INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Report: IoT platform on an Autonomic Cloud

MS Innovative Smart Systems 2016-2017
GAUTIER, RENAUD
LOUBET, GAËL
PERSAND, KAVEENA
Supervisor :
GARZONE, GUILLAUME

Contents

List of Figures

Introduction

In the "Middleware and Service" course, a module named "Adaptability: Cloud and Autonomic Computing" was dedicated to the introduction and the understanding of the concepts of cloud and autonomic computing.

Four practical sessions were dedicated to this subject and three topics were offered. The practical sessions were conducted in a group of three and aimed to offer a Paas-type cloud computing service.

The first topic's objective was to get started with concepts of cloud computing and OpenStack's tool suite. The second aimed to deploy an OM2M architecture on a cloud. Finally, the last topic tackled autonomic management of scalability and elasticity of a service in case of heavy load.

During those sessions, we had access to a cloud infrastructure hosted by LAAS-CNRS, allowing us to deploy IaaS-type services, based on OpenStack and being part of INTEL-INSA-LAAS minilabs. That infrastructure was hosted on the public domain laasnetexp.fr and consisted of four INTEL servers, each having 36 physical cores distributed over two processors.

The notion of cloud computing often crops up with ubiquitous computing. The former could be used to deliver an implementation of the latter though the virtualisation of resources whether they are in terms of computing power or data storage. Cloud computing is often encountered daily through cloud data storage. Though being a rudimentary and limited application of the notions of cloud, it is nonetheless a trending application of cloud computing. With companies such as Amazon, Google, Oracle and Microsoft offering cloud services packages and other notable companies using these resources to host their own customer services, cloud computing has gained quite some popularity recently. The cloud service that we have deployed during the practical sessions revolved more around using a cloud for deploying sensor networks.

1 Understanding Cloud Computing

1.1 Definition of Cloud Computing

Cloud computing is a type of Internet-based computing that provides shared computer processing resources and data (1) to different users on demand. Clients' processes exploit the computing capacity of distant servers. Thus, resources are mutualised between users. Each resource can be configured independently on virtual machines according to the profile of the user and the current workload. Users also see provided resources as unlimited and relocated.

Cloud computing allows sharing of users' data and the virtualisation of users' instances, which are executed on relocated virtual machines.

Cloud computing has been bolstered by the decrease in cost of storage units, the adoption of autonomic computing, and the widespread adoption of hardware virtualisation at processor level.

1.2 Advantages of Cloud Computing

For users, cloud computing is an easy way to quickly deploy services. They do not have to manage their own local architecture and thus, through cloud computing the user is capable of deploying services without the cost of maintaining hardware resources. The amount of resources available on cloud services is significantly higher than what the client would generally own.

Due to the flexibility given by OpenStack, a client can manage the storage and computing resources while the system is running. That is why one can see the resources as being infinite and delocalised.

Since resources are shared and that it is easy to determine the effective usage of a user, maintenance cost remains low. The client's bill is usually commensurate to his consumption, id est the amount of resources used and the duration of use.

Dynamic management of hardware resources is also facilitated. Consequently, the service provider can adapt in real-time resource allocation with respect to the clients' demands and servers' load. He can thus modify the resources allocated to a client without the change being perceived by the client. By virtualising resources, the service provider can offer custom, flexible, and load variation resistant services.

Hence, availability and continuity of service are two assets of cloud computing.

Using a cloud structure allows the service provider to focus on the service itself, with minimal effort for resource management. Different levels of services can be offered to the client. These can be classified into the following categories:

- IaaS: Infrastructure As A Service, is a type of cloud computing that focuses on offering services to manage hardware architectures. The user can request hardware resources allocation.

- PaaS: Platform As A Service, is a type of cloud computing that focuses on offering services to manage virtual architectures. The user can request computing resources; and must manage its software and data repositories.

- SaaS: Software As A Service, is a type of cloud computing that focuses on offering services to use softwares. The user can request software resources and must manage its data.

2 Using Cloud Computing infrastructure

2.1 Position of OpenStack

OpenStack is a tool suite, an environment that allows deployment of a cloud infrastructure. It offers infrastructure management and supervision service via a platform. It is thus possible to allocate physical resources to a set of virtual machines that can be deployed. These virtual machines can obviously be configured as per user demands, one may hence install applications on the infrastructure offered.

2.2 Features of OpenStack

OpenStack provides different services through it's modular architecture, each one of them focusing on one aspect of the resource management needed to configure the virtual machines.

Nova: (Compute service) manages the resources needed for computing.

Swift: (Object storage) manages the storage of data. Swift implements redundancy by duplicating data between multiple servers, to respond to cases where a server goes down.

Neutron: (Networking service) manages and manipulates networks and IP addressing within OpenStack. It can be extended with plug-ins to communicate with network management equipment. Nexus's architecture was designed following the philosophy of next-gen network called SDN, allowing a more flexible network management.

Horizon: (Dashboard service) allows the user to manage all the aforementioned services through a web-page interface. Users are able to create and delete instances, manage security, network, storage space, as well as other configurations of these instances. The default service also provides a RESTful API to manage the different resources, as well as a command line utility.

Keystone: (Identities management)

Glance: (Images management)

The following configurations were used for the PaaS deployed :

Operating System: Ubuntu 14.04 TLS

OpenStack Version: Kilo (April 2015)

OpenStack Licence: Apache, Version 2.0

Instances were mainly created through the Horizon service, by instantiating images provided by LAAS-CNRS. A custom image (Snapshot) was also created from an instance, in order to have an image with the necessary softwares and scripts already installed.

The instance from the snapshot was created using the REST API of Horizon. By first retrieving a token to request the access to the API, several requests were then issued to retrieve different information needed to create a new instance. The requests issued for information retrieval were as shown below:

Type	Url	Goal
GET	http://localhost:5000/v2.0/tokens	get the token
GET	http://localhost:5000/v2.0/tenants	get the credentials
GET	http://localhost:8774/v2/tenant_id/images	get the list of available images
GET	http://localhost:8774/v2/tenant_id/flavors	get the flavors of the image
GET	http://localhost:8774/v2/networks	get the UUID

Table 1: List of the URL used for creating a new instance

The different requests were established by following a cookbook for using Horizon's REST API. The retrieved data was used to configure an application that sent the following request to create a new instance from the snapshot we created.

Request type	POST
target URL	http://localhost:8774/v2/tenant_id/servers
Headers	X-Auth-Token: token_value ; Content-Type: application/json

Payload	{
	"server": {
	"name": "image_name",
	"imageRef": "reference to the image",
	"flavorRef": "reference to the flavor",
	"key_name":"name of already-uploaded key",
	"networks": [
	{
	"uuid": "id of the network"
	}
	],
	"security_groups": [
	{
	"name": "default"
	}
	],
	"metadata": {
	"My Server Name": "Create Instance via REST API"
	}
	}
	}

Table 2: Request used for instantiating a snapshot

With :

- item_name: the name given to the image
- reference to the image: in the "http://localhost:8774/v2/tenant_ID/images/image_ID" format
- reference to the flavor: in the "http://localhost:8774/v2/tenant_ID/images/flavor_ID" format
- name of already-uploaded key : name of an already-configured ssh key
- id of the network: UUID of a network

3 Deploying and Adapting an IoT Platform Through Autonomic Computing on the Cloud

3.1 Deploy a PaaS Architecture Based on OM2M

3.1.1 Architecture of OM2M

OM2M is an open source project supported by Eclipse and initiated by LAAS-CNRS, whose purpose is to implement IoT standards such as oneM2M and SmartM2M. As defined in the oneM2M standards, OM2M provides a horizontal machine-to-machine communication platform, which can be deployed on servers, gateways and even devices, which allows them to interact with each other through an interoperable platform. The platform allows the management of certain aspects of the hardware, and can be controlled through a REST API.

Its architecture comprises three different nodes :

- applicative nodes (AE - application entity) corresponding to the sensors and actuators on the network
- infrastructure node (IN) allowing the storage and the processing of the information
- middle nodes (MN) acting as gateways between AEs and the IN, or with other MNs

Other Non oneM2M nodes (NODN) can be added to the structure, representing devices or servers. Those non oneM2M compatible nodes can exist on the network and interact with intermediary nodes.

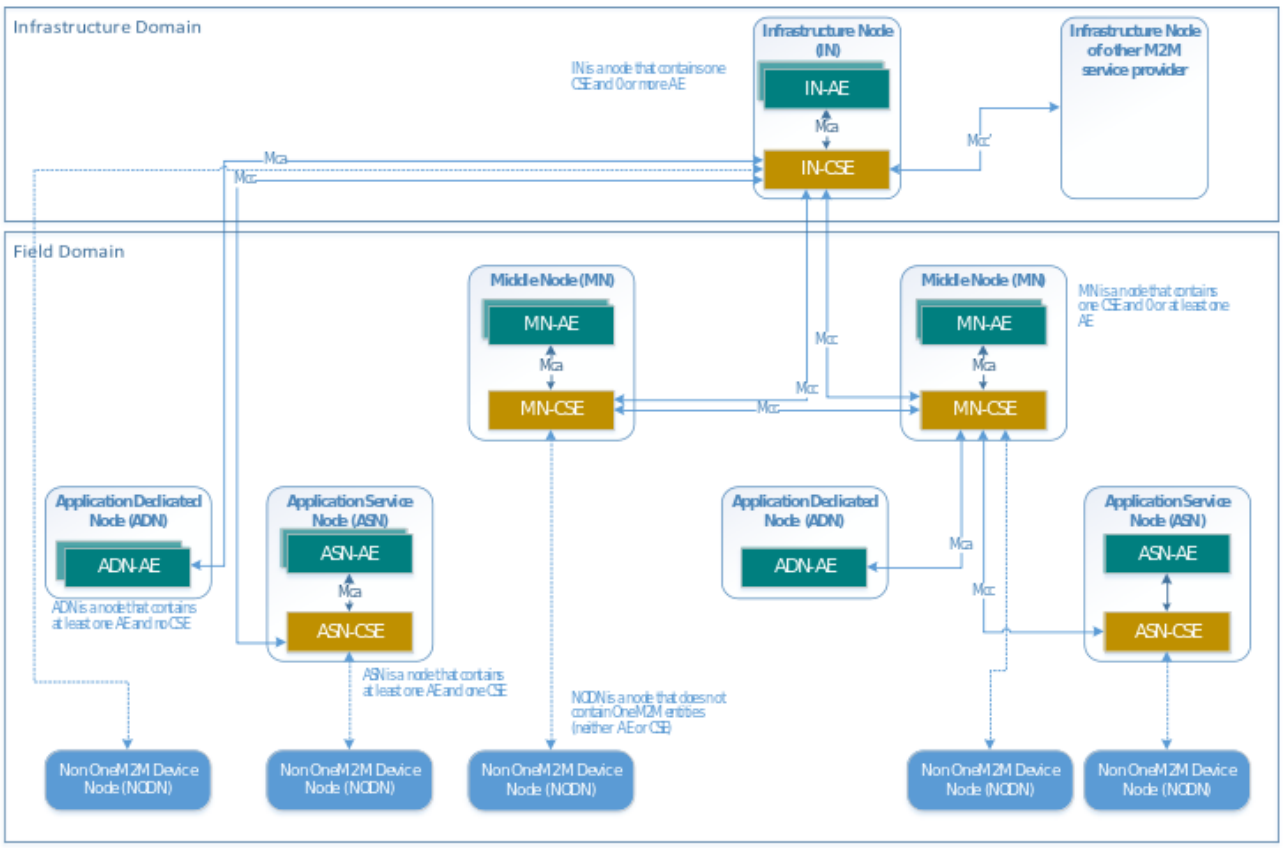

Figure 1: OM2M Architecture

In our case, we deployed an IN with its database deployed on another instance (thus allowing this resource being shared between INs) and an MN (with a local database). Everything was automated from the resource creation on the IN using the REST API, starting up of the different instances, to the shutdown of each instance.

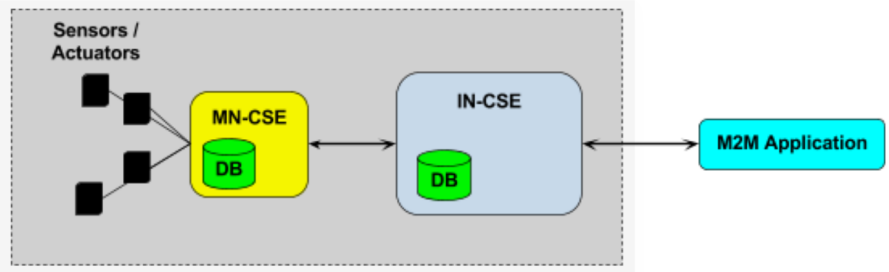

Figure 2: Deployed architecture

3.1.2 OM2M Deployment on the Cloud

Deployment of OM2M on a cloud allows autonomic management of its allocated resources. Hence, depending on the traffic load it is possible to adapt its computing and storage resources. Resource optimisation is similarly possible and may involve potential economical advantages and an enhanced load management. Without using cloud services and for similar investments, the system would waste most of its resources most of the time and would still be unable to cope with peaks in traffic load.

As a cloud service provider (PaaS), the OM2M platform we deployed was intended for entities intending to deploy a smart device network, which would most likely communicate wirelessly or for implementing machine-to-machine communication. One may assume that the deployment of such services by the clients should be quite rapid, financially inexpensive, and most probably temporary (no client-based equipment relative to deployment of an OM2M platform, proofs-of-concepts, equipment calibration, ...). Aspects such as scalability, accessibility and availability prove to be a turning point for choosing cloud services. In our implementation, the cloud service provider (IaaS) was LAAS-CNRS, whose service is primarily intended for research and teaching purposes and well as for students' use.

3.1.3 IN-CSE Deployment on the Cloud

In this scenario, we have deployed an IN-CSE and its database on the cloud. Deployment of an IN-CSE on the cloud can be advantageous in terms of accessibility, scalability and dynamic resource management. In an oneM2M compliant IoT architecture, each architecture has only one IN-CSE. The IN-CSE is also the pinnacle of this architecture. All MN-CSEs are typically registered to the IN-CSE. Such an architecture may incur a significant load on the IN-CSE. This problem may be curbed by deploying the IN-CSE and its database on a cloud.

The main advantages of deploying an IN-CSE on a cloud are the properties of:

- Accessibility: if the IoT architecture is expected to be deployed on a large geographical area with Internet access, then deploying the IN-CSE on a cloud diminishes the work required to provide access to the architecture's data as the IN-CSE already has access to the Internet. Through virtualization, relocation and obviously duplication, availability is guaranteed at all times and in any place.

- Scalability and dynamic resource management: these are advantages inherent to a cloud-based service. In the case where the IN-CSE centralises the data, it may be loaded with requests. Hence, an IN-CSE should be able to process high request rates. The request rates may not be uniformly distributed in time. If

the IN-CSE is designed to process the highest request rates with good performance, wastage of resources may occur if the service is not solicited at the same rate throughout its active time. By deploying the IN-CSE on the cloud with a proxy, one may deploy an architecture capable of processing high request rates and optimising resources when the service is rarely solicited. In our case this is done by deploying a proxy and the IN-CSE on the cloud. Dynamic re-allocation of the IN is not perceived by the service consumer, as the latter only interacts with the proxy.

3.1.4 MN-CSE Deployment on the Cloud

There is no significant benefit in deploying an MN-CSE node on the cloud. An MN's main role is to act as a gateway between different devices scattered around a small geographical area (for e.g. around a house, on a local network) and the IN that is located on the Web. If the MN is deployed on the cloud, that would require devices to have access to the Web. If the sensors could do so, they could directly register themselves on the IN node (also deployed on the cloud), rendering the MN node insignificant.

Furthermore, smart sensors seldom have the capability to access the Web and implement oneM2M standards, the use of an IPE is hence required to connect smart sensors on the IoT architecture based on oneM2M. The IPE needs to interact with the sensors and the MN on the cloud. Therefore, an additional application (deployed on a computational node) would be required. The inconveniences of deploying an MN node on a cloud outweighs its advantages (accessibility, scalability, and dynamic resource management).

If an MN is deployed locally and connected to an IN-CSE that has been deployed on the cloud, the MN may periodically send backups of its data to the IN. Doing so centralises the data from the various MNs and alleviates their workload.

In our case, we need to deploy our MN on the cloud as per the guidelines given. It may be interesting to separate the MN's database from its instance for the same reasons as mentioned for the IN.

3.1.5 IN-DB Deployment on the Cloud

The main advantage in deploying the IN's database on the cloud is for flexible data storage. The provider may extend the database's storage capacity on will by using the cloud's resources.

3.2 Autonomic PaaS Architecture

3.2.1 Autonomic Computing Paradigm

Autonomic computing is about making systems self-managing, and thus removing humans of the control process. It is often compared to organic systems that factor in numerous parameters to constantly adapt to the environment. The four main properties expected from autonomic systems are:

- Self-configuration
- Self-optimisation
- Self-healing
- Self-protection

Human interaction with the system is shifted from system manager to the one who configures the rules for adaptation.

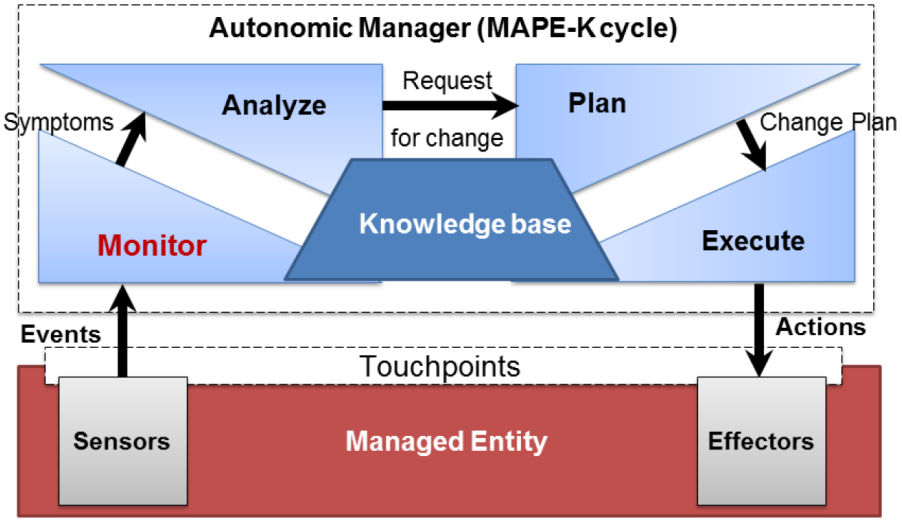

Figure 3: MAPE-K autonomic loop

MAPE-K is a cyclic architecture for autonomic control. The five main components of this architecture are:

- Monitor: measures events from the environment and extract symptoms
- Analyse: analyses symptoms, looks for causes and how to compensate, and hence issues demands for changes
- Plan: interprets demands for changes and determines the sequence of actions to be undertook in an action plan
- Execute: executes the changes requested by the planner on the environment
- Knowledge base: interconnects the other four components by storing and relaying data from each component

3.2.2 Post-creation Script

A post-creation script is executed after creation of an instance and enables self-configuration. In our case, having used a snapshot, we had no need in implementing a post-creation configuration script. An instance created from the snapshot would be identical to the previous one, and hence be configured similarly.

3.2.3 Offering an OM2M platform as PaaS

When acting as a cloud-service provider, the platform provided as a PaaS enables the use of an OM2M platform that is pre-installed, pre-configured, with adaptable resources, and resilient to increases in workload.

To deploy our OM2M platform, we instantiated a master IN-CSE which would run all the time. This would run an IN-CSE that had its database on the cloud. The instance was configured so as to launch the IN-CSE when

the instance is launched. This was accomplished by modifying its System V Init Script, so that the IN-CSE would be launched upon launching the instance, stopped and rebooted accordingly.

To provide this service, ready-to-use virtual machines must be created on demand. A snapshot of an operational and configured instance is used as a stencil for creating these ready-to-use instances. The master IN's instance was used as a snapshot. A MAPE-K based architecture was instantiated to cope with load variation issues (If the number of incoming requests would exceed a certain threshold, then an additional instance would be created if the number of IN instances were under a certain threshold. The latter would be destroyed in case the number of requests plummeted).

Our OM2M service consisted of three different types of instances:

- IN-DB: instance where the database of the IN-CSE was hosted.
- IN-CSE: a replicable instance with OM2M's IN-CSE node installed. The installed IN-CSE node used the database hosted on the IN-DB instance, so that all the instances' IN-CSE node refered to the same database.
- ProxyLoadBalancer: instance that acted as a proxy between the client and the actual IN-CSE, by redirecting all the request traffic. It also supervised and regulated the number of IN-nodes to be commensurate to the request rate. It could destroy or create IN-CSE instances according to the current request rate.

With these three instances, we offered a platform with OM2M's IN-CSE node installed that was capable of autonomic adaptability. The platform would optimise the usage of hardware resources relative to the current traffic and consequently simultaneously avoid congestion and resource wastage. The client only needed to know the IP address of the ProxyBalancerInstance, and use that address as if it was the address of an IN-CSE server.

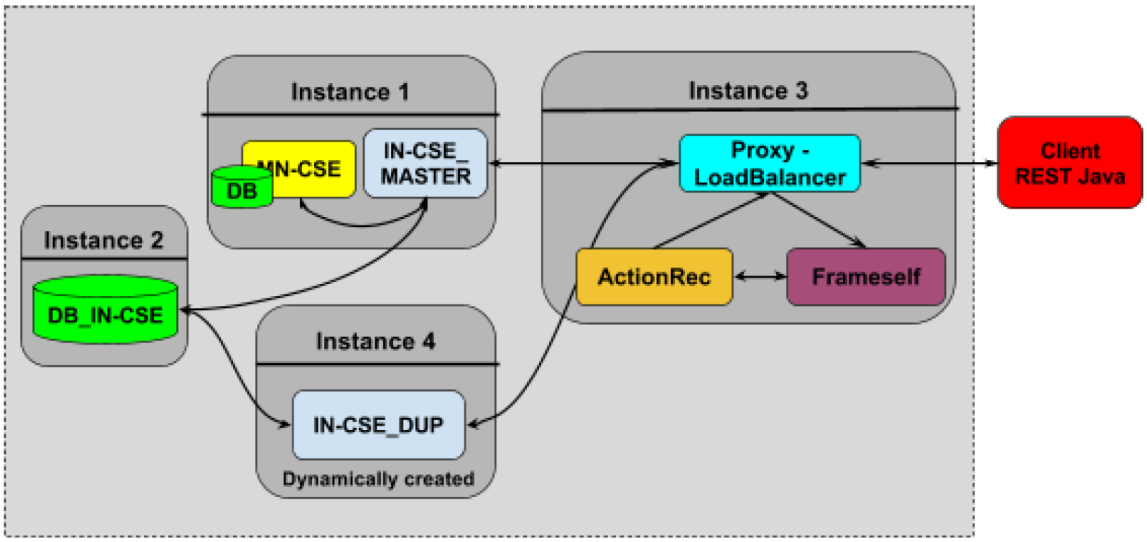

Figure 4: Final OM2M architecture

3.2.4 DROOLS Rules

DROOLS rules established for each component of our MAPE-K loop are attached in Appendix.

Conclusion

As previously stated, the platform deployed was intended for sensor networks. With these often having limited processing power and limited storage space, the complexity of the data processing can be shifted upper in the hierarchy. If the unit to which the complexity is shifted to is deployed on a cloud service, that network's data processing could benefit from the perks inherent to cloud computing (i.e. availability, scalability, ...). This architecture becomes increasingly interesting as the nodes need not have tremendous intelligence to be part of a smart system. The cloud service is seen a a single entity by the client but may be implemented using multiple hardware resources.

The service was successfully deployed but could still have exploited the capabilities of cloud computing to a further extent. As mentioned earlier, the main advantage of deploying the IN's database on a cloud was to benefit from modular storage allocation. This concept could have been integrated in the PaaS to offer a system with better dynamic resource allocation.

The autonomic control implemented was a basic one, and could have been enhanced by allowing the system to adapt its own thresholds. This would have been closer to the goals envisioned by autonomic computing.

Finally the different topics approached during the practical work allowed:

- comprehension of the cloud
- deployment of a cloud as a PaaS architecture
- use of OpenStack
- understanding of autonomic computing's concepts
- installation of an autonomic computing process based on MAPE-K architecture
- use of Frameself

A Appendix: Drools Rules

Monitor rules: Set of rules defined for symptom inference

```
rule "add␣HighRequestsRate"
        when
                Event(category == "RequestsRate", $value: value, Integer.
                    parseInt(value) >= 2)
        then
                Symptom symptom = new Symptom();
                symptom.setCategory("HighRequestsRate");
                symptom.setValue($value);
                symptom.setTimestamp(new Date());
                symptom.setExpiry(new Date(System.currentTimeMillis()+4000));
                insert(symptom);
end

rule "add␣LowRequestsRate"
        when
                Event(category == "RequestsRate", $value: value, Integer.
                    parseInt(value) < 2)
        then
                Symptom symptom = new Symptom();
                symptom.setCategory("LowRequestsRate");
                symptom.setValue($value);
                symptom.setTimestamp(new Date());
                symptom.setExpiry(new Date(System.currentTimeMillis()+4000));
                insert(symptom);
end

rule "add␣HighIN-CSENumber"
        when
                Event(category == "IN-CSENumber", $value: value, Integer.
                    parseInt(value) >= 2)
        then
                Symptom symptom = new Symptom();
                symptom.setCategory("HighIN-CSENumber");
                symptom.setValue($value);
                symptom.setTimestamp(new Date());
                symptom.setExpiry(new Date(System.currentTimeMillis()+4000));
                insert(symptom);
end

rule "add␣LowIN-CSENumber"
        when
                Event(category == "IN-CSENumber", $value: value, Integer.
                    parseInt(value) < 2)
        then
                Symptom symptom = new Symptom();
                symptom.setCategory("LowIN-CSENumber");
                symptom.setValue($value);
                symptom.setTimestamp(new Date());
                symptom.setExpiry(new Date(System.currentTimeMillis()+4000));
                insert(symptom);
end
```

Analyser rules: Set of rules defined for RFC inference

```
rule "add␣IncreaseIN-CSENumber␣rfc"
    when
                Symptom(category == "HighRequestsRate", $reqVal: value)
                Symptom(category == "LowIN-CSENumber", $inVal: value)
    then
        Rfc rfc = new Rfc();
                rfc.setCategory("IncreaseIN-CSENumber");
                rfc.setValue($inVal);
                rfc.setTimestamp(new Date());
                rfc.setExpiry(new Date(System.currentTimeMillis()+4000));
                insert(rfc);
end

rule "add␣DecreaseIN-CSENumber␣rfc"
    when
                Symptom(category == "LowRequestsRate", $reqVal: value)
                Symptom(category == "HighIN-CSENumber", $inVal: value)
    then
        Rfc rfc = new Rfc();
                rfc.setCategory("DecreaseIN-CSENumber");
                rfc.setValue($inVal);
                rfc.setTimestamp(new Date());
                rfc.setExpiry(new Date(System.currentTimeMillis()+4000));
                insert(rfc);
end
```

Planner rules: Set of rules defined for plan inference

```
rule "add␣AddIN-CSEDup"
    when
       Rfc(category == "IncreaseIN-CSENumber", $value: value)
    then
            ArrayList<Attribute> attributes = new ArrayList<Attribute>();
            attributes.add(new Attribute("in-number",$value));
            Action action = new Action();
            action.setCategory("AddIN-CSEDup");
            action.setName("AddIN-CSEDup");
            action.setAttributes(attributes);
            action.setEffector(new Effector("Proxy"));
            action.setTimestamp(new Date());
       insert(action);
end

rule "Remove␣RemoveIN-CSEDup"
    when
       Rfc(category == "DecreaseIN-CSENumber", $value: value)
    then
            ArrayList<Attribute> attributes = new ArrayList<Attribute>();
            attributes.add(new Attribute("in-number",$value));
            Action action = new Action();
            action.setCategory("RemoveIN-CSEDup");
            action.setName("RemoveIN-CSEDup");
            action.setAttributes(attributes);
            action.setEffector(new Effector("Proxy"));
            action.setTimestamp(new Date());
       insert(action);
end
```

References

[1] "Wikipedia entry: Cloud computing." [Online]. Available: https://en.wikipedia.org/wiki/Cloud_computing

[2] "Techtarget definition: Cloud computing." [Online]. Available: http://searchcloudcomputing.techtarget.com/definition/cloud-computing

[3] "Wikipedia entry: Autonomic computing." [Online]. Available: https://en.wikipedia.org/wiki/Autonomic_computing

[4] "What is autonomic computing?" [Online]. Available: http://whatis.techtarget.com/definition/autonomic-computing

[5] "Autonomic computing: It's about making smarter systems." [Online]. Available: http://www.ibm.com/developerworks/lotus/library/ls-autonomic_computing/