:All>
  </ws:CreationConstraints/>
</ws:Template>
```
Reasoner implementation

- Receive Data
- Daemon_1
- Daemon_K
- Archiver
- Store Data
- Statistics DB
- Read
- Rel-Observers
- Prolog Observer
- Executors
- Push event
- Event Bus
- New Call for Proposal
Involved Agents

- An **Archiver Agent** that starts a GoogleAppEngine linker to monitor Storage Write Time;

- A **Routing Agent** that forwards messages according to a configured path.

- An Agent based implementation of the **Event Bus**

- A **Rule Based Observer** that periodically processes collected data and detects saturation or SLA violations;

- An **Executor** that publishes critical events on the Event Bus

- An **Observer** that uses a Prolog Engine to detect events and ask for SLA re-negotiation.

- An **Executor** that asks for re-negotiation.
Relational Observer

- From the SLA it is possible to select provider
- Resource
- Parameter to be monitored
Defining a new rule
Defining a new rule

Values aggregation: average

Rule relation:
- =
- % of SLA val
- <=
- >=

Verification M:
- Periodical, with period [s]:

On event from: Logger

Action:
- Send event

Active rules:
- Add Rules

Disable rules:
- Add Rules

Executors:
- ALL
- Select Executors

OK Cancel
Defining a new rule

Values aggregation: average

Rule relation: <

% of SLA value: 98

absolute value: 

Verification Mode:

Periodical, with period [s]: 

On event from: Logger

Action:

Send event

Active rules: Add Rules

Disable rules: Add Rules

Executors:

ALL

Select Executors

OK Cancel
Examples of rules

Rule list:

IF the average value is lower than the 98% of sla value THEN send event. Verification on event from 'ReconfigurationExecutor'. State: ACTIVE
IF the average value is not equal to the 98% of sla value THEN disable rules. Verification on event from 'Logger'. State: ACTIVE
IF the last value is equal to 1000.0 THEN send event. Rule period [s]: 60. State: IDLE
IF the minimum value is greater than the 98% of sla value THEN active rules. Rule period [s]: 1000. State: ACTIVE
Prolog Based Observer

:- dynamic(addEvent/6).

saturation(ID) :- event(storage,ID,free_space,average,lower,X), X < 5.
saturation(ID) :- event(storage,ID,free_space,average,lower_equal,X), X <= 5.
saturation(ID) :- event(storage,ID,free_space,last,lower,X), X < 5.
saturation(ID) :- event(storage,ID,free_space,last,lower_equal,X), X <= 5.
slow(ID) :- event(compute,ID,free_memory,average,lower,X), X < 5, event(compute,ID,cpu_usage,average,greater,Y), Y > 95.
slow(ID) :- event(compute,ID,free_memory,average,lower,X), X < 5, event(compute,ID,cpu_usage,average,greater_equal,Y), Y >= 95.
slow(ID) :- event(compute,ID,free_memory,average,lower_equal,X), X <= 5, event(compute,ID,cpu_usage,average,greater,Y), Y > 95.
slow(ID) :- event(compute,ID,free_memory,average,lower_equal,X), X <= 5, event(compute,ID,cpu_usage,average,greater_equal,Y), Y >= 95.

action(saturation,update,storage).
action(slow,add_resource,compute).

% program interface
performAction(ACTION,RESOURCE,ID) :- action(saturation,ACTION,RESOURCE), saturation(ID).
performAction(ACTION,RESOURCE,ID) :- action(slow,ACTION,RESOURCE), slow(ID).
addEvent(RESOURCE,ID,P ARAMETER,AGGREGATION,RELATION,PERCENTAGE_VALUE) :- asserta(event(RESOURCE,ID,P ARAMETER,AGGREGATION,RELATION,PERCENTAGE_VALUE)).

%
Published Events

Tue May 10 13:53:18 CEST 2011
Event type: observer_event
parameter_result: 953.06666666666
parameter_type: writetime
relation: lower
aggregation: average
parameter_sla_value: 1000.0
resource: Storage
parameter_limit: 980.0
resource_id: google_storage01
provider: google

Tue May 10 13:53:18 CEST 2011
Event type: reconfigurator_event
resource: storage
action: change_provider
resourceId: google_storage01
Collaborations

- Inputs for Cloud ontology
- Inputs for platform components like monitoring, billing, negotiations, VM management (open-source)
- Test the API implementation and platform
- Participate to the user forum
- Bring new use cases
- Offer your Cloud resources
- Coordinated efforts for standardization and avoidance of vendor lock-in
Conclusion

- When interoperability and portability of applications across heterogeneous Clouds are supported, autonomous services which can automatically manage collection, *negotiation and monitoring* of Cloud resources *represent an added value for users within the sky computing paradigm*.

- *Cloud Agency implements a delegated software* that can access the computing utility market relative to the state-of-art of Cloud computing, on behalf of the user, to maintain always the best resources configuration that satisfies the application requirements.

- It is in charge to *provision* the collection of Cloud resources, *from different providers*, that continuously meets the requirements of user’s applications.

- According to the available offers, it generates a *service-level agreement* that *represents the result of resource negotiation* and booking with available providers.

- The user is able to delegate to the Agency the *monitoring of resource utilization*, the necessary *checks of the agreement fulfillment* and eventually *re-negotiations*.
DETAILS IN


- **SLA management:** Salvatore Venticinque, Rocco Aversa, Beniamino di Martino, Massimiliano Rak and Dana Petcu, *A Cloud Agency for SLA Negotiation and Management*, EuroPar 2010 - workshops, Springer, LNCS 6586, 547-554


- **Cloud agency:** Salvatore Venticinque, Rocco Aversa, Beniamino Di Martino, Dana Petcu, *Agent based Cloud provisioning and management*, Accepted for CLOSER 2011, May 2011.

- **API design – layers:** Dana Petcu, Massimiliano Rak, Ciprian Craciun, *Towards a cross-platform Cloud API*, Accepted for CLOSER 2011, May 2011.


- **Scheduling agents:** Marc E. Frincu, Norha M. Villegas, Dana Petcu, Hausi Muller and Romain Rouvoy, *Self-Healing Distributed Scheduling Platform*, Accepted for CCGrid 2011, May 2011
Cloud Computing: Project and Initiatives 2011
http://ccpi.unina2.it
mOSAIC Project Dissemination and Collaboration Workshop

August 30th, 2011 - Bordeaux – France

Collocated with: *Euro-Par 2011 International Conference*

**Main topics**
- Clouds federation
- Clouds interoperability
- Cloud programming
- Cloud computing standards
- HPC and cluster at demand
- Cloud at home
- Cloud delivery models
- Cloud-based applications
- Cloud middleware
- Cloud security

**Workshop Co-Chairs**

Beniamino Di Martino, Second University of Naples, Italy
Dana Petcu, Western University of Timisoara, Romania

**Proceedings:** Lecture Notes in Computer Science (Springer)
Thanks for your attention!
Questions ??

salvatore.venticinque@unina2.it