Regione Toscana

POR CReO 2007-13 PROGRAMMA OPERATIVO REGIONALE OBIETTIVO COMPETITIVITÀ REGIONALE e OCCUPAZIONE Fondo Europeo di Sviluppo Regionale

REPUBBLICA ITALIANA

Unione Europea

Cloud Simulator,
Design, User Manual and Test

Versione 0.1
Data 16/09/2014

Progetto iCaro
La piattaforma cloud per l'accelerazione del business delle PMI toscane
[CUP 6408.30122011.026000074]

COMPUTER GROSS

liberologico.com

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DISIT

CircleCap Technology&Mind

Informazioni sul documento

ID Deliverable	3.27.3
Titolo Deliverable	Cloud Simulator, design, user manual and test
ID Attività	3.5
N. Versione / Revisione	0.1
Natura: Bozza / Definitivo	Definitivo
Partner responsabile	UNIFI DISIT
Distribuzione: Riservato / Pubblico	Pubblico
Riferimenti Autore	DISIT lab
Data redazione	16-09-2014
Riferimenti revisore	Paolo Nesi
Data revisione	16-09-2014
Riferimenti soggetto che approva	Paolo Nesi
Data approvazione e consegna	16-09-2014

Controllo delle revisioni

Oggetto	Numero	Data
Final version	0.1	16-09-2014

Nota di riservatezza

Index

Figure Index

Legenda Acronimi e sigle

Acronimo / Sigla	Dettaglio
SM	Supervisor & Monitor

1. Cloud Simulator Requirements

- **R1**: The system should allow simple and fast generation of all entities connected to a data center or a business configuration. It should be possible, therefore, the insertion of information related to these entities and, with these data, the generation of an XML file described syntactically by the schema provided by KB.
- **R2**: The system should allow simple and fast generation of entities related to ServiceMetric, to consent study of other tools that analyze the KB.
- **R3**: For all entities generated as XML, the system should allow saving files in local (in the device that is used to access the system). Furthermore, it should be possible sending the XML file to KB in order to make information persistent.
- **R4**: Business configurations should be created over an existing data center (contained in KB), that is, the system should allow a choice of one data center contained in the KB, over which to generate a business configuration.
- **R5**: The system should allow analysis of metrics, which are contained in KB and are associated with host and virtual machines.
- **R6**: The system should allow the collection of data related to real working Host Machines. From these data should be extracted patterns to obtain models of workload for using them during simulation.
- **R7**: The system should provide two types of simulation
  - A ***fast*** simulation in which is possible to select entities (host and virtual machine) that are to be simulated and to which should be possible associate one of the models generated at point R6.
  - Simulated data must be written in an RRD file and must be sent to NAGIOS server, so that another tool can calculate on simulated data, as well as real data, the High Level Metrics to save in KB.
  - A ***real-time*** simulation in which is possible to generate new patterns from those calculated at point R6. The user should be capable to modify and to add entities to simulated data center, for example, with the addition of new virtual machines. These changes are necessary for analyzing workload generated on the real data center from such modifications.

2. Cloud Simulator Domain Model

The domain model used by this cloud simulator is based on the ontology under the KB. For this reason, many entities are similar, if not equal, to those contained on KB, except for few entities that have been created to simplify development of the simulator.

In detail the classes that begin with word *Group* are used to take the data inserted by a user on the forms and with these to create a determinate number (indicated by a user on the forms) of an object whose name is indicated by the rest of the name of the creator class eliminating the word *Group* (i.e. *GroupHostMachine* is the class responsible to create object of class *HostMachine*). While the ontology is represented (Figure 2) in hierarchical schema, the domain model (Figure 1) highlights the associations from the classes. A very important association to note is one that associates the *VirtualMachine* class to *IcaroService* class: with this association is possible to connect the part of the ontology/domain model describing business configuration to part describing data center. This association allows connection from perspective of the virtual things to perspective of the physical things.

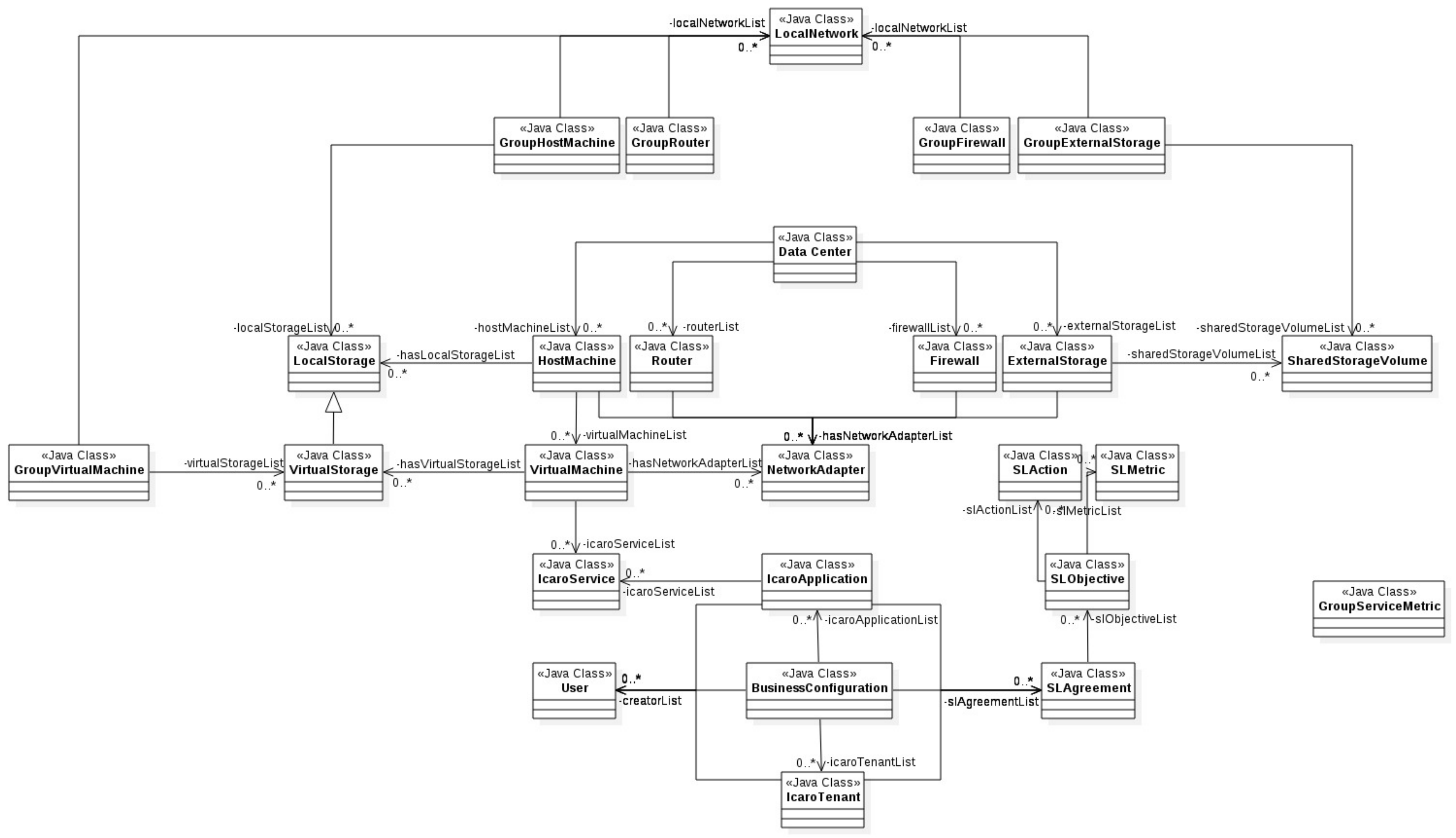

«Java Class» LocalNetwork
-localNetworkList 0..*
-localNetworkList 0..*
«Java Class» GroupHostMachine
«Java Class» GroupRouter
«Java Class» GroupFirewall
«Java Class» GroupExternalStorage
«Java Class» Data Center
-localStorageList 0..*
-hostMachineList 0..*
0..* -routerList
-firewallList 0..*
0..* -externalStorageList
-sharedStorageVolumeList 0..*
«Java Class» LocalStorage
-hasLocalStorageList 0..*
«Java Class» HostMachine
«Java Class» Router
«Java Class» Firewall
«Java Class» ExternalStorage
-sharedStorageVolumeList 0..*
«Java Class» SharedStorageVolume
0..* -virtualMachineList
0..* -hasNetworkAdapterList
«Java Class» GroupVirtualMachine
-virtualStorageList 0..*
«Java Class» VirtualStorage
-hasVirtualStorageList 0..*
«Java Class» VirtualMachine
-hasNetworkAdapterList 0..*
«Java Class» NetworkAdapter
«Java Class» SLAction
0..*
«Java Class» SLMetric
-slActionList 0..* -slMetricList
0..* -icaroServiceList
«Java Class» IcaroService
0..* -icaroServiceList
«Java Class» IcaroApplication
«Java Class» SLObjective
«Java Class» GroupServiceMetric
0..* -icaroApplicationList
0..* -slObjectiveList
«Java Class» User
0..* -creatorList
«Java Class» BusinessConfiguration
-slAgreementList 0..*
«Java Class» SLAgreement
0..* -icaroTenantList
«Java Class» IcaroTenant

Figure 1 – Domain model under Icaro Cloud Simulator

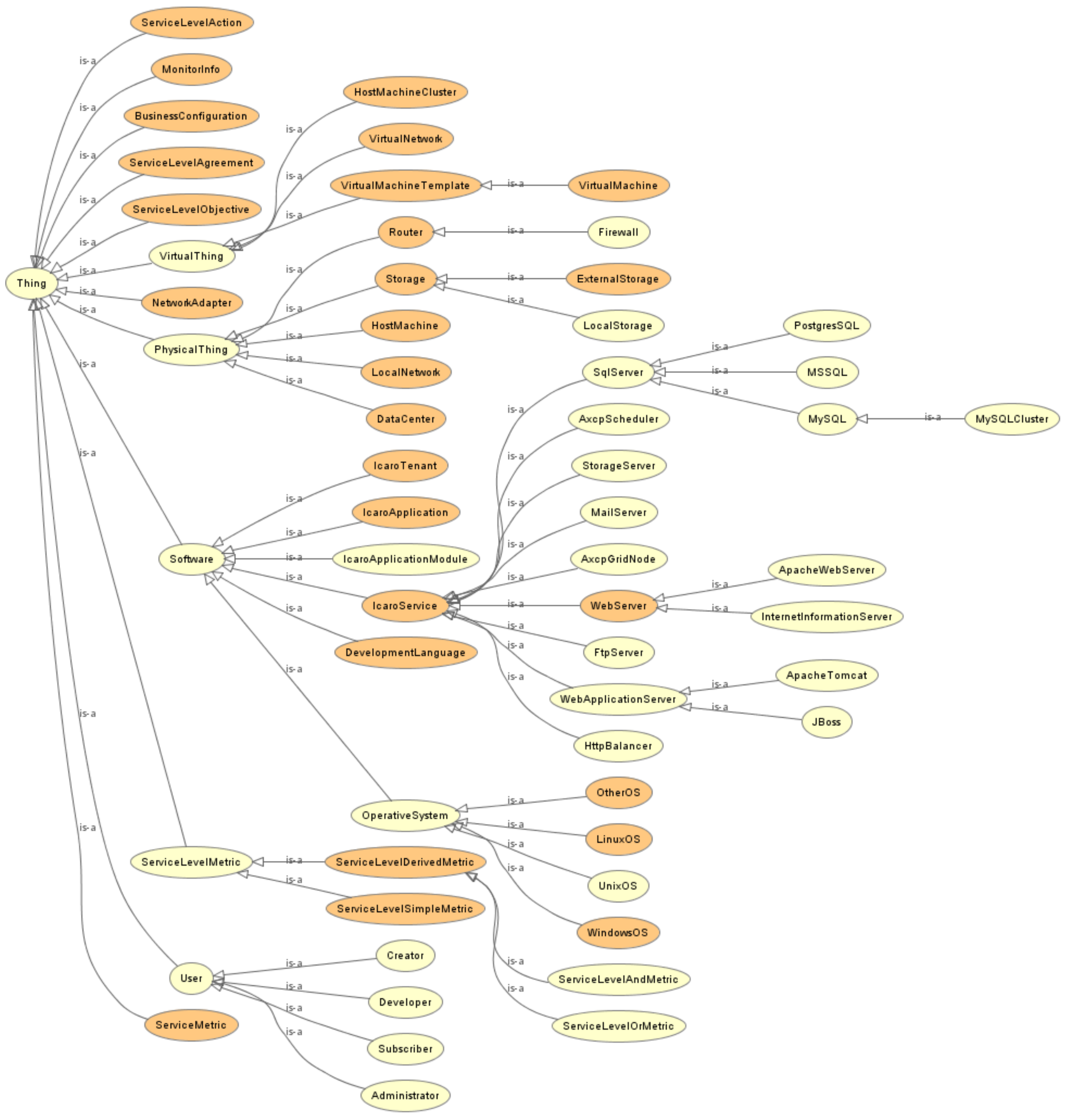


Figure 2 – Ontology under Knowledge Base

3. Architecture

In this chapter is described the architecture of classes belonging to Icaro Cloud Simulator.

The framework JSF was used (see chapter 7) to develop this system because it allows:

- A separation between the application logic and the presentation layer
- To write the system with Java and so it is possible to find some useful library to perform operations required by the tool
- Simple develop of the web interface with reusable component called Composite Component (https://docs.oracle.com/javaee/7/tutorial/jsf-advanced-cc.htm#GKHXA)
- Use of the Expression Language (https://docs.oracle.com/javaee/7/tutorial/jsf-el.htm#GJDDD) , which provides an important mechanism for enabling the presentation layer (web pages) to communicate with the application logic.
- Use of the Converters and Validators in every Input Text of the web pages (https://docs.oracle.com/javaee/7/tutorial/jsf-page-core.htm#GJCUT)

To render this description as simple as possible classes are grouped by functions that they implement. The domain model is not described another time: see chapter 3 for the description.

3.1 Viewer

Classes with *Viewer* suffix are responsible to interact with web pages adding and removing everything is visible on the web application such as buttons, panels and input text.

Icaro Cloud Simulator allows insertion of information about entities described in the ontology. This is achieved via panel containing form where users can insert information. On the web page, where is possible to create a data center, each panel represents a group of entities that are indicated by the name of the panel: for example, a panel with name *"#1 groupHostMachine"* allow defining information about a group of host machines that have all the same features (CPU, RAM, OS, etc.). The viewer classes allow the addition and the removal of panels for insert more information or delete them if they are incorrect.

These classes are obviously associated to classes that are responsible for the control: those with suffix *Controller*. In fact, any information inserted in the panel is passed to these classes to generate the XML file, after a check on the data entered.

As already mentioned, in each viewer class are present methods that add panels. Each of this method has a particular name, as it's possible to see in Figure 4, formed by the prefix *addPanel*, the name of the entity associated with that panel and the name of the entity associated with parent panel. If an entity is associated with the principal panel the name of parent panel is null and it is not inserted on the name. For example, *GroupHostMachine* is a principal entity and the method that can add a panel for this entity is *addPanelGroupMachine*(). Instead, *MonitorInfo* is a secondary entity and must be specified in which panel it must be inserted, this information can be extracted from the parameters passed to function: the object htmlPanelParent that represents, as the name suggests, the parent panel on which the new panel must be added and the index of it.

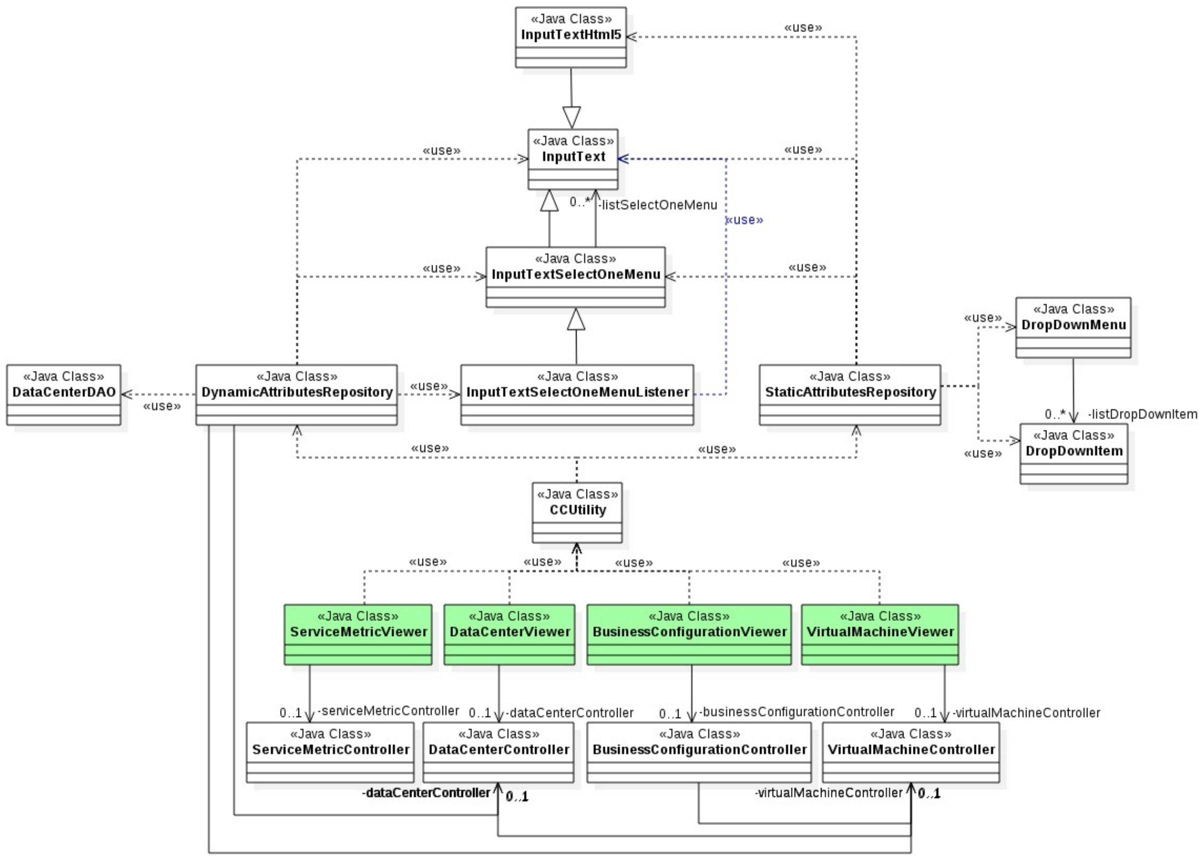
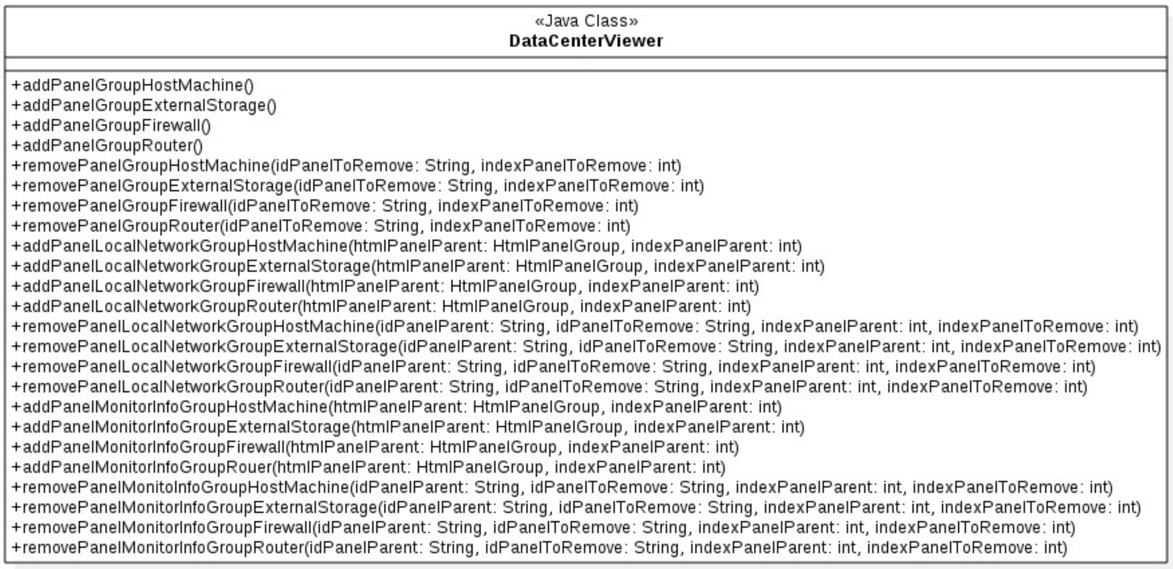

«Java Class» InputTextHtml5
«use»
«Java Class» InputText
«use»
«use»
0..* -listSelectOneMenu
«use»
«Java Class» InputTextSelectOneMenu
«use»
«use»
«use»
«Java Class» DropDownMenu
«Java Class» DataCenterDAO
«use»
«Java Class» DynamicAttributesRepository
«use»
«Java Class» InputTextSelectOneMenuListener
«Java Class» StaticAttributesRepository
0..* -listDropDownItem
«use»
«Java Class» DropDownItem
«use»
«use»
«Java Class» CCUtility
«use»
«use»
«use»
«use»
«Java Class» ServiceMetricViewer
«Java Class» DataCenterViewer
«Java Class» BusinessConfigurationViewer
«Java Class» VirtualMachineViewer
0..1 -serviceMetricController
0..1 -dataCenterController
0..1 -businessConfigurationController
0..1 -virtualMachineController
«Java Class» ServiceMetricController
«Java Class» DataCenterController
«Java Class» BusinessConfigurationController
«Java Class» VirtualMachineController
-dataCenterController 0..1
-virtualMachineController 0..1

Figure 3 – Viewer class architecture

Similarly, also the methods that remove panels are created with the same rule: prefix *removePanel*, the name of the entity associated with that panel and the name of the entity associated with parent panel, from which will be removed the chosen panel. In this case, however, the parameters to pass are more than two: the id of parent panel, the id of panel to remove and the respective indices. When a panel is removed the methods, which delete it, perform the removal of all children of that panel, avoiding reference to a nonexistent panel.


«Java Class»
DataCenterViewer

```
+addPanelGroupHostMachine()
+addPanelGroupExternalStorage()
+addPanelGroupFirewall()
+addPanelGroupRouter()
+removePanelGroupHostMachine(idPanelToRemove: String, indexPanelToRemove: int)
+removePanelGroupExternalStorage(idPanelToRemove: String, indexPanelToRemove: int)
+removePanelGroupFirewall(idPanelToRemove: String, indexPanelToRemove: int)
+removePanelGroupRouter(idPanelToRemove: String, indexPanelToRemove: int)
+addPanelLocalNetworkGroupHostMachine(htmlPanelParent: HtmlPanelGroup, indexPanelParent: int)
+addPanelLocalNetworkGroupExternalStorage(htmlPanelParent: HtmlPanelGroup, indexPanelParent: int)
+addPanelLocalNetworkGroupFirewall(htmlPanelParent: HtmlPanelGroup, indexPanelParent: int)
+addPanelLocalNetworkGroupRouter(htmlPanelParent: HtmlPanelGroup, indexPanelParent: int)
+removePanelLocalNetworkGroupHostMachine(idPanelParent: String, idPanelToRemove: String, indexPanelParent: int, indexPanelToRemove: int)
+removePanelLocalNetworkGroupExternalStorage(idPanelParent: String, idPanelToRemove: String, indexPanelParent: int, indexPanelToRemove: int)
+removePanelLocalNetworkGroupFirewall(idPanelParent: String, idPanelToRemove: String, indexPanelParent: int, indexPanelToRemove: int)
+removePanelLocalNetworkGroupRouter(idPanelParent: String, idPanelToRemove: String, indexPanelParent: int, indexPanelToRemove: int)
+addPanelMonitorInfoGroupHostMachine(htmlPanelParent: HtmlPanelGroup, indexPanelParent: int)
+addPanelMonitorInfoGroupExternalStorage(htmlPanelParent: HtmlPanelGroup, indexPanelParent: int)
+addPanelMonitorInfoGroupFirewall(htmlPanelParent: HtmlPanelGroup, indexPanelParent: int)
+addPanelMonitorInfoGroupRouer(htmlPanelParent: HtmlPanelGroup, indexPanelParent: int)
+removePanelMonitoInfoGroupHostMachine(idPanelParent: String, idPanelToRemove: String, indexPanelParent: int, indexPanelToRemove: int)
+removePanelMonitorInfoGroupExternalStorage(idPanelParent: String, idPanelToRemove: String, indexPanelParent: int, indexPanelToRemove: int)
+removePanelMonitorInfoGroupFirewall(idPanelParent: String, idPanelToRemove: String, indexPanelParent: int, indexPanelToRemove: int)
+removePanelMonitorInfoGroupRouter(idPanelParent: String, idPanelToRemove: String, indexPanelParent: int, indexPanelToRemove: int)
```

Figure 4 – Class DataCenerViewer and its methods

Each panel, that is possible to add at the web page, is generated using the composite component templates that are possible to define in a simple XHTML file. In Icaro Cloud Simulator is saved one composite component for each level of panels that it should be possible to create in a determined web page. For example, on the web page of creation of data center there are two levels of panel, the first level is represented by panels associated with entities that start with prefix *Group* and the second level is represented by panels associated with entities *MonitorInfo* and *LocalNetwork*. So, in ICS there is a folder called *datacenter* containing two composite components called respectively *firstPanel.xhtml* and *secondPanel.xhtml*.

In a composite component is possible to define the structure of the component and the attributes that should be used to add it to a web page. The class *CCUtility* associated to all viewer classes search in the repository classes the methods that can fill attributes of composite component (see section 5.1).

This architecture is designed to make the composite component as flexible as possible: in each composite component is possible to add a no predetermined number of input text of each type (input text for IP, input text for email, etc.) simply defines methods, in the repository class, that return list of input text and then, to associate, this list to relative attribute in the composite component. Not all attributes are mandatory, so if an entity does not require a type of input text just does not create the method in the repository class.

Differences between *Dynamic* and *Static* repository class depend on when the lists of attributes are created: static repository can create the lists at every time because they are predefined in the code and dynamic repository creates the lists after a precise operation and at run-time.

This solution is necessary to make composite components dynamic. If the panels must be fixed on the web page and if they are associated each to one entity, the composite components can be inserted with a simple tag definition in the principal xhtml file.

3.2 Controller

Classes with *Controller* suffix are responsible to interact with classes belonging to the domain, in a manner to create the objects associated with entities inserted from a user on the web pages: obviously before is executed a check on the inserted information. Once these objects are created, it is possible to write an XML file to save information that is in the objects: sending it directly to KB via RestAPI or downloading it on the computer. Furthermore, these classes must be able to retrieve information about entities that are requested by a user from the KB and they perform this operation using DAO classes present on the system.

These classes are important because the other classes that implement different functions must access them to catch the information about entities. For example, the viewer classes must access their respective controller to take object associated to view and it is the same for the simulator classes described in the following sections.

An example about methods that user can find in the controller classes is possible to see it in Figure 6.

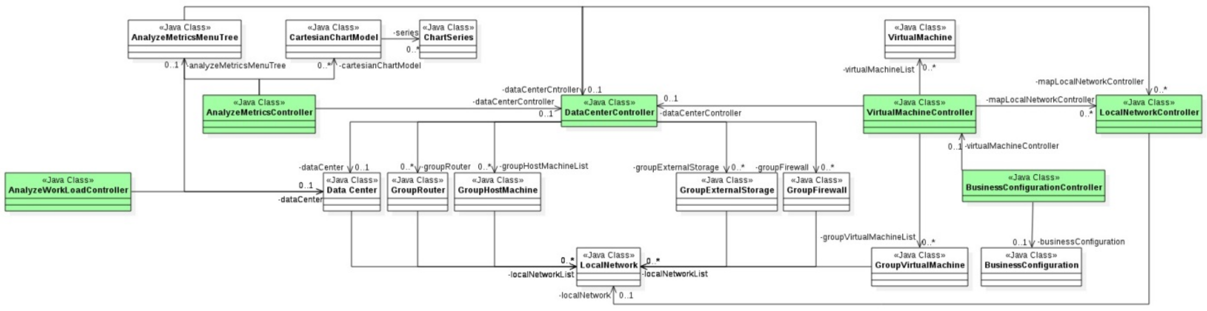

«Java Class» AnalyzeMetricsMenuTree
«Java Class» CartesianChartModel
«Java Class» ChartSeries
«Java Class» VirtualMachine
«Java Class» AnalyzeMetricsController
«Java Class» DataCenterController
«Java Class» VirtualMachineController
«Java Class» LocalNetworkController
«Java Class» AnalyzeWorkLoadController
«Java Class» Data Center
«Java Class» GroupRouter
«Java Class» GroupHostMachine
«Java Class» GroupExternalStorage
«Java Class» GroupFirewall
«Java Class» BusinessConfigurationController
«Java Class» LocalNetwork
«Java Class» GroupVirtualMachine
«Java Class» BusinessConfiguration

Figure 5 – Controller class architecture

There are many differences between the classes in Figure 6. These differences are due to the different type of information that a user can insert on web application.

In fact, when a user would like to create a *DataCenter* he can insert information about groups of entities, such as *ExternalStorage*, *Router*, *Firewall* and *HostMachine* and for this reason, in the class *DataCenterController* are present methods to create each entity from information about the group; whereas is possible to add a *BusinessConfiguration* only in an existing *DataCenter* and therefore it is necessary the method *loadDataCenter* which allows retrieving information about *DataCenter* salved on KB. At last the method called *simulate* is responsible to start a simulation.

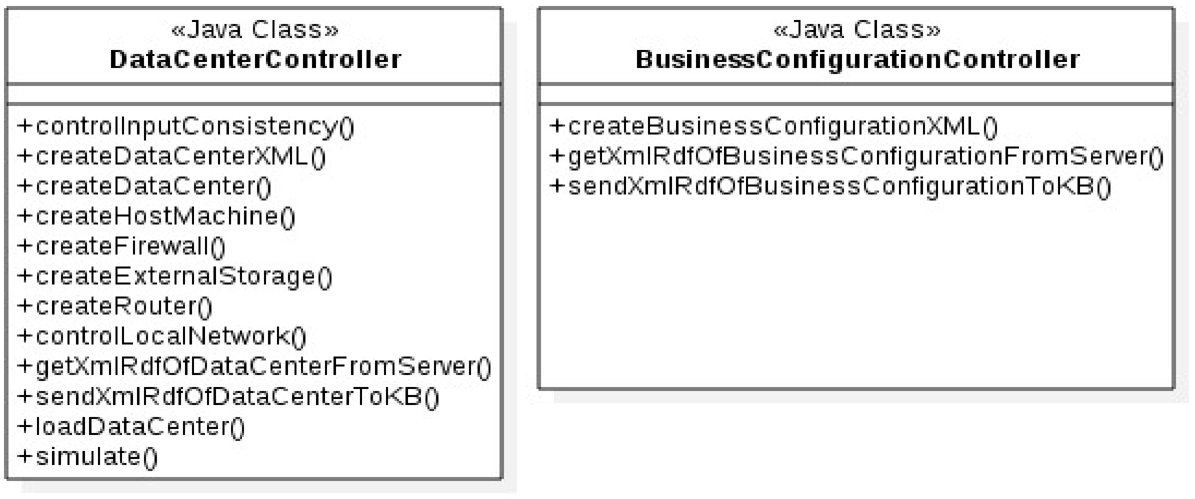

«Java Class»
DataCenterController
+controlInputConsistency()
+createDataCenterXML()
+createDataCenter()
+createHostMachine()
+createFirewall()
+createExternalStorage()
+createRouter()
+controlLocalNetwork()
+getXmlRdfOfDataCenterFromServer()
+sendXmlRdfOfDataCenterToKB()
+loadDataCenter()
+simulate()
«Java Class»
BusinessConfigurationController
+createBusinessConfigurationXML()
+getXmlRdfOfBusinessConfigurationFromServer()
+sendXmlRdfOfBusinessConfigurationToKB()

Figure 6 – Two examples of controller classes

3.3 DAO

Classes with *DAO* suffix are responsible to retrieve entities from KB. In these classes there are not methods that allow adding entities in the KB, because the addition of entities to KB is merely an operation of XML file generation and it is performed by the controller classes. In the DAO classes there are methods that consent to interrogate the KB with SPARQL query and getting desired information about entities.

In Figure 7 is possible to see the associations between DAO classes. Starting to *DataCenterDAO*, this class, whose methods are shown in Figure 8, can access only to *DataCenter* entity information or can access to information about *HostMachine* entities. The first access is a simple SPARQL query, whereas the second access is formed by the first simple SPARQL query followed by other queries that attempt to retrieve information about other entities and in this case about *HostMachines*. Analyzing methods present in each class is possible to understand because there is need of all the associations in Figure 7.

Each DAO class has similar methods to *DataCenterDAO*. In fact, in each class there is a method, such as *getURIDataCenterListByKB*, that allow retrieving a list of URI of desired entities present in the *“parent entity”*. For example, the *“parent entity”* of a *HostMachine* is the *DataCenter* and in the class *HostMachineDAO* there is the method *getURIHostMachineListbyDataCenter*.

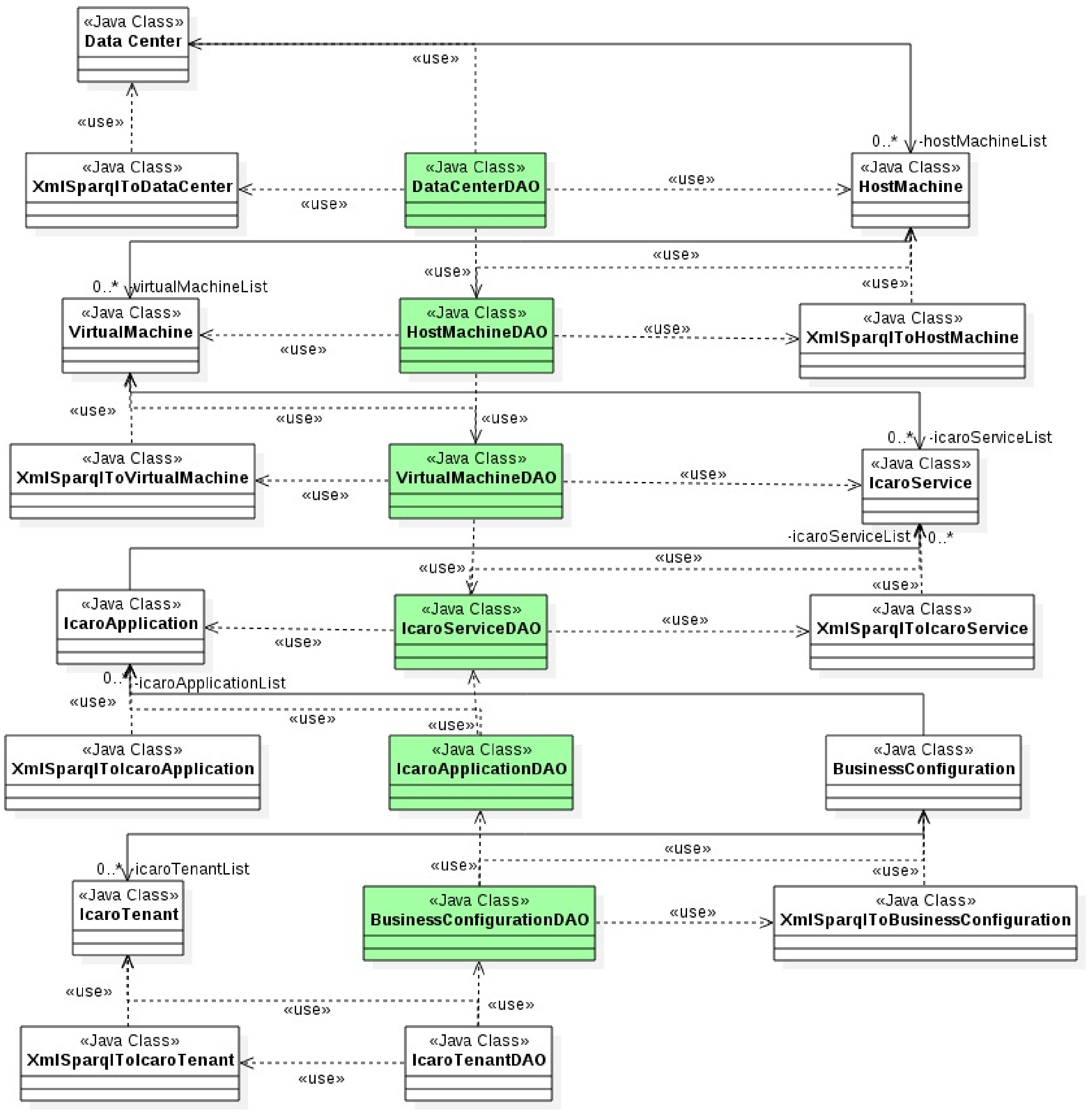

«Java Class» Data Center
«Java Class» XmlSparqlToDataCenter
«Java Class» DataCenterDAO
«Java Class» HostMachine
«Java Class» VirtualMachine
«Java Class» HostMachineDAO
«Java Class» XmlSparqlToHostMachine
«Java Class» XmlSparqlToVirtualMachine
«Java Class» VirtualMachineDAO
«Java Class» IcaroService
«Java Class» IcaroApplication
«Java Class» IcaroServiceDAO
«Java Class» XmlSparqlToIcaroService
«Java Class» XmlSparqlToIcaroApplication
«Java Class» IcaroApplicationDAO
«Java Class» BusinessConfiguration
«Java Class» IcaroTenant
«Java Class» BusinessConfigurationDAO
«Java Class» XmlSparqlToBusinessConfiguration
«Java Class» XmlSparqlToIcaroTenant
«Java Class» IcaroTenantDAO
«use»
0..* -hostMachineList
0..* -virtualMachineList
0..* -icaroServiceList
-icaroServiceList 0..*
0..* -icaroApplicationList
0..* -icaroTenantList

Figure 7 – DAO classes architecture

There is, also, a method to retrieve the list of objects, contained in a parent entities, instead that only the URI: for each entity are retrieved simple information as name and identifier and are not retrieved information about associated entities. So this method is useful when a user does not want to retrieve information about a single entity contained in the list that is returned, but he would like to retrieve only the information which is faster to retrieve.

The previous method uses another method present in DAO classes: the method that retrieves simple information about a single entity and returns an object instead a list of objects.

In the end, there is a method that starts with the prefix *fetch* and that retrieves all the possible information, present in the KB, associated with desired entity. This method can explain the visible associations in Figure 7, in fact, when the information about a DataCenter is requested, the class *DataCenterDAO* retrieve simple information about DataCenter entity and then it calls method to fetch information on class *HostMachineDAO*. In turn, *HostMachineDAO* retrieve simple information about *HostMachine* entity and then it calls method to fetch information on class *VirtualMachineDAO* and so on.

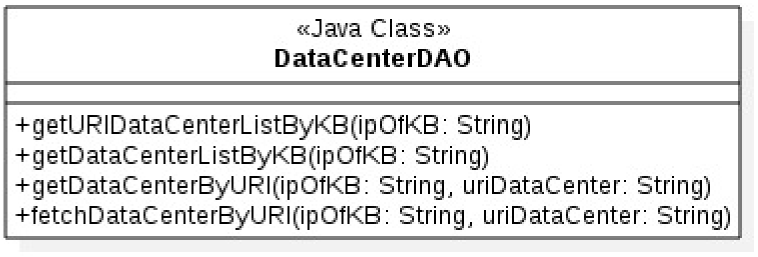

«Java Class»
DataCenterDAO
+getURIDataCenterListByKB(ipOfKB: String)
+getDataCenterListByKB(ipOfKB: String)
+getDataCenterByURI(ipOfKB: String, uriDataCenter: String)
+fetchDataCenterByURI(ipOfKB: String, uriDataCenter: String)

Figure 8 – Class DataCenerViewer and its methods

There are classes that have more/fewer than 4 methods.

Those that have more methods, such a *UserDAO* or *IcaroServiceDAO*, it is due to the structure of ontology, in fact, the entities *User* and *IcaroService* can be considered in more than one parent entities: User can be part of *BusinessConfiguration*, *IcaroApplication* or IcaroTenant and *IcaroService* can be part of *IcaroApplication* or *VirtualMachine*. In these classes there are two additional methods for each parent entity more first.

Those that have fewer methods, such a *MonitorInfoDAO* or *NetworkAdapterDAO*, it is due in the same way to the structure of ontology, in fact, the entities *MonitorInfo* and *NetworkAdapter* have not a visible URI and cannot be considered as stand-alone entities, rather these entities can be retrieved only knowing the parent entity that contains them. In these classes there are not the methods using the URI as parameter.

When a query response is returned it is in XML SPARQL form and it is necessary a converter utility to perform conversion from XML to object. In the system the converter utility is contained in the class whose name start with the prefix *XmlSparqlTo* and the rest indicates the entity converted by the class.

3.4 Simulator

The web application allows simulating the behavior of a dataCenter in two ways: a fast simulation and a real-time simulation. At the moment, the difference between the simulations is not merely on the time of the simulation but also on how the values are generated. In fact, in the real-time simulation the values are randomly generated at the level of IcaroService and they are added to make the workload of the virtualMachine where the IcaroServices run. In the fast simulation, in a different way, the values are replicated by the past values collected from real hostMachines present in a real dataCenter. These differences explain the architecture in

Figure 9.

The left part of the

Figure 9 represents the classes that perform the fast simulation: there are a central class DataCenterSimulationFaster that is responsible of the simulation, a NagiosServer class that contains information about the host where the simulated values will be sent and a DataCenter class that represents the data center that must be simulated.

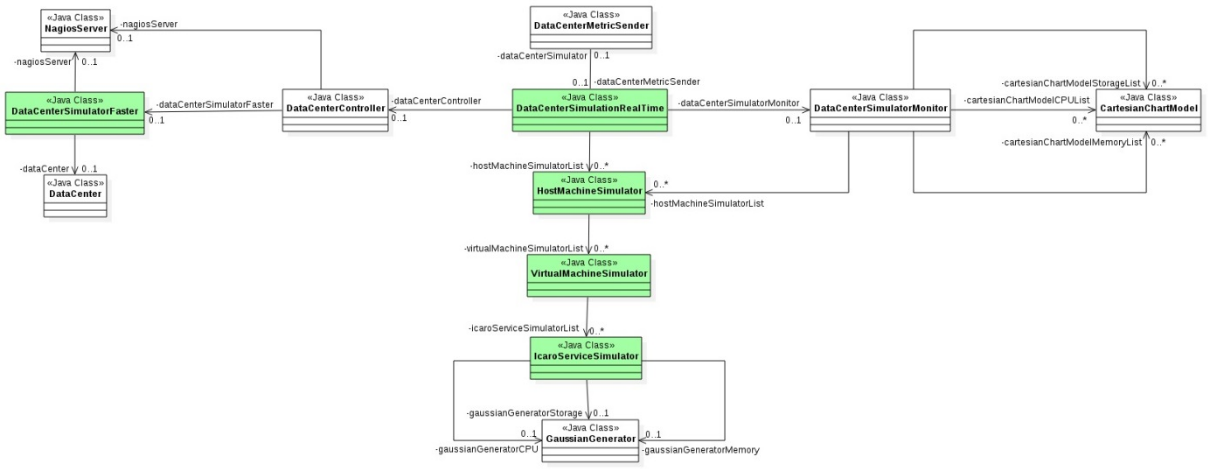

«Java Class» NagiosServer
-nagiosServer
0..1
-nagiosServer 0..1
«Java Class» DataCenterSimulatorFaster
-dataCenterSimulatorFaster
0..1
«Java Class» DataCenterController
-dataCenterController
0..1
-dataCenter 0..1
«Java Class» DataCenter
«Java Class» DataCenterMetricSender
-dataCenterSimulator 0..1
0..1 -dataCenterMetricSender
«Java Class» DataCenterSimulationRealTime
-dataCenterSimulatorMonitor
0..1
«Java Class» DataCenterSimulatorMonitor
-cartesianChartModelStorageList 0..*
-cartesianChartModelCPUList
0..*
«Java Class» CartesianChartModel
-cartesianChartModelMemoryList 0..*
-hostMachineSimulatorList 0..*
«Java Class» HostMachineSimulator
0..*
-hostMachineSimulatorList
-virtualMachineSimulatorList 0..*
«Java Class» VirtualMachineSimulator
-icaroServiceSimulatorList 0..*
«Java Class» IcaroServiceSimulator
-gaussianGeneratorStorage 0..1
0..1
-gaussianGeneratorCPU
«Java Class» GaussianGenerator
0..1
-gaussianGeneratorMemory

Figure 9 – Simulator Classes Architecture

The right part of the

Figure 9 represents the classes that perform the real-time simulation: there are a central class DataCenterSimulationRealTime that is responsible to start (they are runnable classes) one HostMachineSimulator for each host machine presents on the data center that must be simulated, a HostMachineSimulator class that is responsible to start one VirtualMachineSimulator for each virtual machine presents on the data center that must be simulated and so on until the IcaroServiceSimulator class that is responsible to randomly generate the workload values. Furthermore, there are a DataCenterSimulatorMonitor class that is responsible to update the CartesianChartModel, which class is associated to chart viewed from the user and a DataCenterMetricSender class, whose purpose is to send the metrics calculated on the generated values to the KB. In future the metrics must be sent to the NagiosServer as the metrics generated by the fast simulation.

4. Sequence Diagram

In this chapter are reported the most significant sequence diagrams, to show how the classes, described in the previous chapters, and the web pages interact with each other.

4.1 Composite Component in Icaro Cloud Simulator

In the JSF framework (Chapter 7) it is possible to create a Composite Component (CC) so that it is feasible to have a new desired tag representing an object that is frequently used in the developed system. In IcaroCloudSimulator there are CCs for the panels that are added on the web pages of creation of *DataCenter*, *BusinessConfiguration* and *VirtualMachine*. The CCs are organized in the folder *WebContent/resources/panelComponent* that contains one folder for each page that uses the CCs. In each of these folders there is one CC for every level of panels that must be in the three web pages of creation: in the *DataCenter* and *VirtualMachine* web page the maximum level of panel is two and therefore there are two CCs in the sub-folders dataCenter and virtualMachine and in the businessConfiguration web page the maximum level of panel is five and therefore there are five CCs in the sub-folder businessConfiguration.

To add a CC in a web page the simplest way is to insert the associated tag directly on the page, but this is a static insertion and in the IcaroCloudSimulator the insertion must be dynamic and it must leave to user the control on how many panels there are in a determined moment on the current web page. To do this, the CC must be inserted dynamically through the java classes and not directly on the xhtml pages and it can be done with insertion of information about: attributes to set in the CC and the filename of the file where the CC is saved. If a panel of first level is taken as example the attributes that can be inserted are:

- **idCC –** indicates the id to associate at the whole CC
- **indexPanel –** indicates the index that has the panel in the CC. It is necessary to remove the correct parent panel and to add o to remove a child panel.
- **titlePanel –** the title that is shown on the header of the panel
- **removeAction –** method that performs the removal of the whole CC
- **listInputIP –** list of inputText that allow inserting and checks address IP correctness
- **listInputDate –** list of inputText that allow inserting and checks date correctness
- **listInputEmail –** list of inputText that allow inserting and checks email correctness
- **listInputText –** list of inputText that allow inserting text
- **listInputNumber –** list of inputText that allow inserting number
- **listInputSelectOneMenu –** list of selectOneMenu that allow selecting a value from a menu
- **listInputSelectOneMenuDynamic –** list of selectOneMenu that allow selecting a value from a menu (difference to previous is how the menu is filled)
- **listDropDownMenu –** list of DropDownMenu that should allow the choice of entities to add at the entity associated with the current CC
- **objectLinkToPanel –** indicates the object that must be filled with the information inserted in the current panel

- **bindingToBody –** creates a binding between an object in the *Viewer* classes and the current CC to add new child

To avoid the creation of one method for each panel that can be inserted in the web page and to avoid method duplication due to possibility that one panel can be added to two or more different parent panel, the creation of CC is performed as described in section 5.2.

Note that *StaticAttributeRepository* and *DinamicAttributeRepository* classes are created to group all the possible values for the attributes in one single point: to maintain and to change them more efficiently.

```
public List<InputText> getGroupHostMachineListInputText() {
  return new ArrayList<InputText>(Arrays.asList(
    new InputTextHtml5("prefixName", "hasPrefixName", "Insert prefixName of host"),
    new InputTextHtml5("prefixIdentifier","hasPrefixIdentifier", "Insert prefixIdentifier of host"),
    new InputTextHtml5("CPUType", "hasCPUType", "Insert the type (Model) of each CPU in the host"),
    new InputTextHtml5("domain", "isInDomain", "Insert domain of host"),
    new InputTextHtml5("username", "hasAuthUserName", "Insert username of host"),
    new InputTextHtml5("password", "hasAuthUserPassword", "Insert password of host")));
    }
```

Figure 10 – Example of one method containing the values to fill the attributes of a CC

In Figure 10 it is possible to see an example of a method contained in the StaticAttributeRepository class. This method is responsible to create the list of inputText (that accept every type of text) that must be present in the panel of the entity *GroupHostMachine*. InputTextHtml5 is a class that is useful to fill the attributes of a CC representing a Bootstrap style InputText contained in the panel. For this class the first string is the label that must have the InputText, the second string is the name of the field contained in the object indicates by the *objectLinkToPanel* that must be filled with the value inserted in this InputText and the last string is the placeholder that is shown if nothing is inserted in the InputText.

4.2 Add a new panel to “Create a DataCenter” web page

The sequence diagram in Figure 11 is an example that represents how it is possible to add a new panel in an existing form (in this case a dataCenter form) and it shows how the methods of a viewer class work to perform this operation. Furthermore, the few operations, that must be performed each time that the user changes page, are shown.

The operation starts when the user click the link *Create a DataCenter* on the page *home.jsf* and he is redirected on the page *dataCenter.jsf*. This page at the generation of the event *preRenderView* calls two methods of the *Navigator* class: *beginConversation()* and *goToDataCenter()*.

The first method allows starting a new conversation to keep the beans (declared to be *ConversationScope* https://docs.oracle.com/javaee/7/tutorial/cdi-basic008.htm#GJBBK) alive until the conversation is not closed.

The second method simply sets a Boolean variable to *true* and a string variable to *Create a DataCenter*: these variables are used in the web page to show or to hide buttons and links based on which web page is shown at the user (Boolean variable) and to set the proper title on the navigation bar (String variable).

The new page is loaded and the user can choose in the drop down menu shown in Figure 23, what is the entity that he wants to add at the dataCenter. Assuming that the user click to add a HostMachine panel, system calls the *AddPanelGroupHostMachine* method on the class *DataCenterViewer*.

In this method there is a call to method *setFirstPanel* of *CCUtility* class that needs as parameters:

- **indexPanel –** The index of panel that the system will add at the end of this operation
- **panelDataCenterBody –** The parent panel where the new panel will be added
- **"groupHostMachine" –** The name of the entity associated to the new panel
- **"dataCenter" –** The name of the principal entity associated with the current web page

The *CCUtility* class is so called because it is useful to fill the attributes of the CCs and to add it in the current web page. In fact, once that the method *setFirstPanel* is called this creates a map that associates the name of an attribute contained in the CC to the value that the attribute must have. To fill the map, the method checks, on the class *StaticAttributesRepository*, the existence of some method containing the name of entity associated to the panel that must be added. The methods in *StaticAttributeRepository* class are named in such a way that it is possible to simply associate them to the attributes of the CCs (see Figure 10). If the method exists none exception is generated and the name of the method is associated with the correct attribute presents in the CC.

For example, in the stage of filling the attributes of the CC of *groupHostMachinePanel*, the method *setFirstPanel* (through the method *setDefaultAttributes* that in turn uses the method *setOneStaticAttribute*) checks if there is a method named *getGroupHostMachineListInputText* in the class *StaticAttributeRepository*: as it is possible to see in Figure 10 this method exists and its name is inserted in the map associated to the *listInputText* attribute.

In Figure 11 it is possible to see the interactions between the classes and the control operations performed: at first for the attribute *listDropDownMenu* then for all the attributes whose name starts with *listInput,* before those that are in the *StaticAttributesRepository* and after those that are in the *DynamicAttributesRepository*. Once the map is filled the method includeCompositeComponent is called and it is responsible to retrieve the file xhtml containing the CC, to fill the attributes in the CC with the values contained in the map and to add the CC created in the parent panel.

At the end of this chain of operations and calls the *DataCenterViewer* class checks if all panels, necessary to create a DataCenter XML file, are inserted in the web page with the method *controlForButtonCreateXML*and in any case the web page is refreshed to show the new panel to the user.

If the control succeeds then the necessary panels are inserted and the button *CreateXML* is enabled to allow the user to create the XML file with the inserted information. The operations to create the XML file are described in the next section.

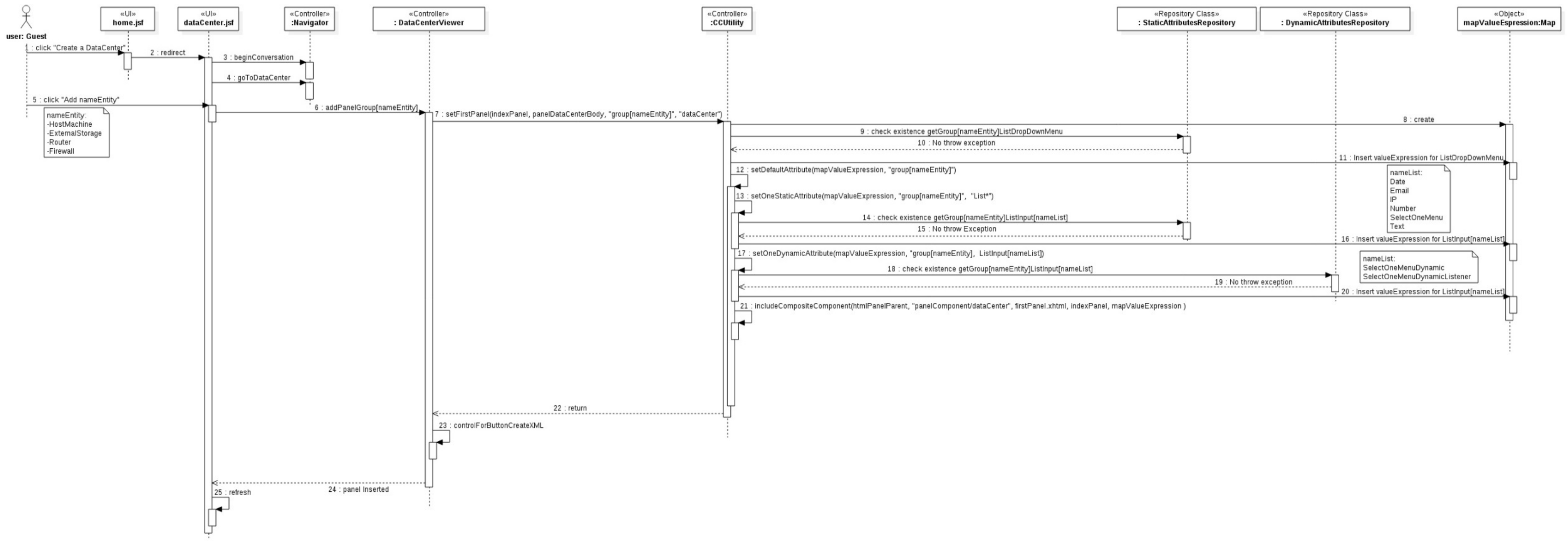

user: Guest
«UI» home.jsf
«UI» dataCenter.jsf
«Controller» :Navigator
«Controller» : DataCenterViewer
«Controller» :CCUtility
«Repository Class» : StaticAttributesRepository
«Repository Class» : DynamicAttributesRepository
«Object» mapValueEspression:Map
1 : click "Create a DataCenter"
2 : redirect
3 : beginConversation
4 : goToDataCenter
5 : click "Add nameEntity"
nameEntity: -HostMachine -ExternalStorage -Router -Firewall
6 : addPanelGroup[nameEntity]
7 : setFirstPanel(indexPanel, panelDataCenterBody, "group[nameEntity]", "dataCenter")
8 : create
9 : check existence getGroup[nameEntity]ListDropDownMenu
10 : No throw exception
11 : Insert valueExpression for ListDropDownMenu
nameList: Date Email IP Number SelectOneMenu Text
12 : setDefaultAttribute(mapValueExpression, "group[nameEntity]")
13 : setOneStaticAttribute(mapValueExpression, "group[nameEntity]", "List*")
14 : check existence getGroup[nameEntity]ListInput[nameList]
15 : No throw Exception
16 : Insert valueExpression for ListInput[nameList]
17 : setOneDynamicAttribute(mapValueExpression, "group[nameEntity], ListInput[nameList])
nameList: SelectOneMenuDynamic SelectOneMenuDynamicListener
18 : check existence getGroup[nameEntity]ListInput[nameList]
19 : No throw exception
20 : Insert valueExpression for ListInput[nameList]
21 : includeCompositeComponent(htmlPanelParent, "panelComponent/dataCenter", firstPanel.xhtml, indexPanel, mapValueExpression)
22 : return
23 : controlForButtonCreateXML
24 : panel Inserted
25 : refresh

Figure 11 – Sequence diagram that show how to add a panel to the dataCenter form

4.3 Create a New DataCenter XML File

In the Figure 12 there is the sequence diagram that explains the operations that must be performed to create correctly the XML file of a DataCenter.

As first operation the user must press the button *CreateXML* after he inserted the minimum information to enable the button (without this basic information, the XML file does not pass the control against the XML Schema).

When the button is pressed the system calls the *createDataCenterXML* method (in the *DataCenterController* class) which, in turn, check with the method *controlInputConsistency* if there is a possible inconsistency between a SharedStorage URI inserted in the form of a HostMachine group and the relative entity SharedStorage that must be inserted in the form of an ExternalStorage group. If there is an inconsistency, then a message is shown to the user to inform him of the error, else the method createDataCenterXML continues with the other operations.

The next operation is performed by the method *createDataCenter* which, with the information contained in the form filled by the user, creates the objects relative to the entities indicated in the form: if in the form the user has decided to have 1000 HostMachines then in this method 1000 objects of the class HostMachine are created and they are associated to the correct DataCenter object which is also created from this method.

These objects, which are created by the last operation, are reachable from the DataCenter object containing all of them and, for this reason, this last object is passed to the *createDataCenterXmlRdf* method that is responsible to create the XML RDF file: this method is in the utility class called XmlRdfCreator.

The method *createDataCenterXmlRdf* as first step creates the object *documentXmlRdf* to which will be appended elements created later: the element RDF is added at the beginning because it contains the root tag (<rdf:RDF>) and the namespace which will be used by the following elements. After the rdfElement are added the elements relative to the objects contained in the DataCenter object.

The creation of the element DataCenter is different to the other elements: the method *createDataCenterElement* of the *DataCenterToXmlRdf* class inserts in the element only the information that can be represented as a string; in this case the name, the identifier and the URI. The other information that requires more than one row in the DataCenter element, as the information about the HostMachine, the ExternalStorage, the Firewall and so on, is inserted with calls to *create[nameEntity]Element* methods contained in the *[nameEntity]ToXmlRdf* class where nameEntity can be: ExternalStorage, Firewall, Router, LocalNetwork and HostMachine. In each of these methods is created an element with the information that can be represented as a string and the information relative to other sub-entities contained in the considered entity: for example in the method *createHostMachineElement* of the class *HostMachineToXmlRdf* are inserted in the HostMachine tag: the information that can be represented as a string (CPU, Memory, username, etc…) and the information about entities as MonitorInfo and NetworkAdapter. To perform the insertion of these last entities, in the method *createHostMachineElement* are called the methods *createMonitorInfoElement* and *createNetworkAdapterElement* that perform insertion as the method *createHostMachineElement*.

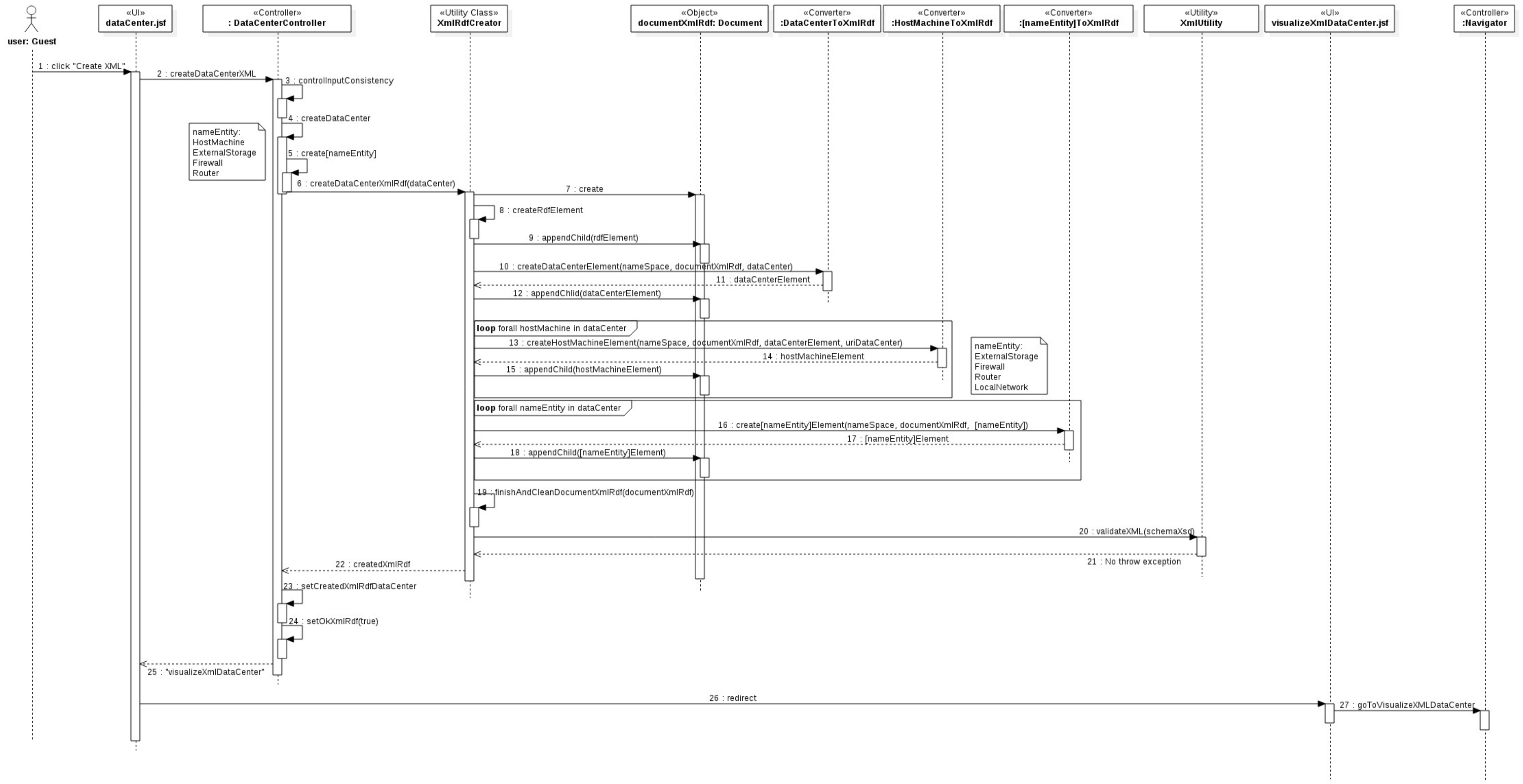

user: Guest
«UI» dataCenter.jsf
«Controller» : DataCenterController
«Utility Class» XmlRdfCreator
«Object» documentXmlRdf: Document
«Converter» :DataCenterToXmlRdf
«Converter» :HostMachineToXmlRdf
«Converter» :[nameEntity]ToXmlRdf
«Utility» XmlUtility
«UI» visualizeXmlDataCenter.jsf
«Controller» :Navigator
1 : click "Create XML"
2 : createDataCenterXML
3 : controlInputConsistency
4 : createDataCenter
nameEntity: HostMachine ExternalStorage Firewall Router
5 : create[nameEntity]
6 : createDataCenterXmlRdf(dataCenter)
7 : create
8 : createRdfElement
9 : appendChild(rdfElement)
10 : createDataCenterElement(nameSpace, documentXmlRdf, dataCenter)
11 : dataCenterElement
12 : appendChlid(dataCenterElement)
loop forall hostMachine in dataCenter
13 : createHostMachineElement(nameSpace, documentXmlRdf, dataCenterElement, uriDataCenter)
14 : hostMachineElement
15 : appendChild(hostMachineElement)
nameEntity: ExternalStorage Firewall Router LocalNetwork
loop forall nameEntity in dataCenter
16 : create[nameEntity]Element(nameSpace, documentXmlRdf, [nameEntity])
17 : [nameEntity]Element
18 : appendChild([nameEntity]Element)
19 : finishAndCleanDocumentXmlRdf(documentXmlRdf)
20 : validateXML(schemaXsd)
21 : No throw exception
22 : createdXmlRdf
23 : setCreatedXmlRdfDataCenter
24 : setOkXmlRdf(true)
25 : "visualizeXmlDataCenter"
26 : redirect
27 : goToVisualizeXMLDataCenter

Figure 12 – Sequence diagram that show how the XML file of a DataCenter is create

In the Figure 13 there is one example of the methods that create XML Element. In this case the code is cleaned from the code that is not necessary to explain what operations are performed by these methods. As mentioned there is a part in the method where is appended the information that can be represented as a string such as the name and the identifier of the considered entity and there is a part where are appended other entities calling the method associated to them, in this case the MonitorInfo entity. The order, of insertion of the information, is performed as indicated in the XML Schema, so it is possible have elements of different type alternated between them.

```
public static Element createFirewallElement(final Namespace nameSpace, final Document documentParent, final Firewall firewall) {

 Element firewallElement = documentParent.createElementNS(nameSpace.getURI(), nameSpace.getPrefix() + ":Firewall");
 firewallElement.setAttributeNS(NameSpaceRepository.RDF.getURI(), NameSpaceRepository.RDF.getPrefix() + ":about", firewall.getUri());
 firewallElement.appendChild(XmlUtility.createSimpleTextElement(nameSpace, documentParent, "hasName", firewall.getHasName()));
 firewallElement.appendChild(XmlUtility.createSimpleTextElement(nameSpace, documentParent, "hasIdentifier", firewall.getHasIdentifier()));
 …
 Element hasMonitorInfoElement = documentParent.createElementNS(nameSpace.getURI(), nameSpace.getPrefix() + ":hasMonitorInfo");
 for (MonitorInfo monitorInfo : firewall.getHasMonitorInfoList()) {
  if (monitorInfo != null) {
   hasMonitorInfoElement.appendChild(MonitorInfoToXmlRdf.createMonitorInfoElement(nameSpace, documentParent, monitorInfo));
  }
 }
 return firewallElement;
}
```

Figure 13 – Example of one method that converts an object to an XML element (the code is simplified)

It is called the method *finishAndCleanDocumentXmlRdf,* when all the objects are transformed in XML element and added to the XML RDF document. The method is responsible to add the header of the XML file (finish) and to escape some character written incorrectly and (unnecessary).

At the end of the previous operations the whole document is validated in the web application before to send it to the KB and because so it is possible to avoid the multiple requests that there would be validating the XML file directly by the KB. If the operation of validation does not throw any exception, then the string that represent the XML document is inserted in a determined variable, of the class DataCenterController, associated to the TextArea where the user is redirected. If an exception is generated, then the user is also redirect to the same page, but he will not see the TextArea, but a message that informs him on the error occurs: the variable that chooses from the two pages is the variable okXmlRdf in the DataCenterController.

4.4 Send the DataCenter XML File to KB

If the user has performed the operations described in the previous chapter, he should be in the web page *visualizeXMLDataCenter.jsf,* where it is possible to send the generated DataCenter XML file to the KB, so that, a new DataCenter entity is created in the ontology. It is possible to see the sequence diagram of this operation in the Figure 14.

To perform this new operation the user must press the button *SendToKB* that, as first step, will show a new form where the user can insert information, about the KB to which the XML file must be sent, as: IP address and port of the KB (It is supposed that every KB in different machine has the same RestAPI name and this information is not modifiable). The user confirms or changes the information that there is in the form and then he clicks the button *Send*: it will call the method *sendXmlRdfOfDataCenter* in the DataCenterController class.

The method hides the form where the user has just inserted the information and then it calls the method *sendXmlToKb* on the KBDAO class passing, as principal parameter, the string that represents the XML file to send the KB. This last method, in turn, calls the method *sendString* on the RestAPI class because, as mentioned, the XML file has been transformed into a string.

To insert the string in the KB it is necessary to perform an *httpRequest* of type *PUT* at the RestAPI exposed by the KB: the *httpResponse* that is returned from the *httpRequest* is used to fill the fields in an object of the class *responseMessageString*. This object is returned back through methods calls to DataCenterController class and according the values contained by the object, a success or a failure panel is shown to the user, to inform him of the result of the insertion.

4.5 Fetch information of a DataCenter

This section describes the operations, performed to fetch the information about a selected DataCenter, that are shown in the sequence diagram in Figure 16.

The whole sequence diagram starts when the system shows the user the web page of choice of the DataCenter: once that the user selects a DataCenter, this last is fetched from the KB with the information about each entity associated with the DataCenter.

On the *dataCenterChoice.jsf* page, is shown to the user an InputText containing the IP address and the port of the default KB, if these data are incorrect, the user can change them and then he can press the button *OK*. When the button is pressed, it calls the method *InvertIPInserted* that inverts the Boolean value, in the class DataCenterController, of a variable indicating if the IP address is inserted.

At the same time, the web page tries to retrieve the list of the DataCenters contained in the indicated KB with a call to the method *getDataCenterChoiceSelectOneMenu* in the class *DynamicAttributeRepository.* The web page will show information about the DataCenter, as the name and the identifier of each DataCenter, in the SelectOneMenu that it must create. To retrieve this information in addition to the URI, it is necessary that the method *getDataCenterChoiceSelectOneMenu* calls the method *getDataCenterListByKB*, whose purpose is to retrieve the list of the DataCenters contained in the KB with information that does not involve other entities: in this way is retrieved the necessary information without useless queries.

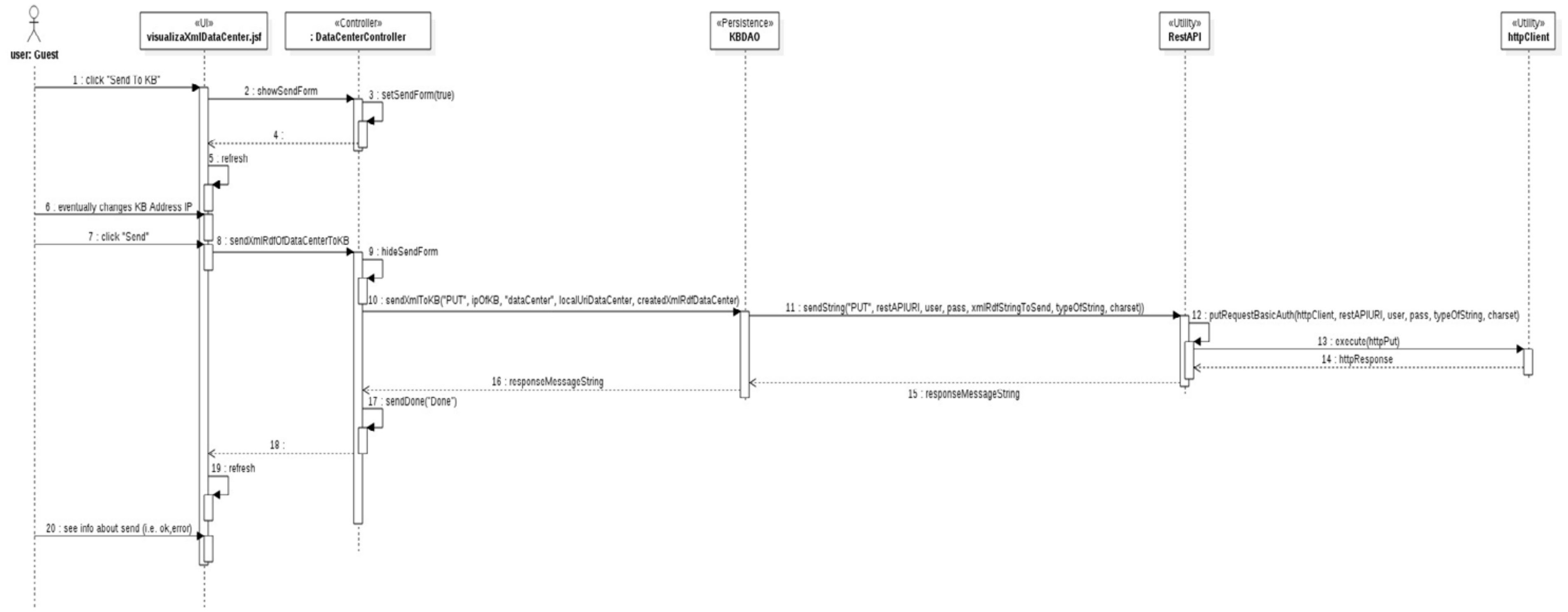

user: Guest
«UI» visualizaXmlDataCenter.jsf
«Controller» : DataCenterController
«Persistence» KBDAO
«Utility» RestAPI
«Utility» httpClient
1 : click "Send To KB"
2 : showSendForm
3 : setSendForm(true)
4 :
5 : refresh
6 : eventually changes KB Address IP
7 : click "Send"
8 : sendXmlRdfOfDataCenterToKB
9 : hideSendForm
10 : sendXmlToKB("PUT", ipOfKB, "dataCenter", localUriDataCenter, createdXmlRdfDataCenter)
11 : sendString("PUT", restAPIURI, user, pass, xmlRdfStringToSend, typeOfString, charset))
12 : putRequestBasicAuth(httpClient, restAPIURI, user, pass, typeOfString, charset)
13 : execute(httpPut)
14 : httpResponse
15 : responseMessageString
16 : responseMessageString
17 : sendDone("Done")
18 :
19 : refresh
20 : see info about send (i.e. ok,error)

Figure 14 – Sequence diagram that show how the XML file of a DataCenter is sent to the KB

Taking the information from the KB to create the list of the DataCenter it is performed as follows:

- The *getURIDataCenterListByKB* method is called to retrieve the list of the URI of each DataCenter contained in the indicated KB
- For all URI in the previously created list, it is called the method getDataCenterByURI that retrieves only simple information about the DataCenter (That does not involve other entities) as it is possible to see in Figure 15 (Only name and identifier are retrieved):
    - The query to send to the KB is created and it is executed calling the method queryKB on the RestAPI class
    - The query response is returned in XML SPARQL and the information contained is parsed to create a DataCenter object with the method *createDataCenter* in the class XmlSparqlToDataCenter
- Each retrieved DataCenter is inserted in a list and it is returned by the *getDataCenterListByKB* method

The *getDataCenterChoiceSelectOneMenu* method uses the list of the DataCenter to create a menu where the user can choose one of the DataCenter contained in the KB: once that he chooses the DataCenter, he must press the button *Load* to retrieve the chosen DataCenter from the KB and view the entities contained in it.

```java
public DataCenter getDataCenterByURI(final String ipAddressOfKB, final String uriDataCenter) {
   String select = "select ?hasName ?hasIdentifier where { <"
                + uriDataCenter + "> rdf:type icr:DataCenter . <"
                + uriDataCenter + "> icr:hasName ?hasName . <"
                + uriDataCenter + "> icr:hasIdentifier ?hasIdentifier}";

   String result = executeQuery(ipAddressOfKB, select);

   return XmlSparqlToDataCenter.createDataCenter(result, uriDataCenter);
}

public DataCenter fetchDataCenterByURI(final String ipAddressOfKB, final String, uriDataCenter) {

   DataCenter dataCenter = getDataCenterByURI(ipAddressOfKB, uriDataCenter);

   HostMachineDAO hostMachineDAO = new HostMachineDAO();
   for (String uriHostMachine : hostMachineDAO.getURIHostMachineListByDataCenter(ipAddressOfKB,
uriDataCenter)) {
      dataCenter.getHostMachineList().add(hostMachineDAO.fetchHostMachineByURI(ipAddressOfKB,
uriHostMachine));
   }

   LocalNetworkDAO localNetworkDAO = new LocalNetworkDAO();
   for (String uriLocalNetwork : localNetworkDAO.getURILocalNetworkListByDataCenter(ipAddressOfKB,
uriDataCenter)) {
     dataCenter.getLocalNetworkList().add(localNetworkDAO.fetchLocalNetworkWithUsedIPByURI(ipAddressOfKB,
uriLocalNetwork));
   }

   return dataCenter;
}
```

Figure 15 – Comparison between "get" and "fetch" methods to retrieve a DataCenter by the URI

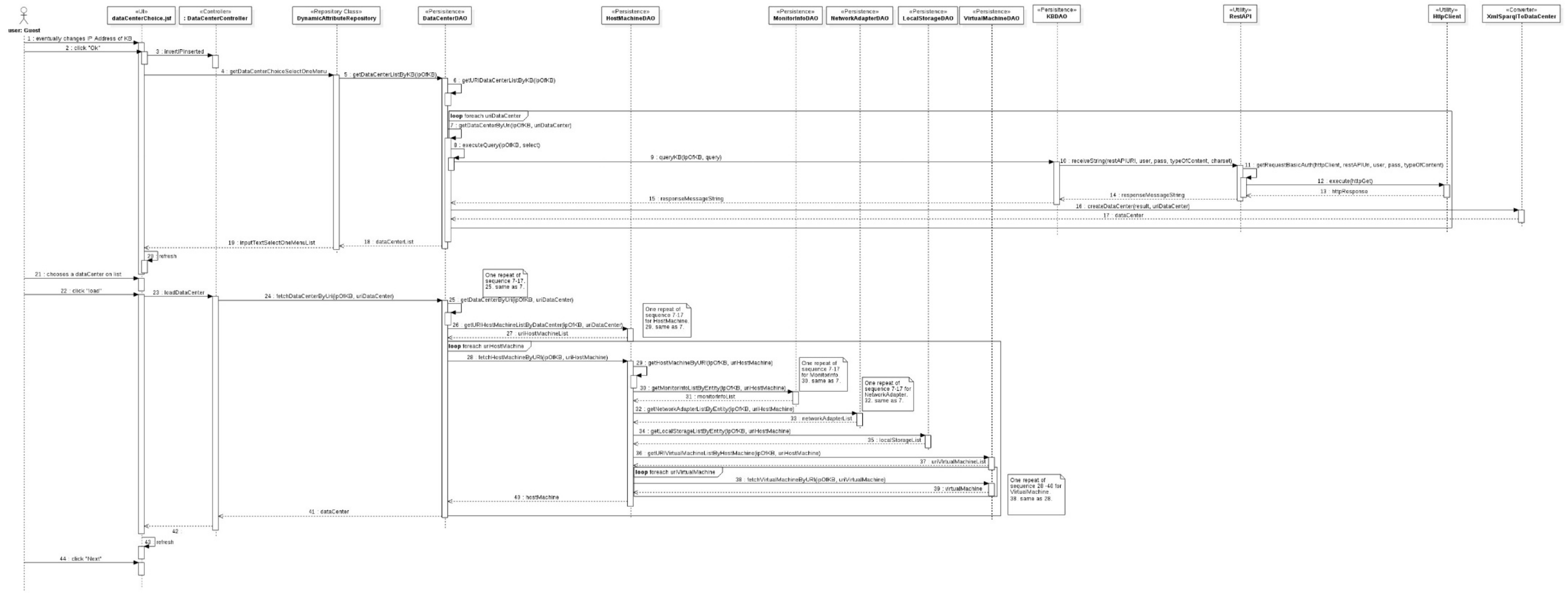

user: Guest
«UI» dataCenterChoice.jsf
«Controller» : DataCenterController
«Repository Class» DynamicAttributeRepository
«Persisitence» DataCenterDAO
«Persistence» HostMachineDAO
«Persistence» MonitorInfoDAO
«Persistence» NetworkAdapterDAO
«Persistence» LocalStorageDAO
«Persistence» VirtualMachineDAO
«Persistence» KBDAO
«Utility» RestAPI
«Utility» HttpClient
«Converter» XmlSparqlToDataCenter
1 : eventually changes IP Address of KB
2 : click "Ok"
3 : invertIPinserted
4 : getDataCenterChoiceSelectOneMenu
5 : getDataCenterListByKB(ipOfKB)
6 : getURIDataCenterListByKB(ipOfKB)
loop foreach uriDataCenter
7 : getDataCenterByUri(ipOfKB, uriDataCenter)
8 : executeQuery(ipOfKB, select)
9 : queryKB(ipOfKB, query)
10 : receiveString(restAPIURI, user, pass, typeOfContent, charset)
11 : getRequestBasicAuth(httpClient, restAPIUri, user, pass, typeOfContent)
12 : execute(httpGet)
13 : httpResponse
14 : responseMessageString
15 : responseMessageString
16 : createDataCenter(result, uriDataCenter)
17 : dataCenter
18 : dataCenterList
19 : inputTextSelectOneMenuList
20 : refresh
21 : chooses a dataCenter on list
22 : click "load"
23 : loadDataCenter
24 : fetchDataCenterByUri(ipOfKB, uriDataCenter)
One repeat of sequence 7-17. 25. same as 7.
25 : getDataCenterByUri(ipOfKB, uriDataCenter)
26 : getURIHostMachineListByDataCenter(ipOfKB, uriDataCenter)
27 : uriHostMachineList
One repeat of sequence 7-17 for HostMachine. 29. same as 7.
loop foreach uriHostMachine
28 : fetchHostMachineByURI(ipOfKB, uriHostMachine)
29 : getHostMachineByURI(ipOfKB, uriHostMachine)
One repeat of sequence 7-17 for MonitorInfo. 30. same as 7.
30 : getMonitorInfoListByEntity(ipOfKB, uriHostMachine)
31 : monitorInfoList
One repeat of sequence 7-17 for NetworkAdapter. 32. same as 7.
32 : getNetworkAdapterListByEntity(ipOfKB, uriHostMachine)
33 : networkAdapterList
34 : getLocalStorageListByEntity(ipOfKB, uriHostMachine)
35 : localStorageList
36 : getURIVirtualMachineListByHostMachine(ipOfKB, uriHostMachine)
37 : uriVirtualMachineList
loop foreach uriVirtualMachine
38 : fetchVirtualMachineByURI(ipOfKB, uriVirtualMachine)
39 : virtualMachine
One repeat of sequence 28 -40 for VirtualMachine. 38. same as 28.
40 : hostMachine
41 : dataCenter
42 :
43 : refresh
44 : click "Next"

Figure 16 – Sequence diagram that show how one DataCenter is fetched from the KB

The *Load* button calls the *LoadDataCenter* method in the *DataCenterController* class that, in turn, call the *fetchDataCenterByURI* method (Figure 15) in the *DataCenterDAO* class, which, at the first step, retrieves the basic information of the DataCenter, as mentioned above, and then it tries to retrieve information about the associated entities. To retrieve information of the associated entities, such as HostMachine, the *fetch* method, in the DataCenterDAO class, calls the method that returns the URI list of the entities of type that is considered: for all URI in this list it is called the *fetch* in the respective DAO of the type that is considered. Each of these *fetch* methods work as described for the *fetchDataCenterByURI* method in a manner that retrieve all the entities associated with the principal entity, in this case a DataCenter, and other entities associated with the second entities and so on. In case of entities that do not have any entity associated such as, MonitorInfo, NetworkAdapter and LocalStorage, only the method *get* is called as it possible to see in Figure 15 and it can also see the difference between these entities and, for example the VirtualMachine entity.

When the DataCenter is loaded with all the associated entities, it is shown at the user that can choose another DataCenter or he can press the *Next* button to pass to another operation which will be performed in the chosen DataCenter.

5. Simulation

The classes that perform the simulation are described in the section 4.4 and in this chapter will be detailed the operations that must be performed to simulate the chosen DataCenter in the two kinds of simulation offers by Icaro Cloud Simulator.

5.1 Real-time simulation

In this type of simulation the values are generated in real-time, so five minutes of the simulation correspond to five minutes of metrics that it is possible to insert in the KB. At the moment to realize this simulation, one thread is created for each entity involved in the simulation: i.e. each thread for each HostMachine, each VirtualMachine and each IcaroService. Missing the models to represent the workload of the entities (It is a future work), the mechanism that, at the moment, allows the simulation is the following that it is possible to see in Figure 17:

- When the user presses the button *Start,* the system calls the method *startSimulation*() of the class *DataCenterSimulationRealTime*

- This method, for each HostMachine contained in the chosen DataCenter, starts a thread based on the HostMachineSimulator class

- Each HostMachineSimulator thread, for each VirtualMachine contained in the HostMachine associated to this thread, starts a thread based on the VirtualMachineSimulator class

- In turn each VirtualMachineSimulator thread, for each IcaroService run in the VirtualMachine associated to this thread, starts a thread based on the IcaroServiceSimulator class

This last class is responsible to generate the workload for each IcaroServiceSimulator thread: every second it updates the three variables that represents the CPU, the memory and the storage workloads (respectively in Mhz, GB and in GB) with random values generate by three Gaussian Generators (In the future works it will be replaced by another generator). At the moment the simulator does not consider the workload caused by the OS or the hypervisor.

Each VirtualMachineSimulator thread every two seconds updates its three variables, which represent the workload, adding the workload generated by the IcaroServiceSimulators associated to IcaroServices that run in the VirtualMachine associated to the considered VirtualMachineSimulator. In a similar way, each HostMachineSimulator thread every four seconds updates its three variables adding the workload generated by the VirtualMachineSimulators associated to VirtualMachines contained in the HostMachine associated to the considered HostMachineSimulator.

Returning to the previous operations, the method *startSimulation()* in the *DataCenterSimulationRealTime* class, in addition to start the HostMachineSimulator threads it also starts another thread based on the class DataCenterSimulatorMonitor: with this class the system can show charts for all the workloads calculated by each HostMachineSimulator.

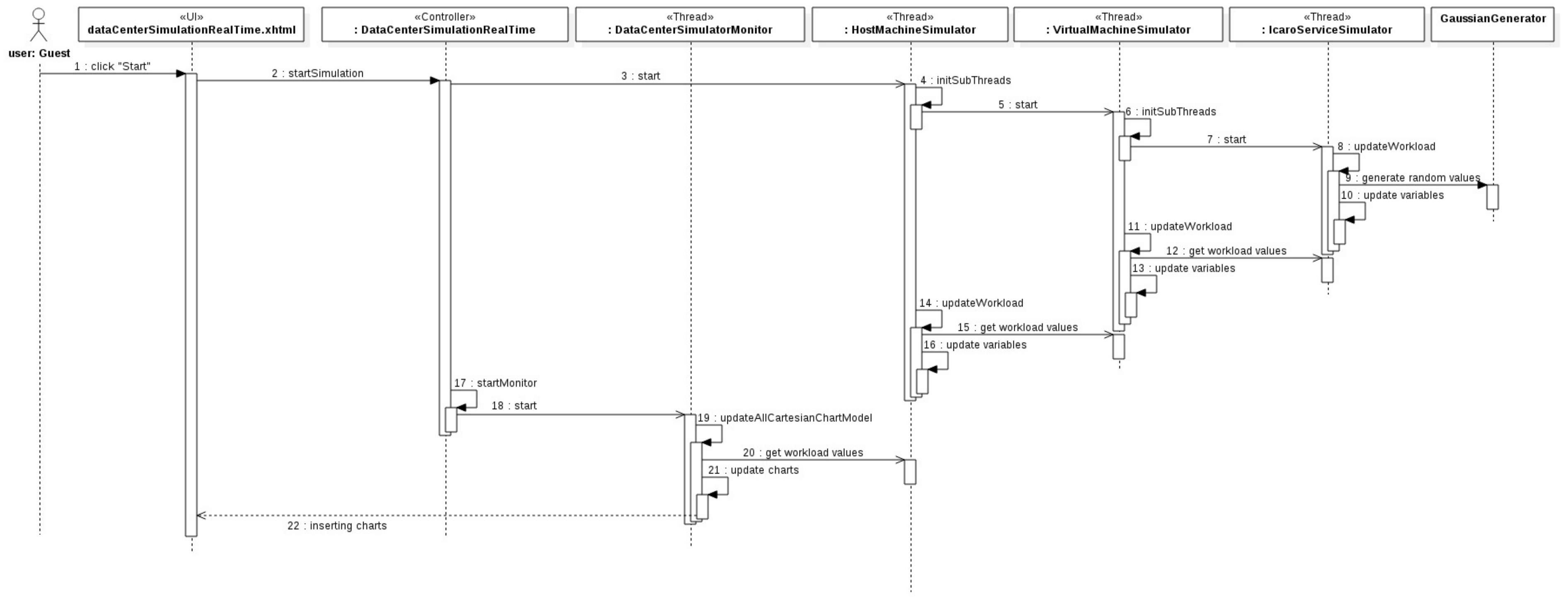


Figure 17 – Sequence diagram that shows how is performed the real-time simulation

The refresh time is chosen by the user, but knowing that a HostMachineSimulator updates its workload each four seconds it does not make sense to go below this threshold. DataCenterSimulatorMonitor thread can extract the values of the HostMachine workload because when it was started the DataCenterSimulationRealTime class passes to it the list of the HostMachineSimulator threads started

Furthermore, the user can decide in any moment to send metric of the generated workload to the KB, inserting the KB address (IP and Port) and the sampling period (In the future works these metrics will be sent to the Nagios Server as RRD files (Round Robin Database)) and pressing the button *SendToKB*. The system, when the button is clicked, calls the method *startSendToKB()* in the *DataCenterSimulationRealTime* class that starts one thread based on the *DataCenterMetricSender* class. This thread, at the period indicated by the user, samples the workload values from all HostMachineSimulator and VirtualMachineSimulator running threads, creates the XML file with all the necessary information needed to the KB to insert correctly the metrics. To improve the performances, before are sampled the metrics of all the resources simulated and after only one XML file are sent to the KB.

5.2 Fast simulation

In this type of simulation the values are generated as quickly as possible depending on the machine where the IcaroCloudSimulator is executed. The values, in this case, are generated from real data that are collected from various machine (Host and Virtual) of a real DataCenter with the periods of one day, seven days and 30 days and that are saved in XML files.

The user can see and associate a real pattern to one VirtualMachine, whose workload must be simulated: to associate a pattern to a VirtualMachine means that all the values simulated for the VirtualMachine for all resources as CPU, disk and storage are taken from the XML collected at the same period from the same VirtualMachine, so that it is possible to maintain the correlation between the workloads of different resources. If the user does not associate any pattern to one or more VirtualMachines, then the patterns are randomly chosen between those collected.

When the user clicks the button start, the system calls the method *simulate*() in the *DataCenterController* class that initializes one thread based on the class DataCenterSimulatorFaster (see Figure 18). The thread, for each VirtualMachine contained in the DataCenter, takes the values from XML file containing the pattern associated, by the user or randomly, to the VirtualMachine that must be simulated and with these values it creates a RRD file so that it can be read by the NAGIOS server. The values taken from the XML file are added to create the workloads of the HostMachines and to create, also in this case, the RRD files.

If the simulation period chosen by the user is less than the pattern period then only the first values are taken to simulation else, the pattern is replicated a number of times necessary to cover the entire simulation period.

In this type of simulation the main problem is how to write the RRD files, in fact this type of file is dependent of the file system and a RRD file written in Windows cannot be read in Linux.

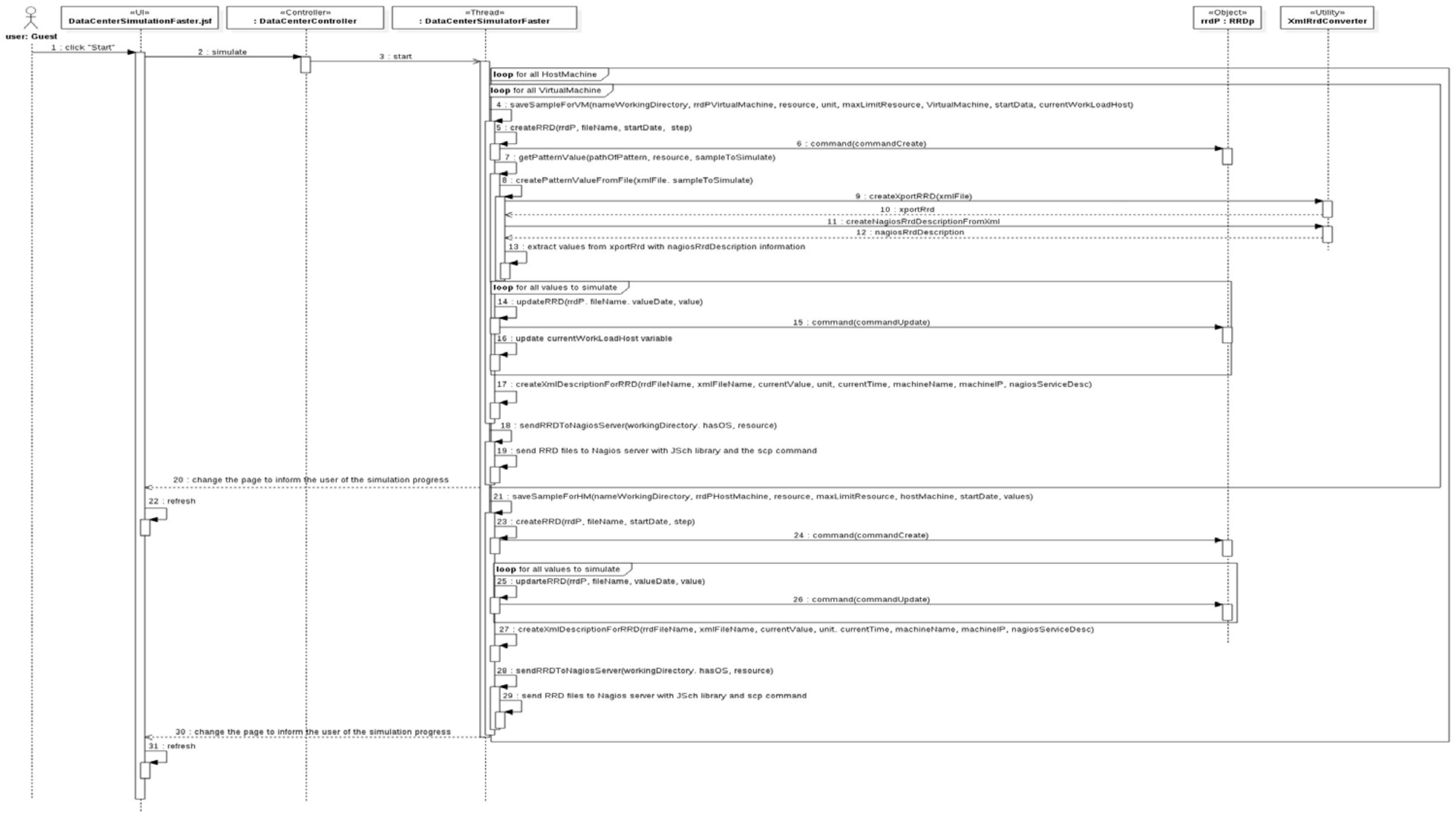

user: Guest
«UI»
DataCenterSimulationFaster.jsf
«Controller»
: DataCenterController
«Thread»
: DataCenterSimulatorFaster
«Object»
rrdP : RRDp
«Utility»
XmlRrdConverter
1 : click "Start"
2 : simulate
3 : start
loop for all HostMachine
loop for all VirtualMachine
4 : saveSampleForVM(nameWorkingDirectory, rrdPVirtualMachine, resource, unit, maxLimitResource, VirtualMachine, startData, currentWorkLoadHost)
5 : createRRD(rrdP, fileName, startDate, step)
6 : command(commandCreate)
7 : getPatternValue(pathOfPattern, resource, sampleToSimulate)
8 : createPatternValueFromFile(xmlFile. sampleToSimulate)
9 : createXportRRD(xmlFile)
10 : xportRrd
11 : createNagiosRrdDescriptionFromXml
12 : nagiosRrdDescription
13 : extract values from xportRrd with nagiosRrdDescription information
loop for all values to simulate
14 : updateRRD(rrdP. fileName. valueDate, value)
15 : command(commandUpdate)
16 : update currentWorkLoadHost variable
17 : createXmlDescriptionForRRD(rrdFileName, xmlFileName, currentValue, unit, currentTime, machineName, machineIP, nagiosServiceDesc)
18 : sendRRDToNagiosServer(workingDirectory. hasOS, resource)
19 : send RRD files to Nagios server with JSch library and the scp command
20 : change the page to inform the user of the simulation progress
21 : saveSampleForHM(nameWorkingDirectory, rrdPHostMachine, resource, maxLimitResource, hostMachine, startDate, values)
22 : refresh
23 : createRRD(rrdP, fileName, startDate, step)
24 : command(commandCreate)
loop for all values to simulate
25 : updarteRRD(rrdP, fileName, valueDate, value)
26 : command(commandUpdate)
27 : createXmlDescriptionForRRD(rrdFileName, xmlFileName, currentValue, unit. currentTime, machineName, machineIP, nagiosServiceDesc)
28 : sendRRDToNagiosServer(workingDirectory. hasOS, resource)
29 : send RRD files to Nagios server with JSch library and scp command
30 : change the page to inform the user of the simulation progress
31 : refresh

Figure 18 - Sequence diagram that shows how is performed the fast simulation

To avoid this problem it is possible to use rrd4j library (https://code.google.com/p/rrd4j/), but there is the problem that the files written with this library must be read with the same library and Nagios does not use this library. So in IcaroCloudSimulator is used the library java-rrd that manages the RRD files with command-line operations through rrdtool. It is possible a workaround to read RRD files written in Windows by Nagios: dumping the RRD files to XML, sending these files to Nagios and dumping the XML back to RRD files in the server. The main problem of this solution is the increased dimension of the XML file dumped respect to the original RRD file. In the end the best solution is to run IcaroCloudSimulator in a Linux machine.

To send the RRD file and the description XML file (each RRD file must have one XML associated that contains meta-information about the RRD) in the Nagios server, IcaroCloudSimulator used the JSch library (http://www.jcraft.com/jsch/) that allows SSH connection to another host and allows execution of command through this secure connection: it is possible to copy the file created locally to the Nagios server with the scp command.

Once sent the RRD files to Nagios it is possible to see the charts created by the values contained in these files with the “Supervisor & Monitor” tool.

6. JSF Framework

Java Server Faces (https://docs.oracle.com/javaee/7/tutorial/jsf-intro.htm#BNAPH) is a framework for the realization of web applications; it provides to programmer tools for the realization of user interfaces and for the management of the navigation flow of the application: in the first case it offers a set of default component ready to use and in the second case using an event logic.

It is composed, basically, by:

- A set of API to represent and to manage the state of the interface components, to handle the events generated by these components and to validate data inserted by the user.
- A specific *tag library* to insert in the web pages the interface components

The management logic of the application is delegated to the *Backing Bean*: specific *annotated* Java classes that presents a series of data member associated to the various interface components containing dynamic information to represent on the page (or on the pages) to manage or accountable to receive information that is inserted by the user through the associated component (i.e. a form).

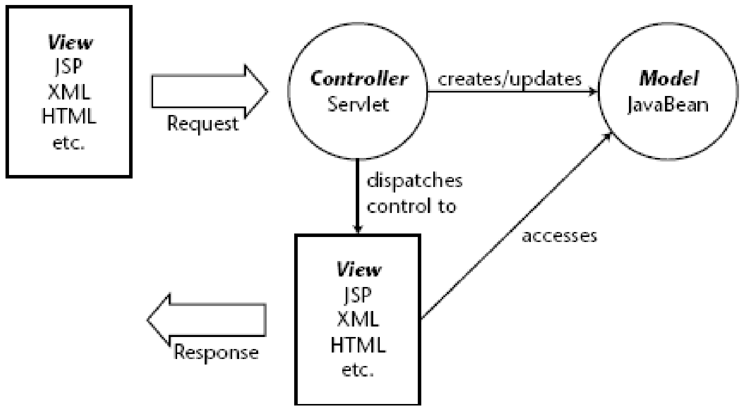


Figure 19 – Model-2 structure of a typical J2EE application

Furthermore, specific methods can be present to be invoked in response to an event generated by one of the element of the interface, for example through a selection of a link or the click on a button.

The JSF applications are structured according to MVC paradigm, in particular according to the so-called Model-2 variant. A generic Model-2 application requires the presence of a single centralized *controller* that receives all requests submitted by the user (via the web browser), interprets them, interacts with the data model and determines to which view pass the control and if the generated output is correct; each view can, if it is necessary, interact with the model to retrieve the necessary data.

In typical Model-2 applications, made with platform J2EE, the role of the controller is delegated to a servlet, the model consists of normal Java classes and JavaBean (In addition, for example, to any classes dedicated to interaction with the Database) and the views are realized through JSF pages, as shown in the diagram in Figure 19. In a JSF application the views are represented by the *tree components* associated to each page, consisting of the components used in the same page; a specific servlet, called *Faces Servlet*, plays a role of *front controller:* it receives the client requests and it pass the control to the correct view, updating the values of the bean properties that have changed due to interaction of the user or the user are inserted some data.

The servlet is responsible to invoke the validators of the various components, in order to control the correctness of information inserted by the user, and to execute the *ActionListeners* associated to components that have generated some kind of *event*.

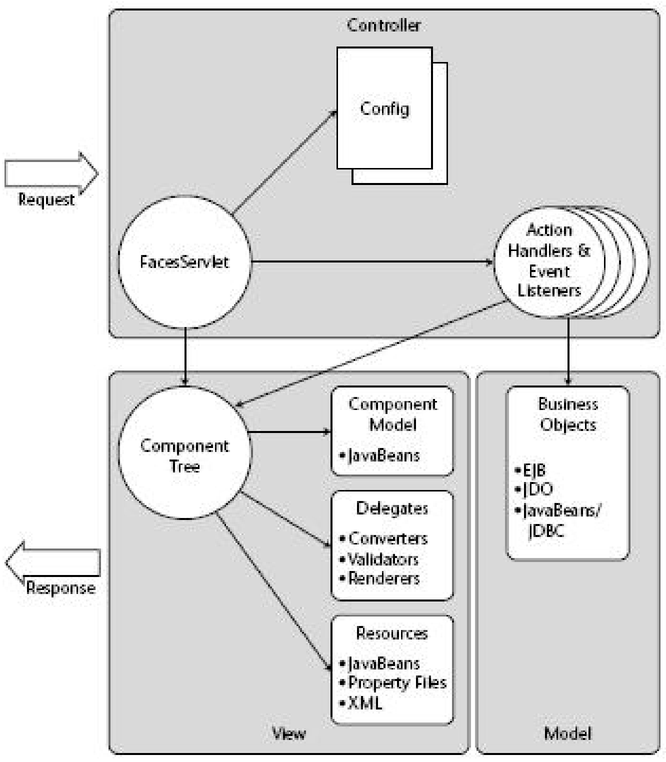


Figure 20 – Model-2 structure with JSF

The controller role is therefore subdivided between Faces Servlet and various listener, as it is shown in Figure 20. Each view can interact with the model through the methods of its *backing bean,* to which are delegated all the operations of data retrieval and data handling. The *bean,* therefore, plays a role of *mediator* between the view and the application model, with functionality that can be seen in the middle between a viewer and a controller.

6.1 Facelets

For the definition of the structure of the pages of a Web application can be used several templating languages: JSF provides the standard JSP, however, can also be used XML-based languages, such as Facelets (https://docs.oracle.com/javaee/7/tutorial/jsf-facelets.htm#GIEPX). The life cycle of Java Server Faces is composed of several phases and, unlike JSP, which processes the elements of a page in the order in which these appear, JSF organizes them in a complex tree structure. These differences lead to a series of incompatibility linked to the fact that the components operated by JSP are displayed in order of appearance within while those of the page JSF as defined by the phase

RenderResponse. Facelets is created to replace the JSP page creation, overcoming such incompatibilities. With Facelets, using the tools offered by JSF, it is possible to create custom interface components and place them within the pages calling them with the associated tags. These components can in turn be combined to create new ones, or can be collected in real libraries, reusable in other applications.

7. User guide

In this chapter is described how to use Icaro Cloud Simulator.

7.1 Home

When a user access to the web application that realize Icaro Cloud Simulator, he can see a blank page with a navigation bar where he can choose what operation he wants to perform between those present (Figure 21).

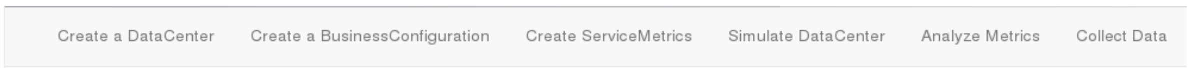
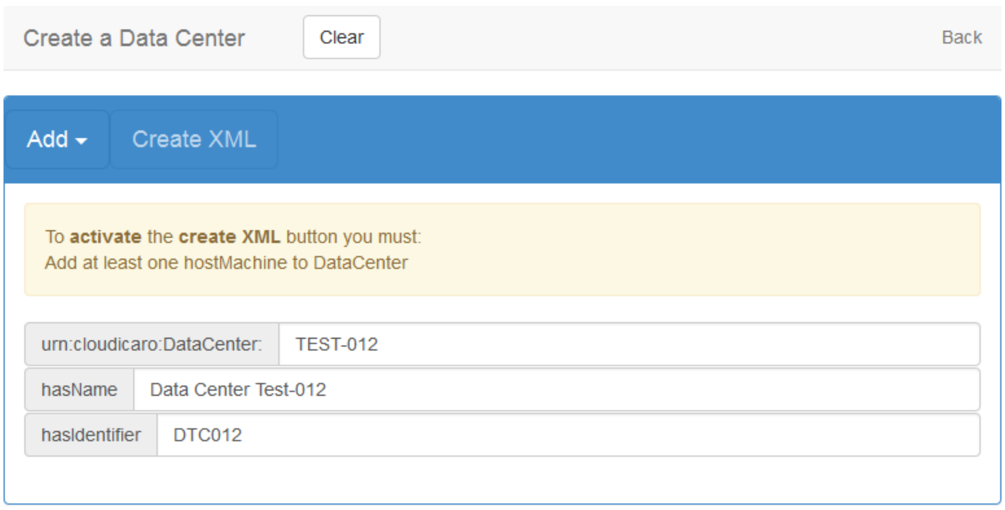


Figure 21 – Operations that is possible to execute with the Icaro Cloud Simulator

7.2 Create a DataCenter



To **activate** the **create XML** button you must:
Add at least one hostMachine to DataCenter

urn:cloudicaro:DataCenter:	TEST-012
hasName	Data Center Test-012
hasIdentifier	DTC012

Figure 22 – Principal page of the dataCenter creation

The principal part of each cloud platform is the data center and it is necessary to have a tool that allows insertion in the KB of a new *DataCenter* which is possible to analyze. This operation can be performed using the link "*Create a DataCenter*" present on the start navigation bar. Once click on the operation link the user accesses to the web page shown in Figure 22

With the form of Figure 22 is possible to insert the following information about the desired dataCenter:

- **urn:cloudIcaro:DataCenter:** the KB is realized through an ontology and in this kind of database each entity is unequivocally identified by URI. With regard to *DataCenter* the URI prefix is always the same, but the final part is variable and it can be chosen by the user, inserting it in this field.

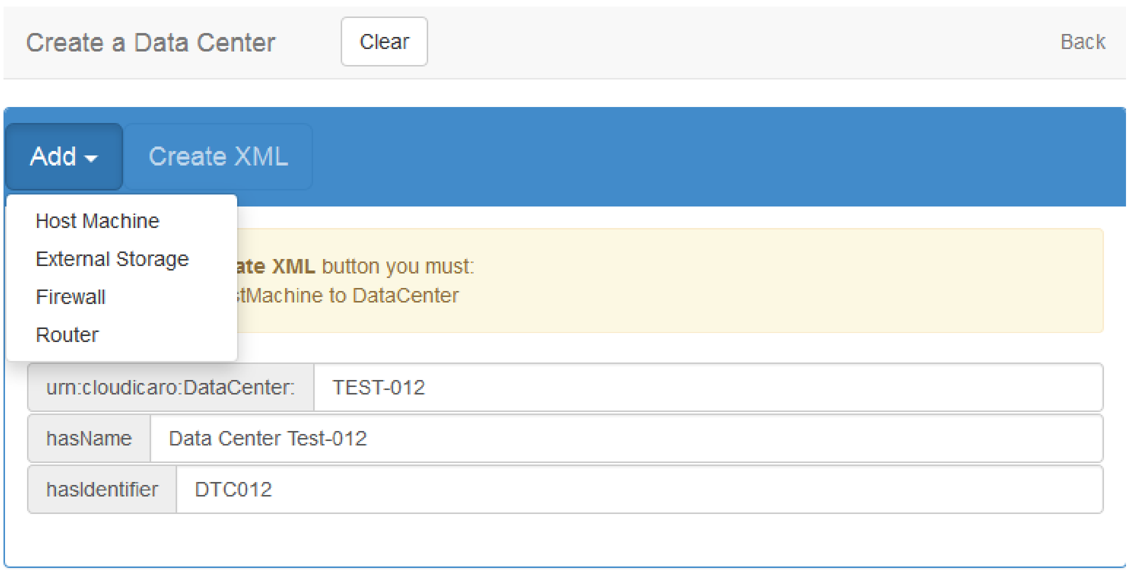
- **hasName:** in this field is possible to insert the name that the user want to assign to the data center that is being created
- **hasIdentifier:** it is possible to associate an identifier to data center and user can do it using this field

As it is possible to see in Figure 22 this information are not sufficient to create a *DataCenter*: the yellow alert indicates other entities that, necessarily, must be inserted to create a representative XML file of just created *DataCenter*.

Clicking on button *Add* is possible to add further entities, optional or mandatory to file creation (Figure 23), to *DataCenter*.

The button *Clear*, which is positioned at the top on the navigation bar in Figure 23, allows resetting all filled forms: once pressed will not be possible to recover the data inserted.

The button *Back*, which is positioned at the top right on the navigation bar in Figure 23, allows returning at Home to choose another operation to execute with the web application.



ate XML button you must:
tMachine to DataCenter

urn:cloudicaro:DataCenter:	TEST-012
hasName	Data Center Test-012
hasIdentifier	DTC012

Figure 23 – Entity that is possible to add in a dataCenter

From this point is shown how is possible to add entities to *DataCenter*, starting from *HostMachine* entity that it is mandatory to the creation of XML file.

7.2.1 Host Machine

To speed insertion of information, the form of the creation of HostMachine entities (Figure 24), allows inserting groups of hostMachine, instead to insert of each single HostMachine.

Unlike inserted information about the dataCenter, in this case is not necessary the insertion of the HostMachine URI: being a multiple insertion the URI is generated automatically and it is based on: the data center where are situated the host machines, the number of the group which is created and the number of host machines present in the group. For example, if a group of *HostMachine* is generated with information inserted on the form in Figure 24, in a *DataCenter* generated with information inserted on the form in Figure 23, will be generated the following URI:

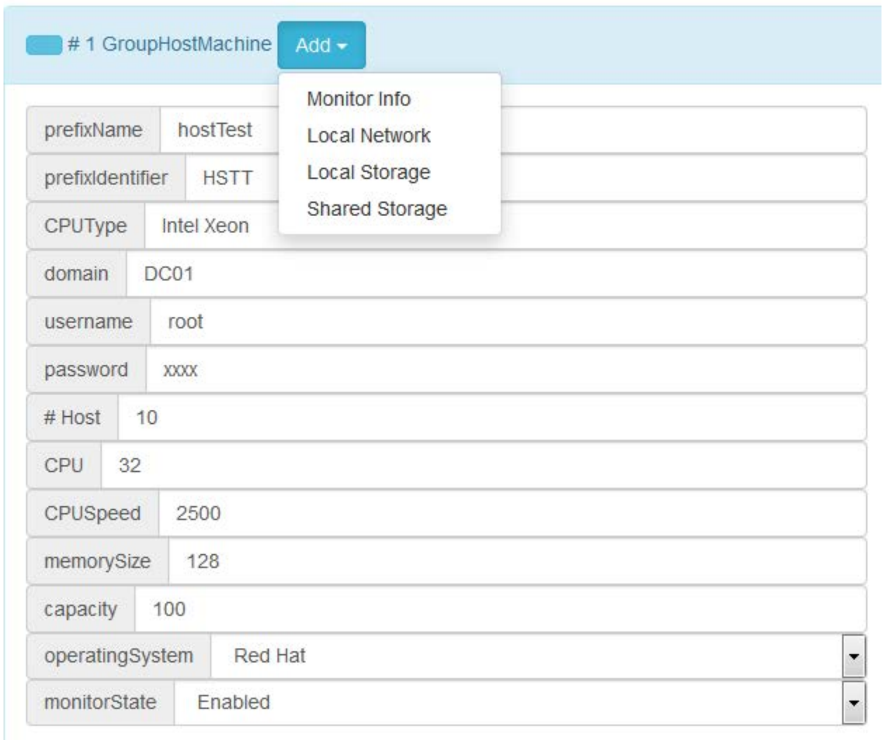
urn:cloudicaro:HostMachine:Test-012_HM01

urn:cloudicaro:HostMachine:Test-012_HM02

urn:cloudicaro:HostMachine:Test-012_HM010

Other fields must be filled with following information:

- **prefixName:** prefix name of all *HostMachine* that will be created by this form. At the prefix, as do for the URI, will be added a final number indicating the *HostMachine* in the group
- **prefixIdentifier:** prefix identifier of all *HostMachine* that will be created by this form. At the prefix, as do for the URI, will be added a final number indicating the *HostMachine* in the group
- **CPUType:** the type of CPU that is installed in each *HostMachine* of the group (e.g. Intel Xeon)
- **domain:** domain within which are situated the host machines. At the moment when this document is written the KB accepts as value for domain *DC01* (Figure 24)
- **username:** the username to access at the host machines. It will be inserted identical for all the *HostMachine* in the group.
- **password:** the password to access at the host machines. It will be inserted identical for all the *HostMachine* in the group.
- **# Host:** number of *HostMachine* that will be created with all features inserted in the other fields.
- **CPU:** number of CPUs in each *HostMachine*
- **CPUSpeed:** speed in GHz of each CPUs inserted in the previous field
- **memorySize:** size of principal memory ('RAM') in GB, in each *HostMachine*
- **capacity:** generic capacity of each *HostMachine*
- **operatingSystem:** operating system installed in each *HostMachine*
- **monitorState:** it shows if the monitor is on or off on each HostMachine. It is possible to choose between these values: Enabled indicates a working monitor and Disabled indicates a no working monitor



prefixName	hostTest
prefixIdentifier	HSTT
CPUType	Intel Xeon
domain	DC01
username	root
password	xxxx
# Host	10
CPU	32
CPUSpeed	2500
memorySize	128
capacity	100
operatingSystem	Red Hat
monitorState	Enabled

Figure 24 –Creation form of a hostMachine group

Each HostMachine can has associated entities and in the following sections it is possible to see what entities can be added.

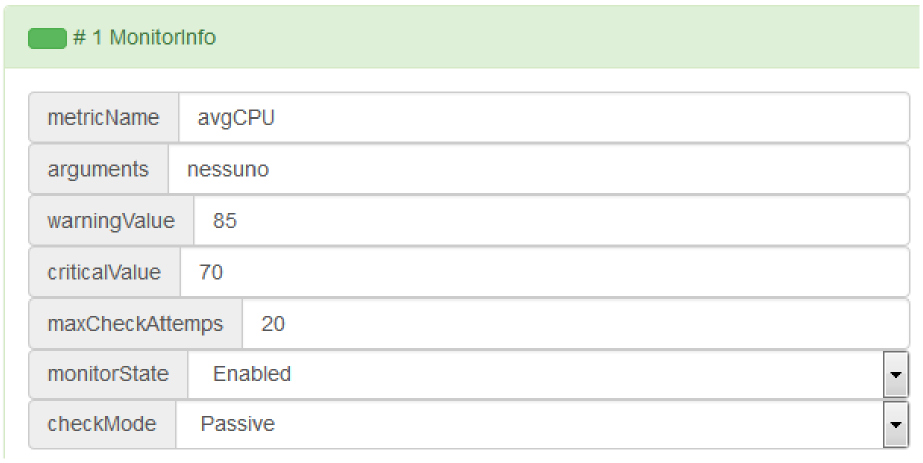
7.2.1.1 Monitor Info

This entity represents useful information for the SM that must monitor the metrics of real host machine present on data center. With the creation of the group of host machine at each HostMachine belonging to group will be associated an own MonitorInfo entity with the same information of the other.

The form for the insertion of a monitorInfo entity is what it can be seen in Figure 25, and the information that must be included is:

- **metricName:** name of the metric that SM must keep under control
- **arguments:** whatever it may be useful to control performed by
- **warningValue:** value that can trigger actions to prevent problems if reached by the metric
- **criticalValue:** value that can trigger actions to prevent problems if reached by the metric
- **maxCheckAttemps:** number of attempts that must be performed on the value of the metric before taking corrective action

- **monitorState:** it shows if the monitor is on or off on this MonitorInfo. It is possible to choose between these values: Enabled indicates a working monitor and Disabled indicates a no working monitor
- **checkMode:** it shows how the control must be performed. It possible to choose two values: Passive e Active


1 MonitorInfo

metricName	avgCPU
arguments	nessuno
warningValue	85
criticalValue	70
maxCheckAttemps	20
monitorState	Enabled
checkMode	Passive

Figure 25 – Creation form of a MonitorInfo entity

7.2.1.2 Local Network

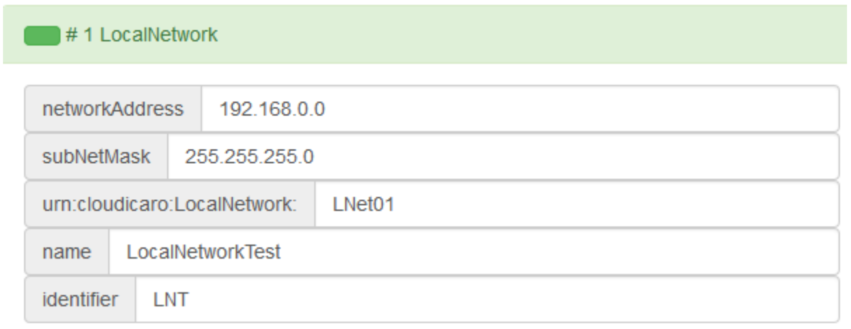
Each host machine can have one or more network adapter: they vary according to number of local network to which host machine is connected. Given that the system will generate a group of *HostMachine* and not a single *HostMachine*, it was considered appropriate not to include the same IP address for each *HostMachine*. Entering data on local network which leads the HostMachine group and application:

- automatically generate IP addresses to be associated with the *NetworkAdapter* of each *HostMachine*
- check that no two *LocalNetwork* inserted with the same URI and different data
- Check that the availability of IP addresses of each *LocalNetwork* is able to meet the request of IP addresses, based on the number of entities that must have a *NetworkAdapter* to access a specific *LocalNetwork*.

The data of each local network can be inserted through the form of Figure 26 entering the following data:

- **networkAddress:** the address of *LocalNetwork* which is being created
- **subNetMask:** the subnet mask of *LocalNetwork* which is being created. With this value the system can know how much IP address it is possible to generate in the new *LocalNetwork*

- **urn:cloudIcaro:LocalNetwork:** as just seen for the data center, also in this case it must be inserted the variable part of the URI of *LocalNetwork*. Such variable part must be unequivocal respect to other *LocalNetwork*
- **name:** name of the *LocalNetwork*
- **identifier:** identifier of the *LocalNetwork*



1 LocalNetwork

networkAddress	192.168.0.0
subNetMask	255.255.255.0
urn:cloudicaro:LocalNetwork:	LNet01
name	LocalNetworkTest
identifier	LNT

Figure 26 – Creation form of a LocalNetwork entity

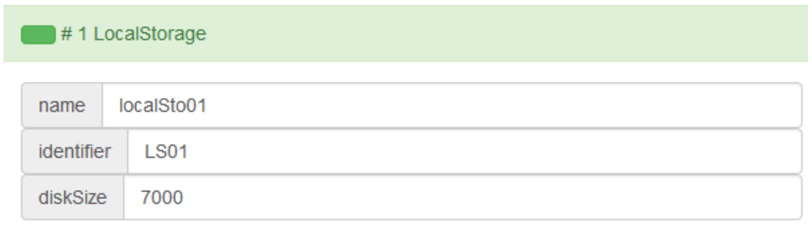
7.2.1.3 Local Storage

This entity behaves exactly as the entity monitorInfo: each created entity will be inserted in each *HostMachine* of the group is being generated. Each *LocalStorage* entity, unlike monitorInfo, will have an own URI automatically calculated by the system, considering *DataCenter* and *HostMachine* within which is situated and number of *LocalStorage* created in a *HostMachine* group:

urn:cloudicaro:LocalStorage:Test-012_HM01_LS0

The information that must be inserted in the form of Figure 27 is:

- **name:** name of the *LocalStorage*
- **identifier:** identifier of the *LocalStorage*
- **diskSize:** size in GB of disk in *LocalStorage*



1 LocalStorage

name	localSto01
identifier	LS01
diskSize	7000

Figure 27 – Creation form of a LocalStorage entity

7.2.1.4 Shared Storage

Unlike other entities associated to a *HostMachine*, *SharedStorage* entity is not completely defined in the form of the creation of *HostMachine* group. In this form the user can only insert the URI of a *SharedStorage*, to associate it at each *HostMachine* of the group that is generating, through the field visible in Figure 28.

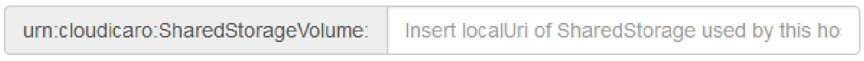

urn:cloudicaro:SharedStorageVolume:
Insert localUri of SharedStorage used by this ho

Figure 28 – Single field on which insert URI of a SharedStorage

In the following sections, it is described where it is possible to define such entity and the controls that are executed so that a *SharedStorage* URI inserted in the form of the creation of a *HostMachine* group, it is equal to a really created sharedStorage.

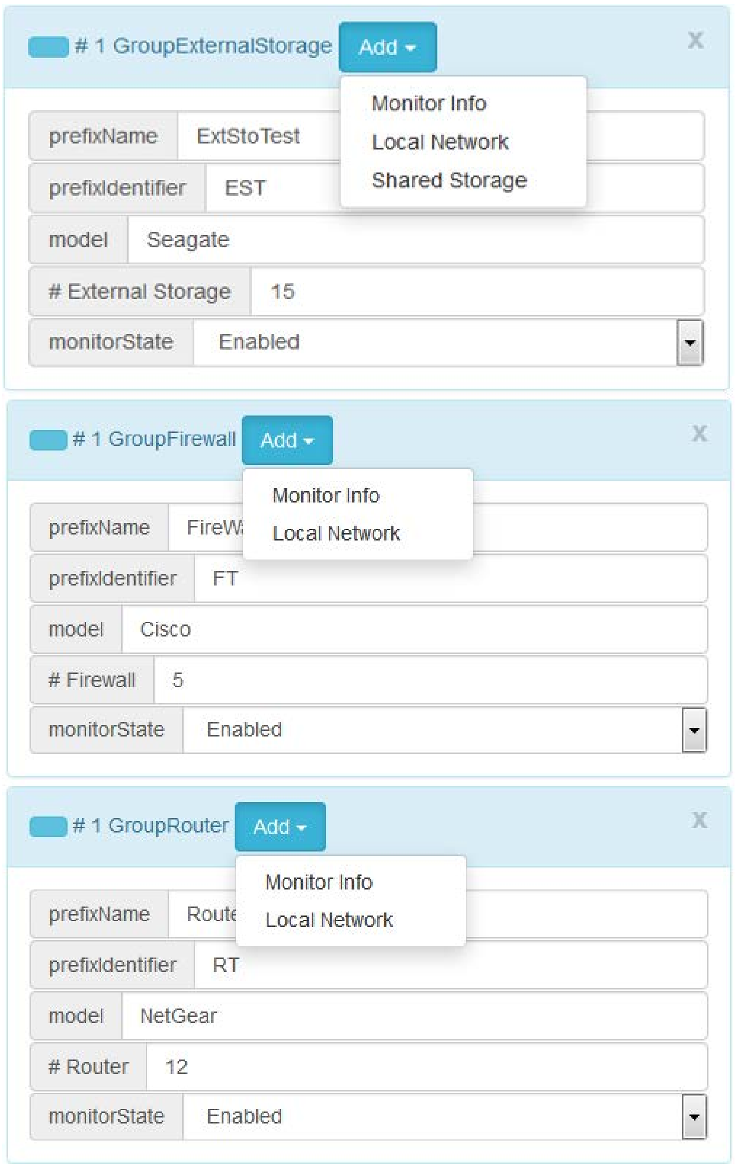
7.2.2 External Storage, Firewall, Router

The forms to insert entities *ExternalStorage*, *Firewall* and *Router* are identical. The only difference is in the *ExternalStorage* form with an additional entry on *dropDown* menu: this entry is relative to insertion of *SharedStorage* entity described in the previous section. For this similarity the forms are described together and they are shown in Figure 29.

For these entities it is valid what claimed for *HostMachine*: to speed the insertion of the entities of testing *DataCenter*, in the form is not inserted a single entity at a time but it is inserted information about a group of entities to create single entity automatically. Naturally to create a group with dimension 1 it is equal to create a single entity.

The information that must be inserted to create these entities:

- **prefixName:** name that must have the entity is being created. As the entity *HostMachine* the text inserted will represent the prefix of the name, followed by a number that identify the group itself and the entity in the group
- **prefixIdentifier:** identifier that must have the entity is being created. As the entity hostMachine the text inserted will represent the prefix of the identifier, followed by a number that identify the group itself and the entity in the group
- **model:** the mode(p.es. Cisco xxxx, WesternDigital 12345) of the entities that is being created
- **# Entity:** the number of entity that the user want create with the features inserted in other
- **monitorState:** it shows if the monitor is on or off on each *HostMachine*. It is possible to choose between these values: Enabled indicates a working monitor and Disabled indicates a no working monitor.


1 GroupExternalStorage

prefixName	ExtStoTest
prefixIdentifier	EST
model	Seagate
# External Storage	15
monitorState	Enabled

1 GroupFirewall

prefixName	FireW
prefixIdentifier	FT
model	Cisco
# Firewall	5
monitorState	Enabled

1 GroupRouter

prefixName	Route
prefixIdentifier	RT
model	NetGear
# Router	12
monitorState	Enabled

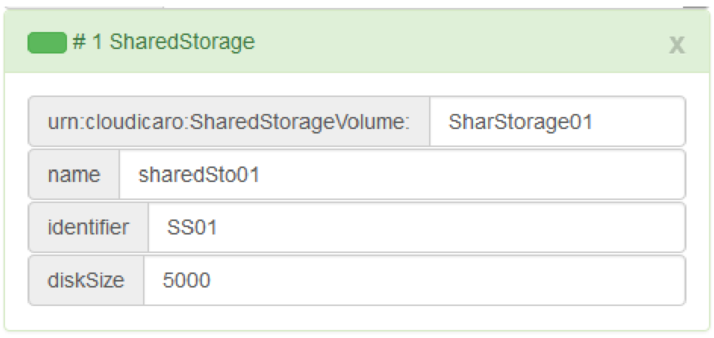
Figure 29 – Creation forms of ExternalStorage, Firewall, and Router groups

For entities that can be added as *MonitorInfo* and *LocalNetwork* it is possible to see the description in the previous paragraphs about entities associated to *HostMachine*: more precisely the paragraphs "*8.2.1.1 Monitor Info*" and "*8.2.1.2 Local Network*".

7.2.2.1 Shared Storage

As already seen in "*8.2.1.4 Shared Storage*" it is possible to associate to each *HostMachine* a *SharedStorage*, inserting in the creation form of a *HostMachine* group the URI of a *SharedStorage*.

In *ExternalStorage* form it is possible to create the entity *SharedStorage* because this last must necessary be part of an *ExternalStorage*. The form for the creation of this entity is in Figure 30.



1 SharedStorage

urn:cloudicaro:SharedStorageVolume:	SharStorage01
name	sharedSto01
identifier	SS01
diskSize	5000

Figure 30 – Creation form of a SharedStorage entity

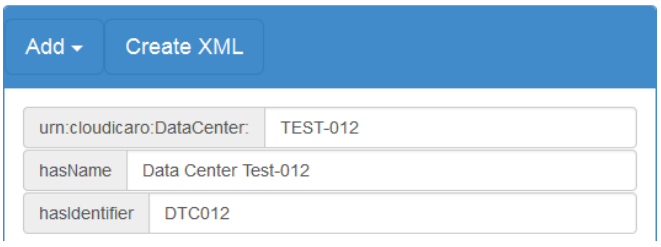
As it is possible to see the data inserted are the same of an entity *LocalStorage*, with the addition of the URI necessary to associated it to a *HostMachine*.

The data that must be inserted are:

- **urn:cloudIcaro:SharedStorageVolume:** URI of the *SharedStorage* that it can be associated to a *HostMachine*. Each *ExternalStorage* will be associated to a single entity generated by this form.
- **name:** name of the *SharedStorage*
- **identifier:** identifier of the *SharedStorage*
- **diskSize**: size in GB of the disk contained by *SharedStorage*

7.2.3 Create XML

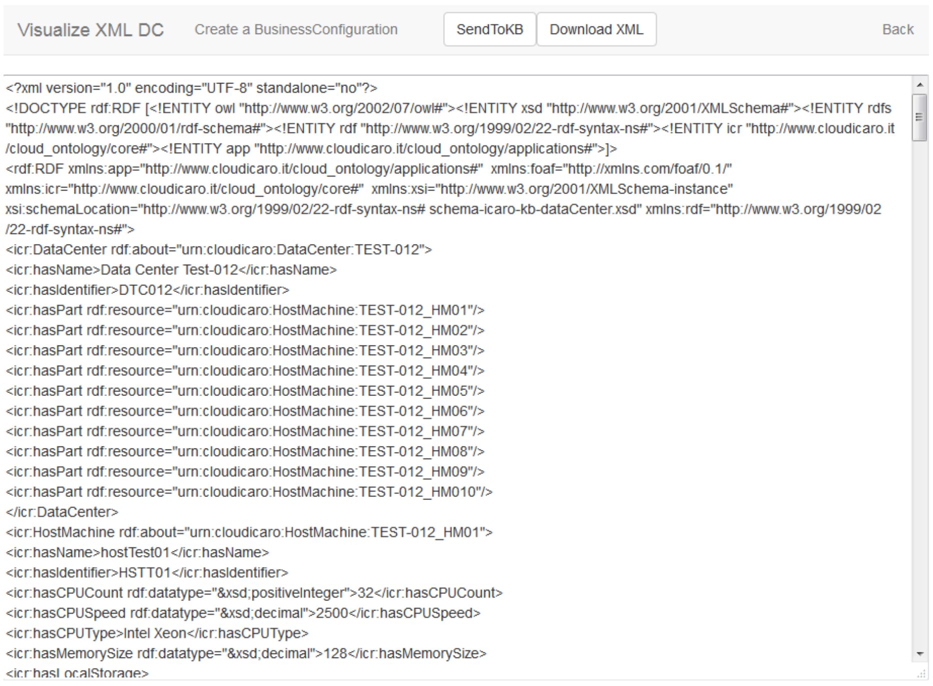
If all information, requested by the yellow alert present in Figure 22, are inserted: the button *CreateXML* is active (Figure 31), the alert disappears and it is possible to perform the creation of the XML file that represents the *DataCenter* relative to entered information.



urn:cloudicaro:DataCenter:	TEST-012
hasName	Data Center Test-012
hasIdentifier	DTC012

Figure 31 – Active *CreateXML* button

Once the button *CreateXML* is pressed, if there is no problem (see following paragraph), the system is redirect to web page in Figure 32.



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE rdf:RDF [<!ENTITY owl "http://www.w3.org/2002/07/owl#"><!ENTITY xsd "http://www.w3.org/2001/XMLSchema#"><!ENTITY rdfs
"http://www.w3.org/2000/01/rdf-schema#"><!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#"><!ENTITY icr "http://www.cloudicaro.it
/cloud_ontology/core#"><!ENTITY app "http://www.cloudicaro.it/cloud_ontology/applications#">]>
<rdf:RDF xmlns:app="http://www.cloudicaro.it/cloud_ontology/applications#"  xmlns:foaf="http://xmlns.com/foaf/0.1/"
xmlns:icr="http://www.cloudicaro.it/cloud_ontology/core#"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/1999/02/22-rdf-syntax-ns# schema-icaro-kb-dataCenter.xsd" xmlns:rdf="http://www.w3.org/1999/02
/22-rdf-syntax-ns#">
<icr:DataCenter rdf:about="urn:cloudicaro:DataCenter:TEST-012">
<icr:hasName>Data Center Test-012</icr:hasName>
<icr:hasIdentifier>DTC012</icr:hasIdentifier>
<icr:hasPart rdf:resource="urn:cloudicaro:HostMachine:TEST-012_HM01"/>
<icr:hasPart rdf:resource="urn:cloudicaro:HostMachine:TEST-012_HM02"/>
<icr:hasPart rdf:resource="urn:cloudicaro:HostMachine:TEST-012_HM03"/>
<icr:hasPart rdf:resource="urn:cloudicaro:HostMachine:TEST-012_HM04"/>
<icr:hasPart rdf:resource="urn:cloudicaro:HostMachine:TEST-012_HM05"/>
<icr:hasPart rdf:resource="urn:cloudicaro:HostMachine:TEST-012_HM06"/>
<icr:hasPart rdf:resource="urn:cloudicaro:HostMachine:TEST-012_HM07"/>
<icr:hasPart rdf:resource="urn:cloudicaro:HostMachine:TEST-012_HM08"/>
<icr:hasPart rdf:resource="urn:cloudicaro:HostMachine:TEST-012_HM09"/>
<icr:hasPart rdf:resource="urn:cloudicaro:HostMachine:TEST-012_HM010"/>
</icr:DataCenter>
<icr:HostMachine rdf:about="urn:cloudicaro:HostMachine:TEST-012_HM01">
<icr:hasName>hostTest01</icr:hasName>
<icr:hasIdentifier>HSTT01</icr:hasIdentifier>
<icr:hasCPUCount rdf:datatype="&xsd;positiveInteger">32</icr:hasCPUCount>
<icr:hasCPUSpeed rdf:datatype="&xsd;decimal">2500</icr:hasCPUSpeed>
<icr:hasCPUType>Intel Xeon</icr:hasCPUType>
<icr:hasMemorySize rdf:datatype="&xsd;decimal">128</icr:hasMemorySize>
<icr:hasLocalStorage>
```

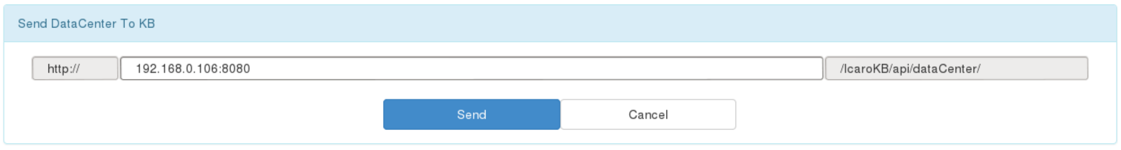
Figure 32 – Page of creation of XML file relative to a dataCenter entity

In this web page is visible a text area that contains the generated XML file: this text is not modifiable, but it is selectable to be copied in any other application as a text editor.

Clicking on the button "*Download XML*" the XML file is automatically sent to the browser, that the user are using to access the web application and it is possible to open it directly or download it on the own computer. The downloaded file is a simple text file with .xml extension and it has a filename:

datacenterYYYY-MM-DDTHH_MM_SS

showing what is the entity contained in the XML (in this case a *DataCenter*) and the timestamp of when the file is sent to the browser.

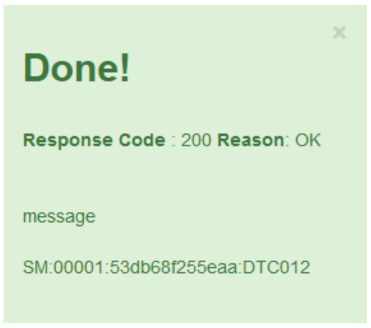


http://	192.168.0.106:8080	/IcaroKB/api/dataCenter/

Figure 33 – Form for the sending of a new dataCenter to KB

Clicking on the button "SendToKB" the form in Figure 33 is opened. In this form it is possible to insert the IP address and the port at which is possible to find the API REST exposed from the KB to perform the insertion of a dataCenter.

If the sending (to KB of the *DataCenter* XML file) succeeds, the panel in Figure 34 is shown.



Done!

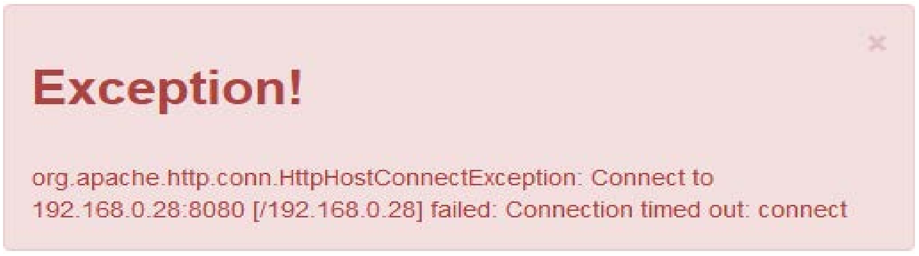
Response Code : 200 **Reason**: OK

message

SM:00001:53db68f255eaa:DTC012

Figure 34 – Message of successful sending to KB of a new DataCenter

If the sending to KB of the *DataCenter* XML file fails for any reason the panel in is shown.



Exception!

org.apache.http.conn.HttpHostConnectException: Connect to 192.168.0.28:8080 [/192.168.0.28] failed: Connection timed out: connect

Figure 35 – Message of unsuccessful sending to KB of a new DataCenter

The *Back* link redirect to creation web page of a *DataCenter* where are shown the data previously inserted that can be modified. Once those changes are made a new XML file is generated and the web page in Figure 32 is newly reloaded.

The "*Create a BusinessConfiguration*" link serves to create an entity BusinessConfiguration on the DataCenter that has just generate. To know how to create a BusinessConfiguration with the web application it is possible to see the chapter 8.3.

7.2.3.1 Exception in XML file generation

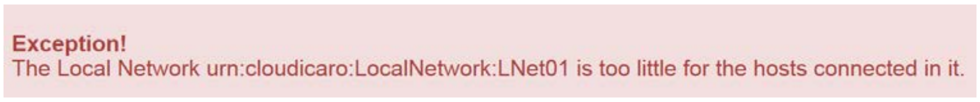
How it is written in the section "*8.2.1.2 Local Network*" the web application:

- Will check that no exist two *LocalNetwork* inserted with the same URI but with different information
- Will check if the availability, of IP addresses of each *LocalNetwork*, is able to satisfy the request of IP addresses base on the number of entity that must have a *NetworkAdapter* to access a determined *LocalNetwork*.

As mentioned in the section "*8.2.1.4 Shared Storage"* the application:

- Check if a *SharedStorage* URI, inserted in the creation form of a *HostMachine* group, matches to a *SharedStorage* effectively created.

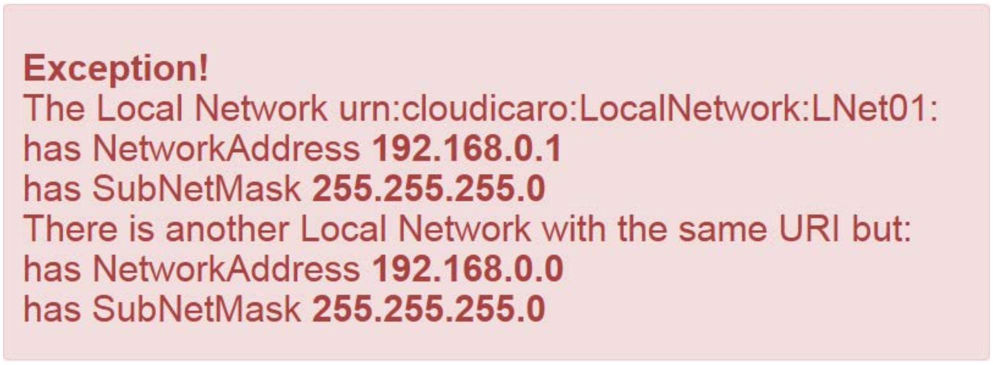
If the previous checks performed by the application fail, so error messages are generated to inform the user that inserted data are wrong. These messages are generated in a new page from which is possible to return to that where it is possible to insert data, so that it is feasible to change the data and avoiding generation of new exceptions.



Exception!
The Local Network urn:cloudicaro:LocalNetwork:LNet01 is too little for the hosts connected in it.

Figure 36 – Exception generated if a localNetwork does not have enough IP addresses

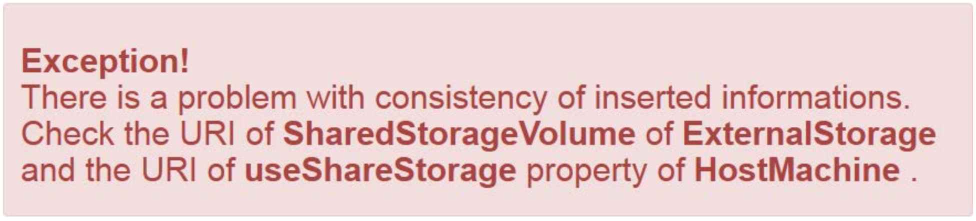
If the inserted *LocalNetwork* has not enough IP addresses for all the entities that are associated to the network, then the exception in Figure 36 is generated.



Exception!
The Local Network urn:cloudicaro:LocalNetwork:LNet01:
has NetworkAddress **192.168.0.1**
has SubNetMask **255.255.255.0**
There is another Local Network with the same URI but:
has NetworkAddress **192.168.0.0**
has SubNetMask **255.255.255.0**

Figure 37 – Exception generated if two localNetwork with same URI and different information are inserted

If a user wants that two *HostMachine* groups are in the same *LocalNetwork*, he must insert in each group the same *LocalNetwork* and the same data. If during this insertion the two *LocalNetwork* have the same URI they will be considered the same network from the application and IP addresses and subNetMask should be the same to not generate the exception in Figure 37.



Exception!
There is a problem with consistency of inserted informations.
Check the URI of **SharedStorageVolume** of **ExternalStorage** and the URI of **useShareStorage** property of **HostMachine** .

Figure 38 – Exception generated if a SharedStorage associated to a HostMachine it is not created in one ExternalStorage

As the previous, the last exception that can be generated checks that data inserted in two different form matches between them. At each *HostMachine* group is possible to associate an entity *SharedStorage*, but this last must be defined in an *ExternalStorage*. If a *SharedStorage* associated to a *HostMachine* group does not coincide with a *SharedStorage* created in an *ExternalStorage* the exception in Figure 38 is generated.

7.3 Create a BusinessConfiguration

As mentioned in section "*8.2.3 Create XML*" it is possible to create a *BusinessConfiguration* directly from the visualization web page of a new *DataCenter* just created. In this case the web page redirects the user directly to the form where is possible to create a *BusinessConfiguration*.

Instead, clicking on the button "*Create a BusinessConfiguration*", at the top in the navigation bar on homepage (Figure 21), the web application redirects the user to an "intermediate" page where is possible to choose a *DataCenter*, contained in the KB, in which to perform an insertion of a new *BusinessConfiguration*.

7.3.1 Choose DataCenter

In this page it is feasible to insert the IP Address and the port (Figure 39) at which it is possible to find a working KB.

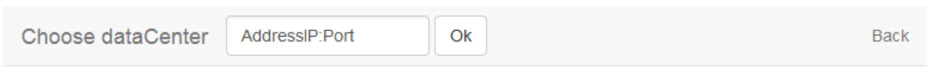
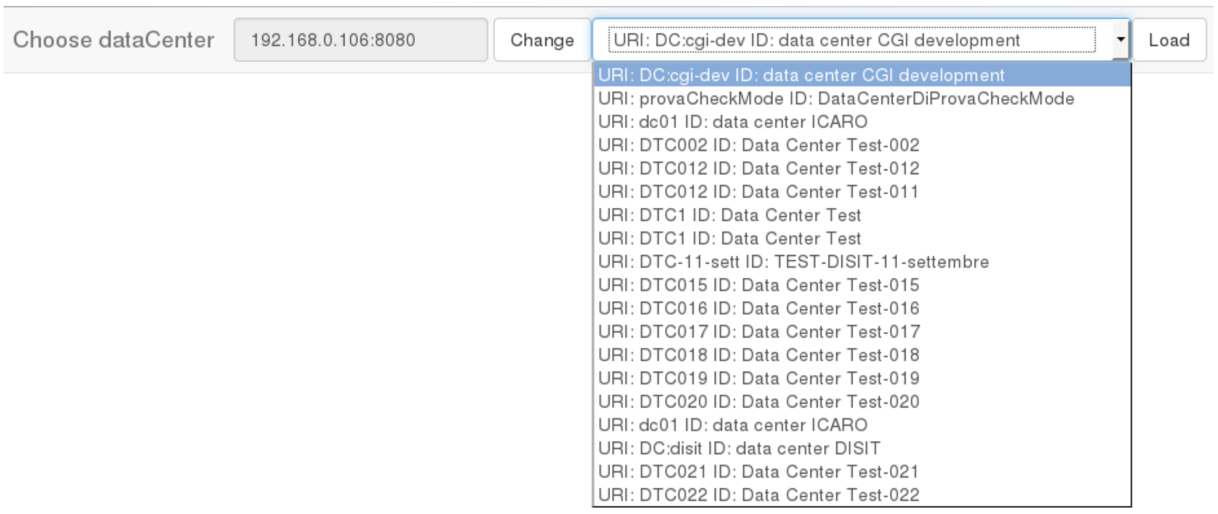


Figure 39 – Insertion of the IP Address and the port of a working KB

If the inserted data are correct a list of *DataCenters* present on KB is shown as in Figure 40.



URI: DC:cgi-dev ID: data center CGI development

URI: DC:cgi-dev ID: data center CGI development
URI: provaCheckMode ID: DataCenterDiProvaCheckMode
URI: dc01 ID: data center ICARO
URI: DTC002 ID: Data Center Test-002
URI: DTC012 ID: Data Center Test-012
URI: DTC012 ID: Data Center Test-011
URI: DTC1 ID: Data Center Test
URI: DTC1 ID: Data Center Test
URI: DTC-11-sett ID: TEST-DISIT-11-settembre
URI: DTC015 ID: Data Center Test-015
URI: DTC016 ID: Data Center Test-016
URI: DTC017 ID: Data Center Test-017
URI: DTC018 ID: Data Center Test-018
URI: DTC019 ID: Data Center Test-019
URI: DTC020 ID: Data Center Test-020
URI: dc01 ID: data center ICARO
URI: DC:disit ID: data center DISIT
URI: DTC021 ID: Data Center Test-021
URI: DTC022 ID: Data Center Test-022

Figure 40 – List of DataCenter contained in the KB achievable at 192.168.0.106:8080

If this KB is not that desired it is possible to change the inserted data clicking on the button *Change*.

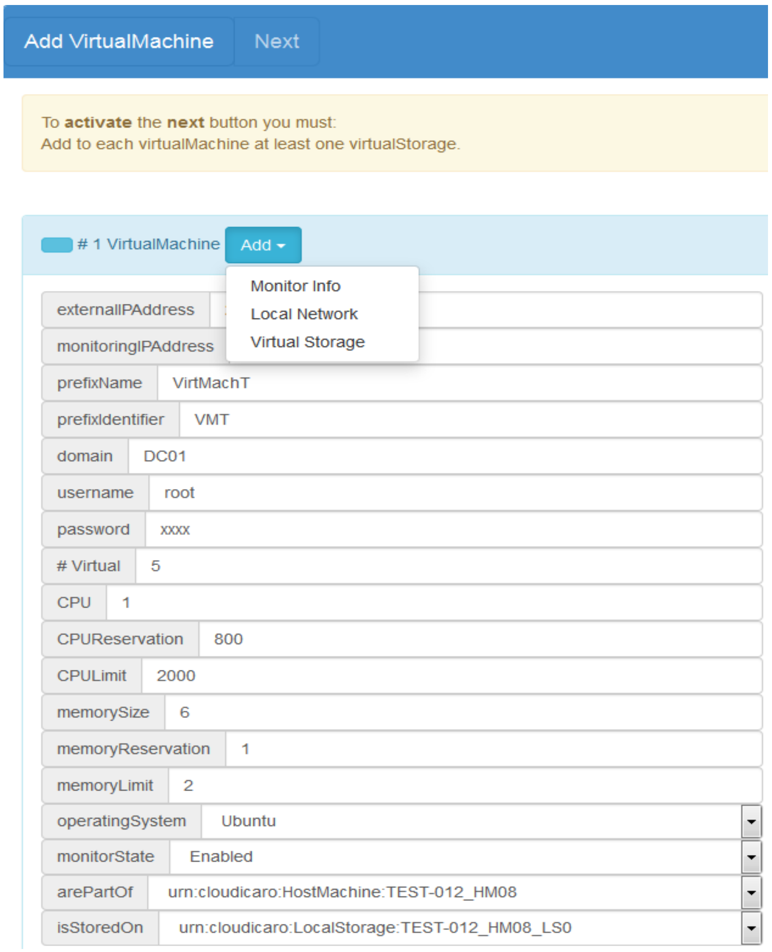
If the user wants to know *HostMachine*, *VirtualMachine* and *IcaroService* in one of the *DataCenter* in the list, he must press the button *Load*, once selected the desired *DataCenter*.

Once that the entities of a *DataCenter* are shown, the user can decide to create a *BusinessConfiguration* on the selected *DataCenter*: he can perform this operation clicking the button *Next* to access the form of insertion of a new *BusinessConfiguration*.

N.B.: When the user clicks on the button "Next" it is considered as current dataCenter on which to insert the new *BusinessConfiguration* the last that was retrieved through the button "Load". To select a *DataCenter* from the list without clicking the button "Load", it does not retrieve the *DataCenter* and it is considered as current *DataCenter* the last *DataCenter* for which the button "Load" is pressed. For this reason when the list of dataCenter appears, the button "Next" is not visible until a dataCenter is loaded from KB.

7.3.2 Create Virtual Machine

Once a *DataCenter* is chosen and retrieved from the KB, it is possible to create the new *BusinessConfiguration*. To do this, uppermost the *VirtualMachines*, at which the services should be installed, must be.



To **activate** the **next** button you must:
Add to each virtualMachine at least one virtualStorage.

externalIPAddress	
monitoringIPAddress	
prefixName	VirtMachT
prefixIdentifier	VMT
domain	DC01
username	root
password	xxxx
# Virtual	5
CPU	1
CPUReservation	800
CPULimit	2000
memorySize	6
memoryReservation	1
memoryLimit	2
operatingSystem	Ubuntu
monitorState	Enabled
arePartOf	urn:cloudicaro:HostMachine:TEST-012_HM08
isStoredOn	urn:cloudicaro:LocalStorage:TEST-012_HM08_LS0

Figure 41 – Creation form of a VirtualMachine group

The *VirtualMachines* can be inserted through the form in Figure 41, which has many fields equal to those present in the creation from of a *HostMachine* group, but there are some different. The information contained in the fields are described here:

- **externalIPAddress:** IP Address that is shown by *VirtualMachine* towards the outside of the local network and which can be used to access it
- **monitoringIPAddress:** IP Address belonging to *VirtualMachine* on which monitoring is performed
- **prefixName:** prefix name of all *VirtualMachine* that will be created by this form. At the prefix, as do for the URI, will be added a final number indicating the *VirtualMachine* in the group.
- **prefixIdentifier:** prefix identifier of all *VirtualMachine* that will be created by this form. At the prefix, as do for the URI, will be added a final number indicating the *VirtualMachine* in the group.
- **domain:** domain within which are situated the *VirtualMachines*. At the moment when this document is written the KB accepts as value for domain *DC01* (Figure 41)
- **username:** the username to access at the *VirtualMachine*. It will be inserted identical for all the *VirtualMachine* in the group.
- **password:** the password to access at the *VirtualMachine*. It will be inserted identical for all the *VirtualMachine* in the group.
- **# Virtual:** number of *VirtualMachine* that will be created with all features inserted in the other fields.
- **CPU:** number of CPUs in each *VirtualMachine*
- **CPUReservation:** minimum speed (in MHz) reserved for each CPU that is inserted in the previous field
- **CPULimit:** maximum speed (in MHz) reserved for each CPU that is inserted in the previous field. This limit can be set according of how much CPU is used by all *VirtualMachine* present in a *HostMachine*
- **memorySize:** indicates the maximum theoretical memory, physically present in the *HostMachine*, that can be used from every *VirtualMachine* on the group
- **memoryReservation:** indicates the minimum memory, physically present in the *HostMachine*, reserved for every *VirtualMachine* on the group. This memory is always available for each *VirtualMachine* and it does not depend on the other *VirtualMachine* that runs on the same *HostMachine*
- **memoryLimit**: indicates the maximum practical memory, physically present in the *HostMachine*, that can be used effectively by the *VirtualMachine*. This value depends on how much memory is used from the other VirtualMachine that runs on the same *HostMachine*.
- **operatingSystem**: number of CPUs in each *VirtualMachine*
- **monitorState**: it shows if the monitor is on or off on each *VirtualMachine*. It is possible to choose between these values: Enabled indicates a working monitor and Disabled indicates a no working monitor.
- **arePartOf:** indicates where the VirtualMachine group is sited, that is this field indicates the URI of HostMachine where the VirtualMachine group runs.

- **isStoredOn:** indicates where the *VirtualMachine* group is stored in the *HostMachine* selected in the previous field. It is possible to choose either a LocalStorage or a *VirtualStorage* associated to previous selected *HostMachine*

As for the *HostMachine*, also in this case is not necessary to insert the URI of every *VirtualMachine* of the group. Being a multiple entry, the URI is generated automatically and it is based on: the name of *DataCenter* and the *HostMachine* where the *VirtualMachine* is sited, the number of *VirtualMachine* contained by the group. For example, if a *VirtualMachine* group is generated with the information contained in the form of Figure 41, the URI that will be generated:

urn:cloudicaro:VirtualMachine:Test-OneHost_HM01_VM01

urn:cloudicaro: VirtualMachine:Test-OneHost_HM01_VM02

.

.

urn:cloudicaro: VirtualMachine:Test-OneHost_HM01_VM05

As for the other entities also for the *VirtualMachine* is possible to associate sub entities as: *MonitorInfo*, *LocalNetwork* e *VirtualStorage* (Figure 42).

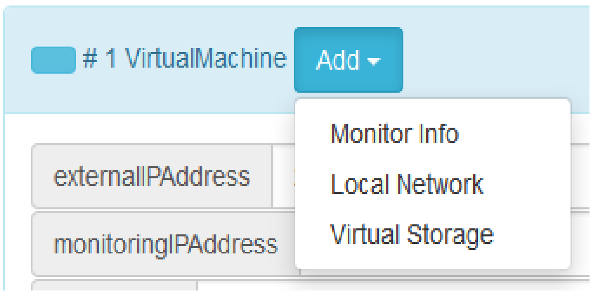


Figure 42 – Entities that it is possible to add on a virtualMachine

To see how to add those entities and for details about information to be included, it is possible to consult the sections "*8.2.1.1 Monitor Info*", "*8.2.1.2 Local Network*" e "*8.2.1.3 Local Storage*", as the form and the information to be included not vary with respect to the entities that it is possible to add on a *HostMachine*. The only change is that the URI of a virtualStorage is generated automatically compared to URI of a *LocalStorage* generated manually:

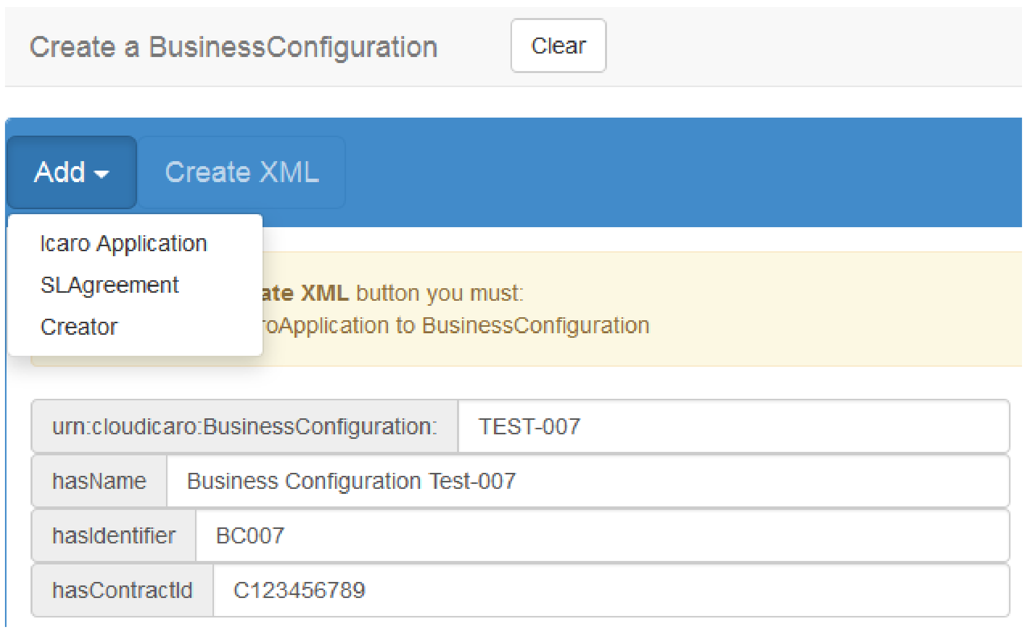
urn:cloudicaro:VirtualStorage:Test-OneHost_HM01_VM02_VS0

As the creation form of a *DataCenter* also this form show a yellow alert which informs the user about what entities must be included to enable the *Next* button. This button allows you to go to the next page to create the new *BusinessConfiguration*.

The *Clear* button allows resetting the page erasing all information inserted and all form added to principal form.

The *Back* link allows redirecting to web page of choice of a *DataCenter*, in the case that user should change the chosen *DataCenter* with another contained in the KB or should change the KB previously chosen.

Chose the *DataCenter* on which create the new *BusinessConfiguration* and created V*irtualMachine* on which insert *IcaroService*, it is explained how to create a *BusinessConfiguration* through the form that it is possible see in Figure 43.



urn:cloudicaro:BusinessConfiguration:	TEST-007
hasName	Business Configuration Test-007
hasIdentifier	BC007
hasContractId	C123456789

Figure 43 – Principal creation page of a Business Configuration

Through the form of Figure 43 it is possible to insert information about a *BusinessConfiguration*

- **urn:cloudIcaro:BusinessConfiguration:** the KB is realized through an ontology and in this kind of database each entity is unequivocally identified by URI. With regard to *BusinessConfiguration* the URI prefix is always the same, but the final part is variable and it can be chosen by the user, inserting it in this field.
- **hasName:** name of *BusinessConfiguration*
- **hasIdentifier:** identifier of *BusinessConfiguration*
- **hasContractId:** represents the id of associated contract to *BusinessConfiguration*. This id is useful to know the user that signed the *BusinessConfiguraion*

As it is possible to see in Figure 32 this information is not sufficient for creation of a *BusinessConfiguration*: the yellow alert indicates the other entities that must be inserted necessarily to create the *BusinessConfiguration* XML file that is being created.

Clicking on button *Add* is possible to add further entities, optional or mandatory to file creation (Figure 42), to *BusinessConfiguration*.

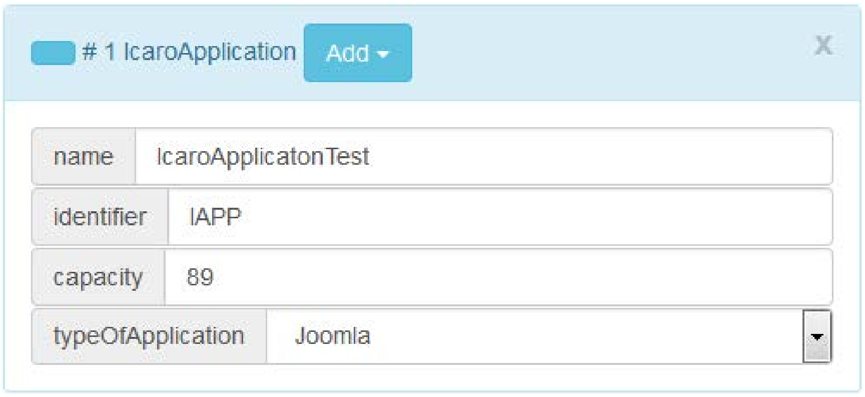
The button *Clear*, which is positioned at the top on the navigation bar in Figure 23, allows resetting all filled forms: once pressed will not be possible to recover the data inserted.

The button *Back*, which is positioned at the top right on the navigation bar in Figure 23, allows returning to web page in which it is possible to change information about VirtualMachines.

From this point is shown how is possible to add entities to *BusinessConfiguration*, starting from *IcaroApplication* entity that it is mandatory to the creation of XML file.

7.3.3 IcaroApplication

The form, that allows adding an *IcaroApplication* entity to *BusinessConfiguration*, is visible in Figure 45.



1 IcaroApplication Add

name	IcaroApplicatonTest
identifier	IAPP
capacity	89
typeOfApplication	Joomla

Figure 44 – Creation form of an IcaroApplication entity

The information that must be inserted is:

- **name:** name of *IcaroApplication*
- **identifier:** identifier of *IcaroApplication*
- **capacity:** generic capacity of each *IcaroApplication*
- **typeOfApplication:** it is possible to choose what kind of *IcaroApplication* must be inserted. The applications, present on the list, are those that can be instanced in cloud Icaro platform.

As for the previous entities also in this case is not necessary to insert the URI of the *IcaroApplication*, but this is not due to multiple entry. The URI of an *IcaroApplication* is created with information about type of application, *BusinessConfiguration* within which the application is inserted and the number of *IcaroApplication* contained in the *BusinessConfiguration*.

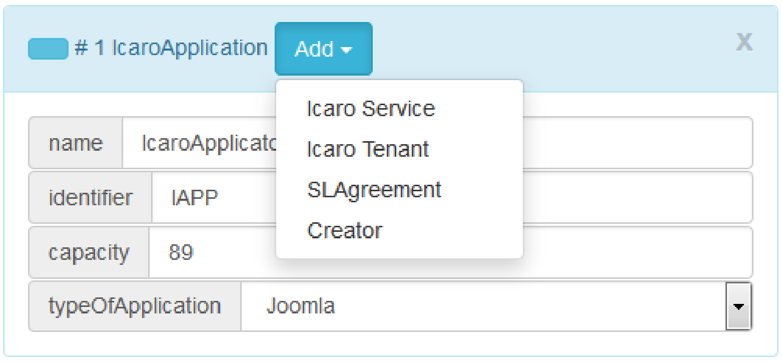


Figure 45 – Entities that is possible to add on an Icaro Application

If an *IcaroApplication* is inserted with information in Figure 45 and in Figure 43, the URI will be generated as:

urn:cloudicaro:Joomla:Test-007_0

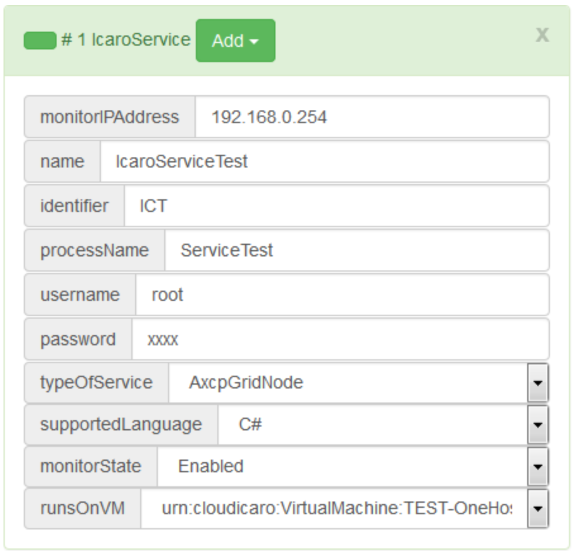
As it is possible to see in Figure 45 a user can add further entities to an *IcaroApplication*.

7.3.3.1 Icaro Service

Every *IcaroApplication* needs some services to work and if these services are not inserted then alert messages are generated by the system to inform the user of this problem (see 8.3.4.2).

In Figure 46 is shown the form to insert information about *IcaroServices* entity:

- **monitorIPAddress:** IP Address belonging to *IcaroService* on which monitoring is performed
- **name:** name of *IcaroService*
- **identifier:** identifier of *IcaroService*
- **processName:** the name of the process with which the *IcaroService* is executed in a *VirtualMachine*.
- **username:** username to access *IcaroService*
- **password:** password to access *Icaro Service*
- **typeOfService:** in this field the choice is constrained to the types of services that allow the execution of *IcaroApplication* in iCaro cloud platform. The user is responsible to choose those services that allow execution of an *IcaroApplication*.



monitorIPAddress	192.168.0.254
name	IcaroServiceTest
identifier	ICT
processName	ServiceTest
username	root
password	xxxx
typeOfService	AxcpGridNode
supportedLanguage	C#
monitorState	Enabled
runsOnVM	urn:cloudicaro:VirtualMachine:TEST-OneHo:

Figure 46 – Creation form of an IcaroService entity

- **supportedLanguage**: it is possible to choose which language is supported from the *IcaroServices*. The choice is constrained to languages that will support the Icaro cloud platform.

- **monitorState:** it shows if the monitor is on or off on each *IcaroServices*. It is possible to choose between these values: Enabled indicates a working monitor and Disabled indicates a no working monitor.

- **runsOnVM:** it indicates the *VirtualMachine* in which it is executed the *IcaroService*. The choice is constrained to *VirtualMachines* inserted at the previous step.

It is possible to see in Figure 47 that a user can add further entities to an *IcaroService* entity.

In fact, it is feasible a *MonitorInfo* entity, a *TCPPort* entity and an *UDPPort* entity. To see information about monitorInfo entity it is in "*8.2.1.1 Monitor Info*".

The other two entities contain simply a numeric field where the user can insert the *TCPPort* number and/or *UDPPort* number to which the *IcaroService* replies.

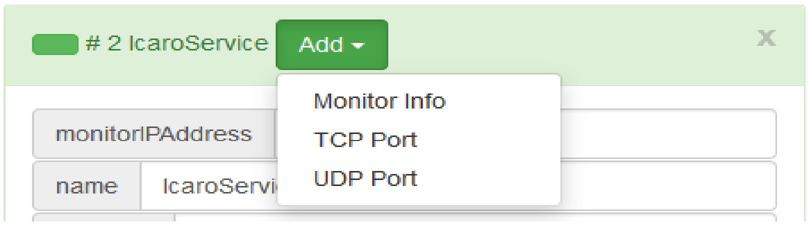


Figure 47 – Entities that is possible to add-on an Icaro Service

7.3.3.2 SLAgreement, SLObjective, SLAction e SLMetric

As planned in the KB at each IcaroApplication can be associated a *SLAgreement* and this entity can be inserted through the form in Figure 48.

The information that can be inserted in a *SLAgreement* entity is:

- **startTime:** indicates when the *SLAgremment* begins to be valid

- **endTime:** indicates when the *SLAgremment* ends to be valid

The *SLAgreement* is unequivocal for each *IcaroApplication* and only one can be created, but it is possible to add to such entity an undefined number of *SLObjective*, which are composed by *SLActions*, that represent the actions undertake from the system when the clauses of *SLAMetric* are not respected (Figure 48).

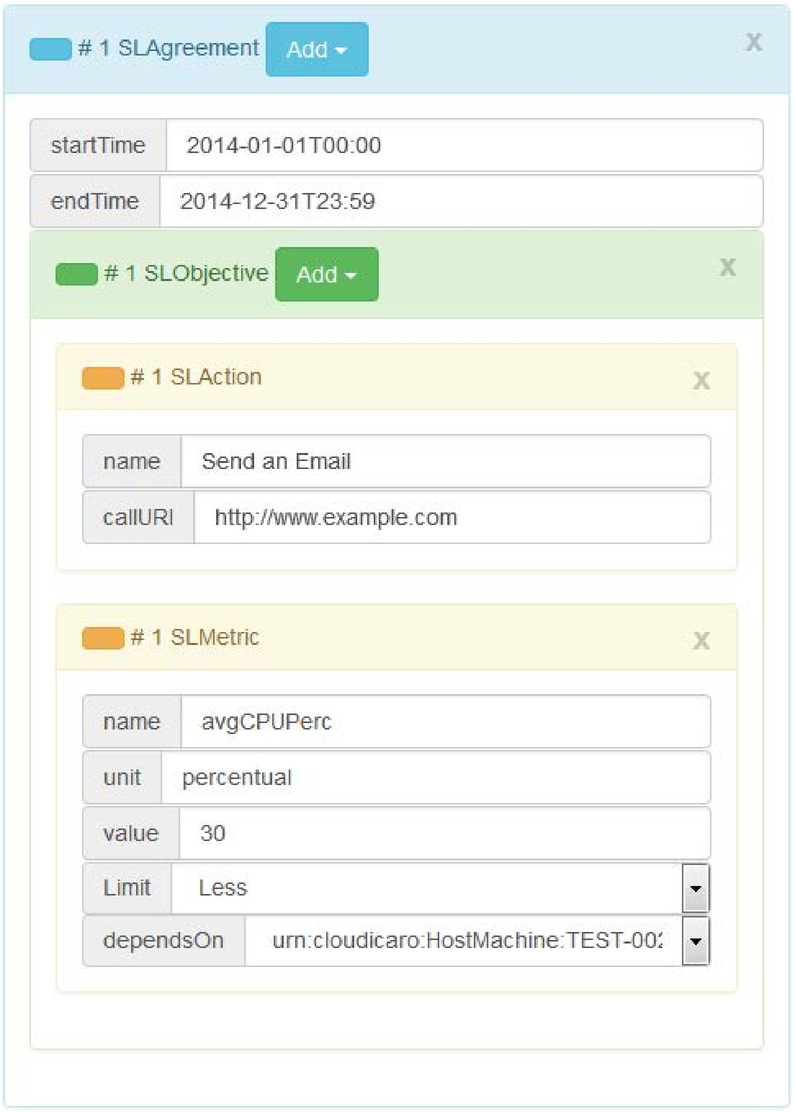
In Figure 48 it is possible to see that the *SLObjective* entity has not their fields, but it is a simple container for entities *SLAction* and *SLMetric*.

The information that can be inserted in a *SLAction* entity is:

- **name:** name of the action that is executed if the *SLMetric* constraints are not respected
- **callURI:** if the action that must be executed, it can be called through an URI, in this field this last must be inserted:

The information that can be inserted in a *SLMetric* entity is:

- **name:** the name of the metric that is stored in the KB and it is under control to respect this *SLAgreement*
- **unit:** unit of measure that is associated to follow field named value



startTime	2014-01-01T00:00
endTime	2014-12-31T23:59

name	Send an Email
callURI	http://www.example.com

name	avgCPUPerc
unit	percentual
value	30
Limit	Less
dependsOn	urn:cloudicaro:HostMachine:TEST-00:

Figure 48 – Creation form of SLAgreement relative entities

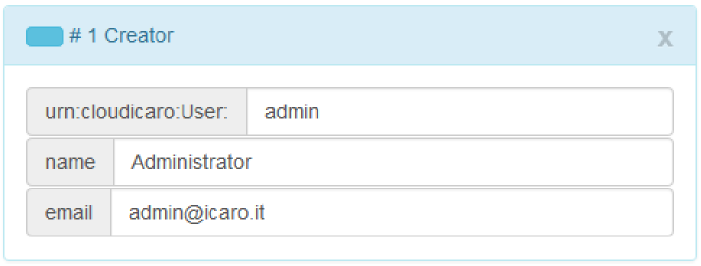
- **value:** the value that must be respected. This value is compared with the limit inserted in the following field

- **limit:** indicates the conditions that must be respected by memories in the KB respect to value inserted in the previous field ("value"). The values that can be chosen are: Greater, Less o Equal and they match with >, < e =.
- **dependsOn:** naturally the same metric can be measured on more resources contained in the cloud platform and in this field must be inserted the resource of interest to respect *SLAgreement* clause. At the moment it is possible to choose between *Host* and *Virtual Machine*

7.3.3.3 Creator

An icaroApplication can be associated to its Creator. For this, the following information must be inserted in the form of Figure 49:

- **urn:cloudIcaro:User:** URI of the Creator
- **name:** name of the Creator
- **email:** email of the Creator


1 Creator

urn:cloudicaro:User:	admin
name	Administrator
email	admin@icaro.it

Figure 49 – Creation form of a Creator entity

This entity is similar to *LocalNetwork* entity seen in the previous sections: this last is inserted to create automatically the *NetworkAdapter* entities and to associate them with the entities that are connected at that network, furthermore the system checks if two *LocalNetwork* with the same URI have also the same information.

In fact, when the data is inserted in this form a user entity is generated and this last will be associated to other entity to realize the property *hasCreator*.

Furthermore, if the inserted user is a creator for two *IcaroApplication* then the system must control that don't exist two *User* with the same URI and different name and email.

In the previous section are described the information and the entities necessary to creation of an entity *IcaroApplication* in a *BusinessConfiguration*. Beyond such entity it is possible to add in a *BusinessConfiguration* the entities *SLAgreement* and *Creator*, whereas such entities have the same form of entities that it is possible to add to an icaroApplication it is recommended to see the previous sections to know how insert such information in the form.

7.3.4 Create XML

Concerning the creation of the XML file of the *BusinessConfiguration* inserted and the page that is accessed once created this file you can see "*8.2.3 Create XML*" on the creation of the XML file of a *DataCenter*.

The only changes are that: in the navigation bar is not present the link to create a new *BusinessConfiguration* and the REST API, which is called for the inclusion of the *BusinessConfiguration* in the KB, is different.

7.3.4.1 Exception in XML file generation

In the previous section it was stated that the web application must check if two creators are inserted to (e.g. one on the businessConfiguration and one on icaroApplication):

- Avoid that two user exist with same URI and different name and/or email

In the case that the check fails, i.e. two *Users/Creators* exist with the same URI and different name and/or email. The message in Figure 50 is shown.

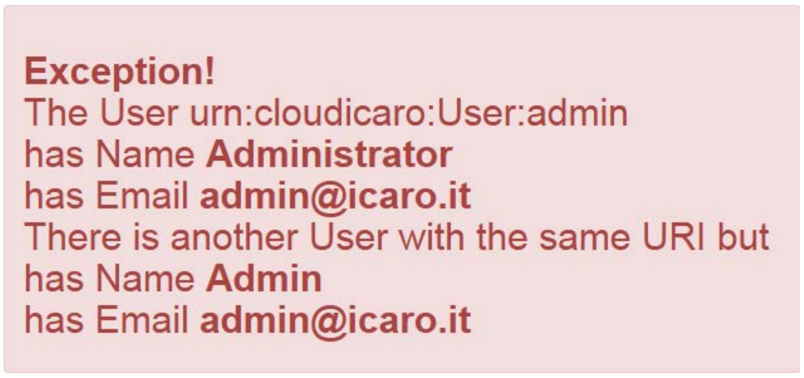

Exception!
The User urn:cloudicaro:User:admin
has Name **Administrator**
has Email **admin@icaro.it**
There is another User with the same URI but
has Name **Admin**
has Email **admin@icaro.it**

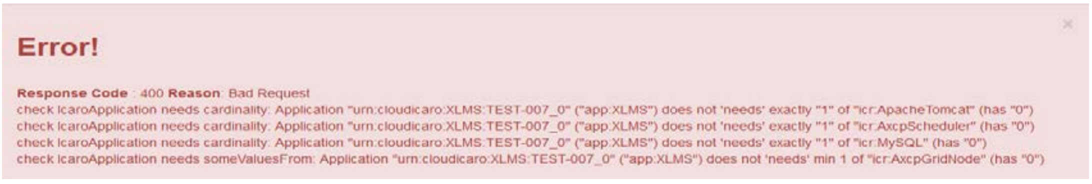
Figure 50 – Exception generated if two creations with same URI and different information are inserted

The user, read the message, can return to previous page to make necessary changes to avoiding that error message is already shown.

7.3.4.2 Exception in XML file insertion to KB

As mentioned at the beginning of section "*8.3.3.1 Icaro Service*", each *IcaroApplication* needs services to work and if these services are not inserted, then the system will warn user through red alert message to this lack.

Such messages, unlike the exceptions that are generated by the system, are generated during the insertion of the file within the KB, as they concern the rules included in the KB and not in the *XSD-SCHEMA* for generating XML file.



```
Error!

Response Code : 400 Reason: Bad Request
check IcaroApplication needs cardinality: Application "urn:cloudicaro:XLMS:TEST-007_0" ("app:XLMS") does not 'needs' exactly "1" of "icr:ApacheTomcat" (has "0")
check IcaroApplication needs cardinality: Application "urn:cloudicaro:XLMS:TEST-007_0" ("app:XLMS") does not 'needs' exactly "1" of "icr:AxcpScheduler" (has "0")
check IcaroApplication needs cardinality: Application "urn:cloudicaro:XLMS:TEST-007_0" ("app:XLMS") does not 'needs' exactly "1" of "icr:MySQL" (has "0")
check IcaroApplication needs someValuesFrom: Application "urn:cloudicaro:XLMS:TEST-007_0" ("app:XLMS") does not 'needs' min 1 of "icr:AxcpGridNode" (has "0")
```

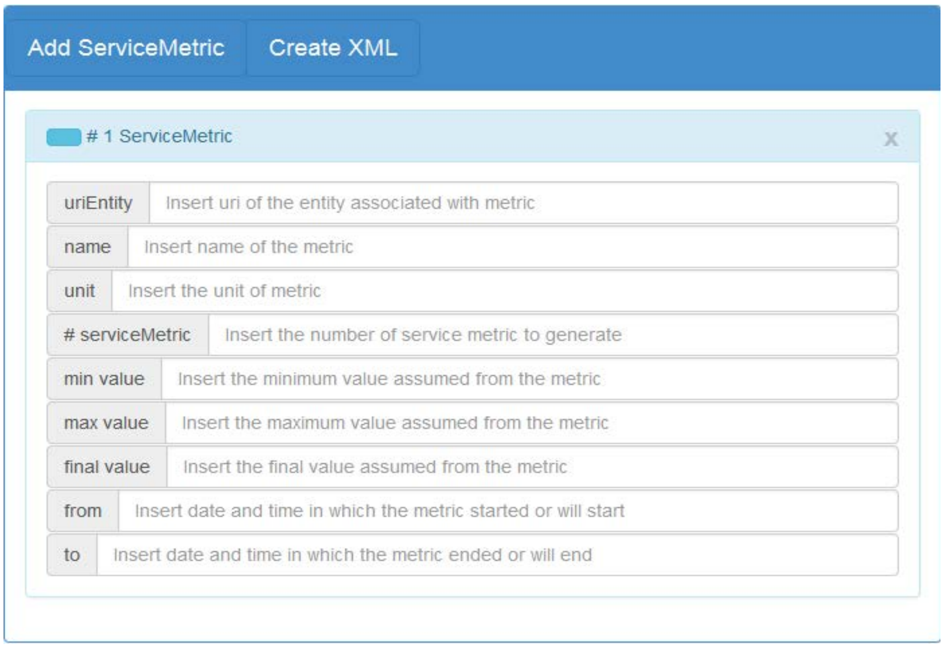
Figure 51 – KB message to notify the missing icaroService necessary to XLMS icaroApplication

For this reason the system show a message like the one in Figure 51 in response to the attempt to send an XML file containing a BusinessConfiguration to the KB. In the file there is an icaroApplication of type XLMS, without the needed services.

7.4 Create ServiceMetrics

This operation allows creating a predefined number of metrics that must be associated with a determined entity (e.g. a *HostMachine* or a *VirtualMachine*), deciding the period in which they are to be distributed. The values that should have during that period and the final value which must be inserted in the last metric.

This operation can be useful if a user want to check correct detection of problems, for example, related to an overrun of the limits imposed by the SLA, through the metrics included in KB.



uriEntity	Insert uri of the entity associated with metric
name	Insert name of the metric
unit	Insert the unit of metric
# serviceMetric	Insert the number of service metric to generate
min value	Insert the minimum value assumed from the metric
max value	Insert the maximum value assumed from the metric
final value	Insert the final value assumed from the metric
from	Insert date and time in which the metric started or will start
to	Insert date and time in which the metric ended or will end

Figure 52 – Creation form of a ServiceMetric entity

For the creation of the metrics, the form that must be filled is that of Figure 52, and the necessary information is as follows:

- **uriEntity:** URI of the entity to which the metrics, automatically generated, are referred
- **name:** name of all metrics
- **unit:** unit of measure that must be associated to values inserted
- **# serviceMetric:** the number of metrics that must be generated
- **minValue:** the minimum value that the metric can assume
- **maxValue:** the maximum value that the metric can assume
- **finalValue:** the last value that the metric must assume. It can be a value outside of the interval minValue – maxValue.
- **from:** start date of the metrics.
- **to:** end date of the metrics.

The period of generation of metrics is a dummy period and metrics will be generated all instantly to be sent to the KB.

7.4.1 Create XML

Concerning the creation of the XML file of the *ServiceMetrics* inserted and the page that is accessed once created this file you can see "*8.2.3 Create XML*" on the creation of the XML file of a datacenter.

The only changes are that: in the navigation bar is not present the link to create a new *BusinessConfiguration*, since the system are creating service metrics, the REST API, that is called for the inclusion of the *ServiceMetrics* in the KB, is different and the generated XML file contains all *ServiceMetrics* instead of generating a single XML for each *ServiceMetric*

7.5 Simulate DataCenter

This operation allows to simulating the functioning of a cloud platform to:

- Automatically generate metrics (in a different way to the previous described)
- Study the behavior of the cloud platform during the insertion of new *BusinessConfiguration*, in particular when are inserted new *VirtualMachine* and *IcaroService*

The simulation can be performed in two ways:

- **Real-Time:** the simulation time is the same as the user. Through this simulation it will be studied the behavior of the cloud during the insertion of new *BusinessConfiguration*.

- **Faster:** in this case the simulation time is not the same as the user: the simulation time depends on the computing power of the system. Start date and end date are selected and in this period are automatically generated the metrics for all entities of the previous chosen *DataCenter*.

The choice between the simulations can be done through the button visible in Figure 53. After choosing one of the two types of simulation the user is redirected on the page of choice of the *DataCenter*, where it is possible to choose which *DataCenter* simulate. To have information about this page it possible to see the section “*8.3.1 Choose DataCenter*”.

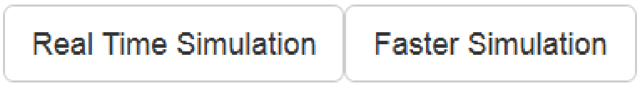
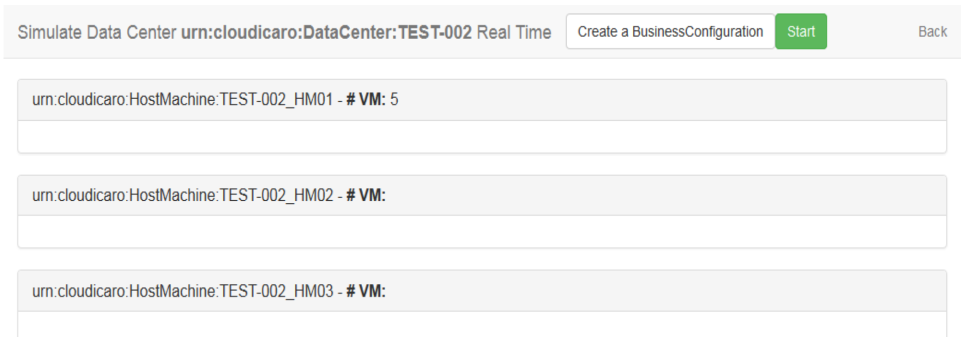


Figure 53 – Button to choose the simulation to perform

7.5.1 Real Time Simulation

The page, that allows simulating a *DataCenter*, is visible in Figure 54.


Simulate Data Center **urn:cloudicaro:DataCenter:TEST-002** Real Time | Create a BusinessConfiguration | Start | Back

urn:cloudicaro:HostMachine:TEST-002_HM01 - **# VM:** 5

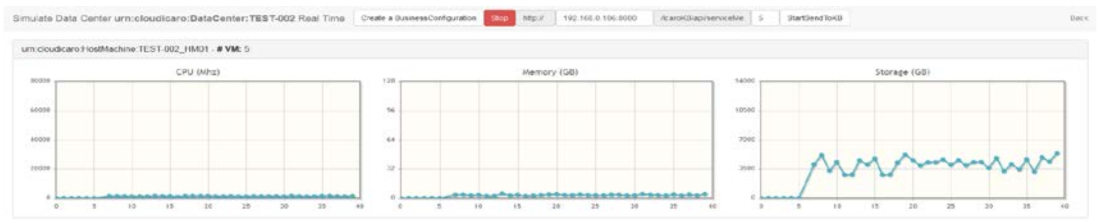
urn:cloudicaro:HostMachine:TEST-002_HM02 - **# VM:**

urn:cloudicaro:HostMachine:TEST-002_HM03 - **# VM:**

Figure 54 – Page of the real-time simulation

In the page it is possible to see some “panel” that will be filled with graph once that the simulation start. At each “panel” is associated a *HostMachine* and in the header is indicated how many *VirtualMachine* are present in the *HostMachine* relative to that “panel”.

To start the simulation the user must clicking on the button “Start”. When the simulation is started, in the page appear the graphs relative to simulation data and the button on the navigation bar are modified as in Figure 55.


Simulate Data Center urn:cloudicaro:DataCenter:TEST-002 Real Time | Create a BusinessConfiguration | Stop | http:// | 192.168.0.106:8080 | /icaroKB/api/service/e | 5 | StartSendToKB | Back

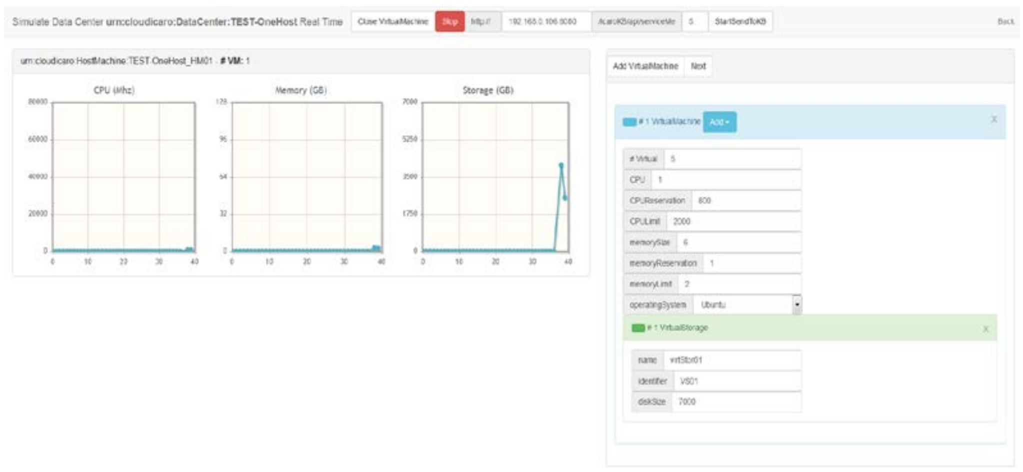
urn:cloudicaro:HostMachine:TEST-002_HM01 - **# VM:** 5

Figure 55 – DataCenter simulation with graphs of: CPU, Memory and Storage

In the three graphs it is possible to see the trend of the simulated data of CPU, Memory e Storage related to a single *HostMachine*. Charts are updated every 5 seconds with “auto-refresh” of the page.

The button *Stop* allows stopping instantly the simulation.

The form that follows the *Stop* button allows choosing a KB (by inserting IP address and port of a running KB) and the time by which the simulated data is sampled and sent to be included in the choice KB. The timestamp that will be inserted in the metrics sent to KB is reported at the time that these are sampled.



http://	192.168.0.106:8080
/icaroKB/api/serviceMe	5

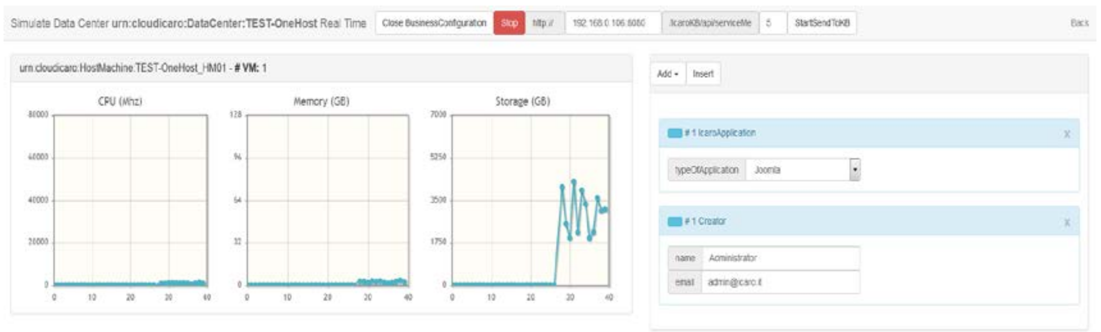
urn:cloudicaro:HostMachine:TEST-OneHost_HM01 - **# VM: 1**

# Virtual	5
CPU	1
CPUReservation	800
CPULimit	2000
memorySize	6
memoryReservation	1
memoryLimit	2
operatingSystem	Ubuntu

name	virtStor01
identifier	VS01
diskSize	7000

Figure 56 – Simulation page with form to add VirtualMachines

The button “*Create a BusinessConfiguration*”, if clicked, at the moment shows a form for the insertion of *VirtualMachine* (Figure 56) and then a form for the insertion of *BusinessConfiguration* (Figure 57): The control logic, that allows adding the entities dynamically to the simulator, must be projected yet.


Simulate Data Center urn:cloudicaro:DataCenter:TEST-OneHost Real Time
Close BusinessConfiguration
Stop
StartSendToKB
Back
Add
Insert
# 1 IcaroApplication
# 1 Creator
CPU (Mhz)
Memory (GB)
Storage (GB)

http://	192.168.0.106:8080
/icaroKB/api/serviceMe	5

urn:cloudicaro:HostMachine:TEST-OneHost_HM01 - **# VM: 1**

typeOfApplication	Joomla
name	Administrator
email	admin@icaro.it

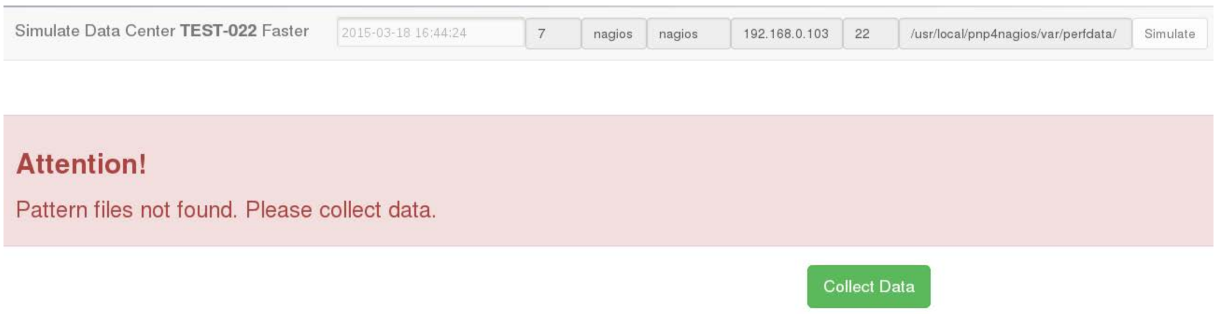
Figure 57 – Simulation page with form to add BusinessConfiguration

7.5.2 Fast Simulation

This type of simulation is useful when the user needs a tool to replicate past data of a real *DataCenter* in a selected simulated *DataCenter*. This simulation runs in a different way respect to the operation of “*Create ServiceMetrics*”:

- Generates the metrics for all entities belonging to the selected dataCenter rather than generates them for a single entity
- Generates the metrics with a period of 5 minutes, instead to distribute them evenly in the indicated period
- It is not provided limits to values that can be assumed form the metric and it is not present a final value.
- The metrics are taken from real data previously collected

If any real data is collected, IcaroCloudSimulator shows the alert message, which is possible to see in Figure 58, and the green button "*Collect Data*" that, if clicked by the user, redirect him to the page where it is possible to start the collection of real data.

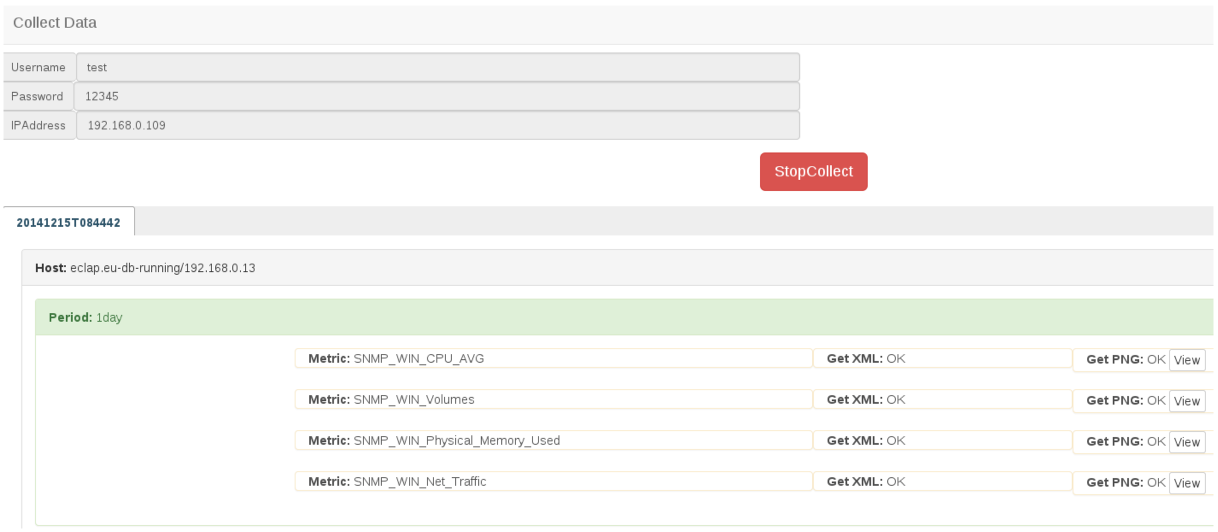

Simulate Data Center **TEST-022** Faster | 2015-03-18 16:44:24 | 7 | nagios | nagios | 192.168.0.103 | 22 | /usr/local/pnp4nagios/var/perfdata/ | Simulate

Attention!
Pattern files not found. Please collect data.

Collect Data

Figure 58 – Alert message shows when there are not real data collected for fast simulation

Once the user presses the button "*StartCollect"* the system starts to retrieve the real data from the Nagios server through the RestAPI offers by the "*Supervisor & Monitor*" tool. The real data are collected with periods of one day, seven days and 30 days until the user clicks the button "*StopCollect*": these operations continues to work, also when the browser is closed.


Collect Data

Username	test
Password	12345
IPAddress	192.168.0.109

StopCollect

20141215T084442

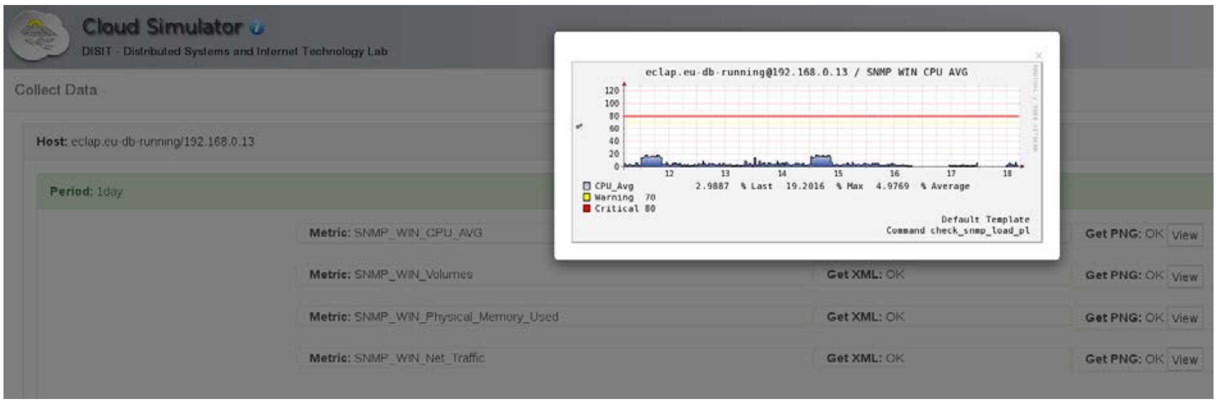
Host: eclap.eu-db-running/192.168.0.13

Period: 1day

Metric: SNMP_WIN_CPU_AVG	**Get XML:** OK	**Get PNG:** OK View
Metric: SNMP_WIN_Volumes	**Get XML:** OK	**Get PNG:** OK View
Metric: SNMP_WIN_Physical_Memory_Used	**Get XML:** OK	**Get PNG:** OK View
Metric: SNMP_WIN_Net_Traffic	**Get XML:** OK	**Get PNG:** OK View

Figure 59 – Web page where it is possible to see the real data collected and manage the collection operations

In the Figure 59, it is possible to see how the collected real data are shown: they are divided in tabs based on the day in which they are collected, in each tab there is one panel for each checked machine and in this last panel there is a variable number of panel. In fact, in the first tab there are four sub-panel: one for the data with period one day, one for the data with period seven days, one for the data with period 30 days and one for the meta-data. In the second tab there is only one sub-panel: that for the data with period one day due to the fact that the collection is performed each day at the same time and the data with period seven days are collected each seven days, i.e. each seven tabs.


Cloud Simulator
DISIT - Distributed Systems and Internet Technology Lab
Collect Data
Host: eclap.eu-db-running/192.168.0.13
Period: 1day
Metric: SNMP_WIN_CPU_AVG — **Get PNG:** OK View
Metric: SNMP_WIN_Volumes — **Get XML:** OK — **Get PNG:** OK View
Metric: SNMP_WIN_Physical_Memory_Used — **Get XML:** OK — **Get PNG:** OK View
Metric: SNMP_WIN_Net_Traffic — **Get XML:** OK — **Get PNG:** OK View

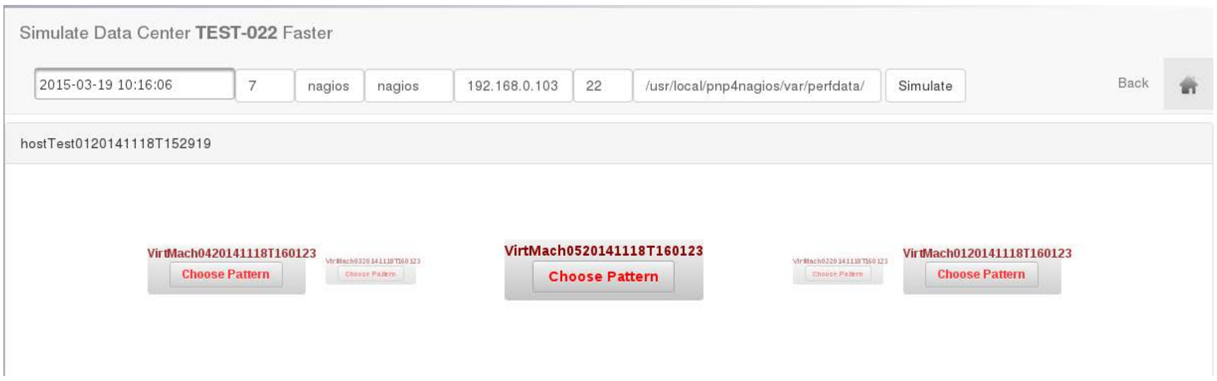
eclap.eu-db-running@192.168.0.13 / SNMP WIN CPU AVG

```
□ CPU_Avg        2.9887  % Last   19.2016  % Max   4.9769  % Average
🟨 Warning  70
🟥 Critical 80
                                                   Default Template
                                            Command check_snmp_load_pl
```

Figure 60 – Example of a chart relative to real data collected

The user can click on the *View* buttons to view the charts associated to the real data collected, see Figure 60. If the real data are present in the file system the fast simulation can be performed and it is possible to associate one pattern to each VirtualMachine that must be simulated, see Figure 61.

If the user does not associate any pattern to some VirtualMachine, the patterns are associated randomly to the VirtualMachine without a pattern.


Simulate Data Center **TEST-022** Faster

2015-03-19 10:16:06	7	nagios	nagios	192.168.0.103	22	/usr/local/pnp4nagios/var/perfdata/

hostTest0120141118T152919

VirtMach0420141118T160123 — Choose Pattern
VirtMach0320141118T160123 — Choose Pattern
VirtMach0520141118T160123 — Choose Pattern
VirtMach0220141118T160123 — Choose Pattern
VirtMach0120141118T160123 — Choose Pattern

Figure 61 – In this page it is possible to associate one pattern to each VirtualMachine to simulate

When the user clicks on the *"Choose Pattern"* button, a modal dialog is shown to the user, where he can choose the pattern viewing the chart related to the pattern that he is choosing: are shown the charts related to the patterns of all resources collected from one machine at the same time, see Figure 62.

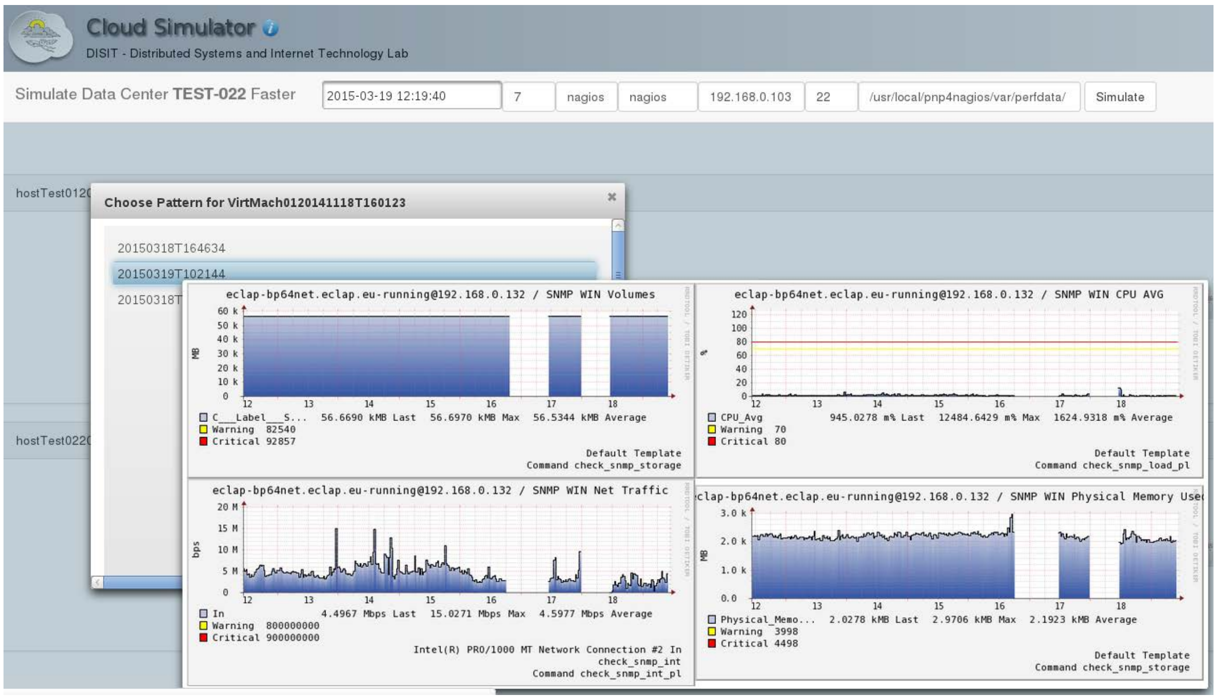
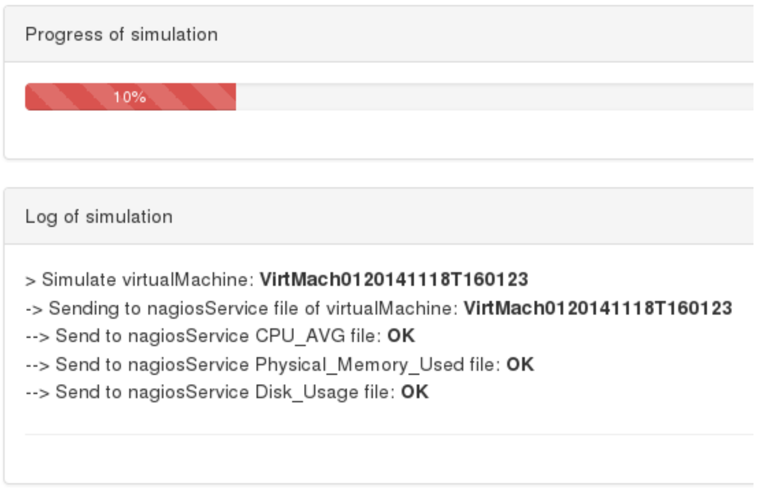

Cloud Simulator
DISIT - Distributed Systems and Internet Technology Lab
Simulate Data Center TEST-022 Faster
2015-03-19 12:19:40
7
nagios
nagios
192.168.0.103
22
/usr/local/pnp4nagios/var/perfdata/
Simulate
hostTest012
hostTest022
Choose Pattern for VirtMach0120141118T160123
20150318T164634
20150319T102144
20150318T
eclap-bp64net.eclap.eu-running@192.168.0.132 / SNMP WIN Volumes
C__Label__S... 56.6690 kMB Last 56.6970 kMB Max 56.5344 kMB Average
Warning 82540
Critical 92857
Default Template
Command check_snmp_storage
eclap-bp64net.eclap.eu-running@192.168.0.132 / SNMP WIN CPU AVG
CPU_Avg 945.0278 m% Last 12484.6429 m% Max 1624.9318 m% Average
Warning 70
Critical 80
Default Template
Command check_snmp_load_pl
eclap-bp64net.eclap.eu-running@192.168.0.132 / SNMP WIN Net Traffic
In 4.4967 Mbps Last 15.0271 Mbps Max 4.5977 Mbps Average
Warning 800000000
Critical 900000000
Intel(R) PRO/1000 MT Network Connection #2 In
check_snmp_int
Command check_snmp_int_pl
eclap-bp64net.eclap.eu-running@192.168.0.132 / SNMP WIN Physical Memory Used
Physical_Memo... 2.0278 kMB Last 2.9706 kMB Max 2.1923 kMB Average
Warning 3998
Critical 4498
Default Template
Command check_snmp_storage

Figure 62 – Charts of the real data collected at the same time from all resources of a one machine

Then the user must insert the start data of the simulation, the number of days that must be simulated and it must click the button *Simulate*. The metrics will be created instantly and then they are sent to Nagios server: the web page shows to the user two panel with respectively the progress and the log of simulation (Figure 63).



Progress of simulation

10%

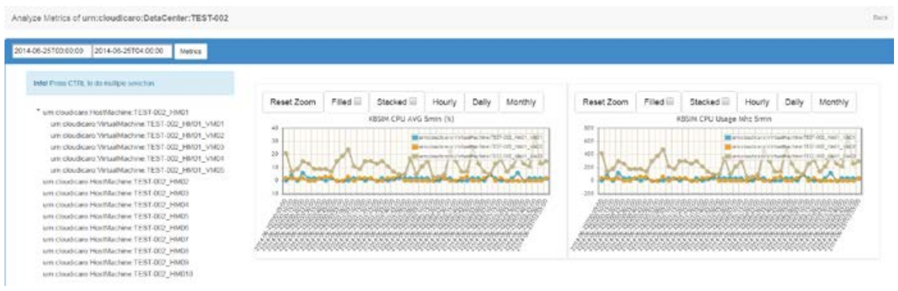
Log of simulation

```
> Simulate virtualMachine: VirtMach0120141118T160123
-> Sending to nagiosService file of virtualMachine: VirtMach0120141118T160123
--> Send to nagiosService CPU_AVG file: OK
--> Send to nagiosService Physical_Memory_Used file: OK
--> Send to nagiosService Disk_Usage file: OK
```

Figure 63 – Page of fast simulation where it is possible to see the log of simulated entity

7.6 Analyze Metrics

Through this operation it is possible to analyze the metrics that are present in the KB. As mentioned in section "*8.3.1 Choose DataCenter*", to use this operation it is necessary to choose and to load form the KB the *DataCenter* that the user want analyze. Once the *DataCenter* is loaded the user is redirected on the web page in Figure 64.



Analyze Metrics of urn:cloudicaro:DataCenter:TEST-002

2014-06-25T00:00:00 2014-06-25T04:00:00

Info! Press CTRL to do multiple selection

- urn:cloudicaro:HostMachine:TEST-002_HM01
  - urn:cloudicaro:VirtualMachine:TEST-002_HM01_VM01
  - urn:cloudicaro:VirtualMachine:TEST-002_HM01_VM02
  - urn:cloudicaro:VirtualMachine:TEST-002_HM01_VM03
  - urn:cloudicaro:VirtualMachine:TEST-002_HM01_VM04
  - urn:cloudicaro:VirtualMachine:TEST-002_HM01_VM05
- urn:cloudicaro:HostMachine:TEST-002_HM02
- urn:cloudicaro:HostMachine:TEST-002_HM03
- urn:cloudicaro:HostMachine:TEST-002_HM04
- urn:cloudicaro:HostMachine:TEST-002_HM05
- urn:cloudicaro:HostMachine:TEST-002_HM06
- urn:cloudicaro:HostMachine:TEST-002_HM07
- urn:cloudicaro:HostMachine:TEST-002_HM08
- urn:cloudicaro:HostMachine:TEST-002_HM09
- urn:cloudicaro:HostMachine:TEST-002_HM010

Figure 64 – Web page that allow to analyze the metrics in the SM

It is possible to note in the figure on the left the list of *HostMachines* and *VirtualMachines* contained in the *DataCenter* selected in the previous page.

On the list it is possible to select the *VirtualMachines* and the *HostMachines* for analyzing the metrics. Once the entity are selected the user must insert start date and end date of the period that he want analyze.

Clicking the button *Metrics* the charts of Figure 64 are created. If two selected entities are the same metric then the data will be grouped in a single chart and a curve will be shown for each entity: in Figure 64 there are 3 curves because 3 selected entities are the same metric "*KBSIM CPU AVG 5min*".

The button "*Reset Zoom*" take back the chart to its original resolution.

The *Filled* checkBox fills the underlying area of each curve that in Figure 64 is transparent.

The *Stacked* checkBox sums the contributions of each curve realizing a graph with accumulated data.

The last three buttons *Hourly*, *Daily* e *Monthly* are used to scale the graph resolution to have a better view on the trend data.

8. References

1. *Cloud computing: state-of-the-art and research challenges.* **Q. Zhang, L. Cheng, R. Boutaba.** s.l. : Journal of Internet Services and Applications, Vol. 1, pp. 7-18.

2. *Cloud computing: Issues and challenges.* **T. Dillon, C. Wu and E. Chang.** Perth : IEEE, 2010. Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on. pp. 27-33.

3. *Challenges towards Elastic Power Management in Internet Data Centers.* **J. Liu, F. Zhao, X. Liu, and W. He.** Montreal, Quebec, Canada : s.n., 2009. Workshop on Cyber-Physical Systems (WCPS),.

4. *GreenCloud: A Packetlevel Simulator of Energy-aware Cloud Computing Data Centers.* **D. Kliazovich, P. Bouvry, S. U. Khan.** 3, s.l. : Journal of Supercomputing, 2012, Vol. 62, p. 1263-1283.

5. *Cloud computing simulators: A detailed survey and future direction.* **A. Ahmed and A. S. Sabyasachi.** Guargon : IEEE, 2014. Advance Computing Conference (IACC), 2014 IEEE International. pp. 866 - 872.

6. *The Glasgow Raspberry Pi Cloud: A Scale Model for Cloud Computing Infrastructures.* **F. P. Tso, D. R. White, S. Jouet, J. Singer and D. P. Pezaros.** 2013. Distributed Computing Systems Workshops (ICDCSW), 2013 IEEE 33rd International Conference on. p. 108-112.

7. *CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms.* **R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose and Rajkumar Buyya.** 1, New York : Wiley Press, 2011, Software: Practice and Experience (SPE), Vol. 41, pp. 23-50. ISSN: 0038-0644.

8. *Study and Comparison of CloudSim Simulators in the Cloud Computing.* **R. Malhotra and P. Jain.** 4, 2013, SIJ Transactions on Computer Science Engineering and its Applications (CSEA), Vol. 1, pp. 111-115.

9. *iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator.* **A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero and I. M. Llorente.** 1, s.l. : Springer, 2012, Journal of Grid Computing, Vol. 10, p. 185-209.

10. *DCSim: A data centre simulation tool for evaluating dynamic virtualized resource management.* **M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya,.** 2012. The 6th International DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standard and the Cloud.

11. *CloudAnalyst: A CloudSim-based Visual Modeller for Analysing Cloud Computing Environments and Applications.* **B. Wickremasinghe, R. N. Calheiros and R. Buyya.** s.l. : IEEE Computer Society, 2010. 24th International Conference on Advanced Information Networking and Applications (AINA). p. 446-452.

12. ***The Network Simulator Ns2.*** **[Online] 2010. http://nsnam.isi.edu/nsnam/index.php/Main_Page.**

13. ***Introduction of Cloud Computing and Survey of Simulation Software for Cloud.*** **M. Aggarwal. 13, 2013, TIJ's Research Journal of Science & IT Management, Vol. 2. 2251-1563.**

14. ***CloudNetSim - Simulation of Real-Time Cloud Computing Applications.*** **T. Cucinotta and Santogidis A. Paris : s.n., 2013. 4th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS).**

15. ***NetworkCloudSim: Modelling Parallel Applications in Cloud Simulations.*** **S. Kumar Garg and R. Buyya. Melbourne : IEEE CS Press, 2011. Proceedings of the 4th IEEE/ACM International Conference on Utility and Cloud Computing.**

16. ***EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of Cloud computing applications.*** **R. N. Calheiros, M. A.S. Netto, César A.F. De Rose and R. Buyya,. 5, Software: Practice and Experience, Vol. 43, p. 595–612.**

17. ***MDCSim: A multi-tier data center simulation, platform.*** **Seung-Hwan Lim, Bikash Sharma, Gunwoo Nam, Eun Kyoung Kim and Chita R. Das,. New Orleans : IEEE, 2009. Cluster Computing and Workshops, IEEE International Conference on.**

18. CSIM Development Toolkit for Simulation and Modeling. [Online] http://www.mesquite.com/..

19. ***Modeling and Simulation Frameworks for Cloud Computing Environment: A Critical Evaluation.*** **A. Bashar.**