Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

OpenKnowTech "Laboratorio di Tecnologie per la Integrazione, Gestione e Distribuzione di Dati, Processi e Conoscenze"

Obiettivo Realizzativo OR2 Open Grid

Deliverable D2.2

Un sistema autonomico per la gestione di workflow basati sul modello a coreografia per Grid e Cloud

Giandomenico Spezzano

Giuseppe Papuzzo

RT-ICAR-CS-11-01 **Giugno 2011**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

LABORATORIO PUBBLICO PRIVATO OPENKNOWTECH

Progetto di Ricerca**: DM21301**

OpenKnowTech "*Laboratorio di Tecnologie per la Integrazione, Gestione e Distribuzione di Dati, Processi e Conoscenze*"

Obiettivo Realizzativo OR2

Open Grid

Deliverable D2.2

ALLEGATO D2.2_ALL_AT2.2.a

Un sistema autonomico per la gestione di workflow basati sul modello a coreografia per Grid e Cloud

Informazioni sul documento

Versione:	**1.0**
Data completamento:	**31/12/2010**
Preparato da:	**Giandomenico Spezzano, Giuseppe Papuzzo**
Approvato da:	
Stato	**FINALE**

SOMMARIO

1 Introduction

Workflow management systems (WMSs) are generally utilized to define, manage and execute workflow applications on Grid resources. WMSs must be dynamic and adaptive as Web/Grid services may be subjected to high variability in demand ands uffer from unpredictable peaks of heavy load and/or the resources availability can change over time while the workflow is executing.

When there are insufficient resources to meet the QoS requirement of the application, the system should provide additional computational resources. In these cases, the ability of temporarily extending the capacity of the Grid by renting resources from an external provider can be a viable proposition. Such an opportunity is offered by Cloud Computing which, by leveraging virtual machine technology, delivery IT infrastructure on demand on a pay per use basis. Clouds allow users to acquire and release services on-demand. These services can be Infrastructure as a Service (*IaaS*), Platform as a Service(*PaaS*) and Software asa Service(*SaaS*).

Next generation computing environments will benefit from the combination of Grid and Cloud paradigms providing frameworks that integrate traditional Grid services with on-demand Cloud services. This enable grid workflow systems to easily grow and shrink the available resource pool as the needs of the workflow change over time.A workflow running on a hybrid computing infrastructure should react to workload variations by altering its configuration in order to optimally use the Grids and Clouds available resources and recover from faults. Such modifications should happen automatically and without any human intervention. An autonomic WMS [1,6] exhibits the ability to reconfigure itself to the changes in the environment and can adapt the size tokeep the balance between servicing its workload with optimal performance and ensuring efficient resources allocation.

In this document, we present Sunflower a bio-inspired service-based framework for the execution of autonomic workflows that orchestrate Grid and Cloud services using a decentralized choreography model. Sunflower normally runs workflows using the grid resources but when the performance of web/grid services degrades and the QoS no longer meets the needs of a particular virtual organization or a particular member of a virtual organization, it acquires news hosts using a Cloud computing infrastructure and transfers the execution of the workflow on the Cloud infrastructure. To make the infrastructure as cost-effective as possible, Sunflower releases Cloud resources if the overall load returns to be normal. Sunflower monitors the system/application state and adapts the application and/or resources to respond to changing requirements or environment.To handle peak loads, Sunflower integrates the scheduling algorithm with a EC2-like provisioning component that can dynamically launch virtual machines in a Cloud infrastructure and deploys the required middleware components on-the-fly.

The remainder of this paper is organized as follows. Section 2 provides a description of the architecture for the decentralized execution of workflows in Sunflower according to choreography model. Section3 describes the Cloud-enabled workflow system. Section 4 describes preliminary experimental results and Section5 concludes the paper.

2 Sunflower framework

Sunflower is an adaptive p2p agent-based framework for configuring, enacting, managing and adapting autonomic workflows [?,5]. The novel aspect of Sunflower is that key functions, including resource allocation, are decentralized, which facilitates scalability and robustness of the overall system. The framework monitors the QoS for Web services and effectively self-adapts the workflow engine in response to changes in load patterns and server failures. Sunflower supports *service-level* parallelism usinga QoS-aware service-level schedule algorithm and *flow-level* parallelism by partitioning a workflow into sub-flows and running the sub-flows in multiple peers in parallel.

Workflows are described in Sunflower by the BPEL language [7] in order to exploit existing design tools. Sunflower replaces the standard BPEL engine with a new decentralized engine able to exploit the dynamic information available in the Grid and respond to the dynamic nature of the Grid ??. Decentralized compositions is implemented by coordinating BPEL processes runing on different organizational domains through a *choreography* model. BPEL programs are translated into a Petri Net (PN) for the coordination of the concurrent activities. This representation is then structurally decomposed into a set of distributed sub-flow schemas. On the basis of these schemas, Sunflower enacts a set of autonomic self-organizing workflow executors(*SWEA*)as a federation of agents [4] that interact with each other and with Antares[?,3], a P2P self-organizing Grid information system, to adapt the workflow to unforeseen circumstances. Antares provides a global virtual shared space that can be concurrently and associatively accessed by the *SWEA* agents to discovery and select the most appropriate services. Antares uses bioinspired algorithms to organize the construction of a Grid information system in which content, specifically metadata descriptors that describe the functional and non-functional characteristics of Grid services, is disseminated and logically reorganized on the Grid. Antares usesa number of ant-like agents (*AA*) that autonomously travel through P2P interconnections and use biased probability functions to: (i) replicate resource descriptors in order tofavor resource discovery; (ii) cluster resource descriptors with similar characteristics in nearby Grid hosts; (iii) foster the dissemination of descriptors corresponding to fresh (recently updated) resources and to resources having high Quality of Service (QoS) characteristics. Moreover, as descriptors are progressively reorganized and replicated, the Antares discovery algorithm allows query-ant agents (*QA*) to reach Grid hosts that store information about a larger number of useful resources in a shorter amount of time.

Based on dynamic service performance evaluation, the services with similar or same metrics are gather by Antares into clusters. As a service may join in or depart from Grid and service performance varies, and these variations may happen frequently, an adaptive and dynamic clustering method must be used. Antares uses an ant clustering algorithm that efficient satisfies these requirements.

Scheduling manager makes scheduling decision based on user QoS requirements and information in Antares. All member services in a cluster provide similar or same QoS after service clustering. Consequently, the task scheduling involves twosteps: initial cluster selection from service clusters and further service selection from the selected cluster.

Sunflower connects, in a transparent and fully automated manner, the *SWEA* agents

withWeb/Grid services to manage the invocation of the service and the execution of the workflow task assigned to that service. In Grid eachWeb/Grid service is bound to a single node for its entire lifetime and, moreover, the binding between grid service and its body is generally static. Sunflower assumes that multiple copies of a Web/Grid service, with different performance profiles and distributed in different locations, co-exist. During the execution of the workflow, if a service fails or becomes overloaded, a self-reconfiguring mechanism based ona *binding adaptation* model is used to ensure that the running workflow is not interrupted but its structure is adapted in response to both internal or external changes. *SWEA* agents adapt their structure moving over the grid to position themselves inthe nodes with low workload and where areavailable theWeb/Grid services with the best performance. The framework provides support for the migration-transparent of the agents, by a migration policy, to migrate in order to achieve goals like load balancing, performance optimization or guaranteeing QoS. Self-healing execution of a Grid workflow is achieved through service replacement in case of Web services failures. Furthermore, a backward recovery mechanism is used to bring the system from its erroneous state back into a previously correct state.

Figure 1 shows the architecture of the framework Sunflower. The workflow process is enacted by a set of cooperating Sunflower BPEL engines (SBE) instantiated at all participating nodes.

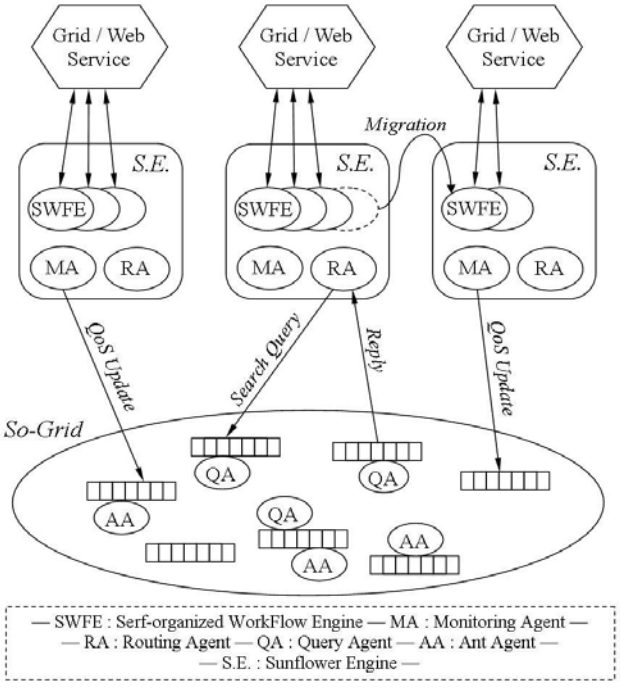

Figure 1. The Sunflower architecture.

A dynamic group of bio-inspired mobile *SWEA* agents representing the workflow executors generated from the user *workflow specification* are initially deployed on the basis of the workflow configuration. The workflow specification is the coordination model that describes how the generated agents cooperate with each other to reach a choreography execution. In Sunflower the workflow specification is obtained by the PN associatedtothe BPEL program. The decentralized execution of the workflowis coordinateby tokens exchanged among the SBE platforms. Tokens contain the whole execution state, including all data gathered during execution and when arrive on a SBE platform provide to execute the *SWEA* agents that perform the fragments of the Petri net and invoke the services. Each *SWEA* agent performs the portion of workflow assigned and determines which agent should be activated next.

To support workflow adaptation the SWEA agents are assisted by router RA agents and monitoring/analizing MA agents that interact with the Antares information system. The MA agent collects details about the performance metrics and workload of Grid ser-vice and when detects a change, due to external events, it inserts a new Web/Grid service descriptor with the new information in the Antares virtual space and notify the change to the RA agent. When the RA agent receives a notification about a modification of the class of QoS, it sends a query toAntares to discovery and select a descriptor of an equivalent optimal service. Then, Antares returns a reference to an endpoint handler for the selected service. The activities of the MA and RA agents are performed continuously. Before to execute the sub-workflow, the SWEA agent contacts the MA agent to verify if the class of QoS of the service to invoke is respected. In the affirmative case, the SWEA agent invokes the service and performs the workflow task, otherwise it uses its migration-policy to decide its actions looking up the RA agent.

3 The Cloud-enabled workflow system

In this section, we present the scheduling mechanism that extends Sunflwer to handle peak load situations generated by excessive demands due to the simultaneously access of many researchers or customers that share web/grid services in a Grid's virtual organization or use computational-intensive web services. To resolve peak load situations Sunflwer before attempts to use alternative grid services with the requested QoS and in case all services have a degraded QOS it uses additional computational resources provided on-demand by a cloud computing infrastructure. A schema of the Cloud-enabled Sunflwer architecture is shown in figur2. Sunflower seamlessly integrates dedicated grid resources and on-demand resources provided by the Eucalyptus's Cloud infrastructure and allows to invoke Web-Grid service simplemented according to theWSRF standard by GT4.

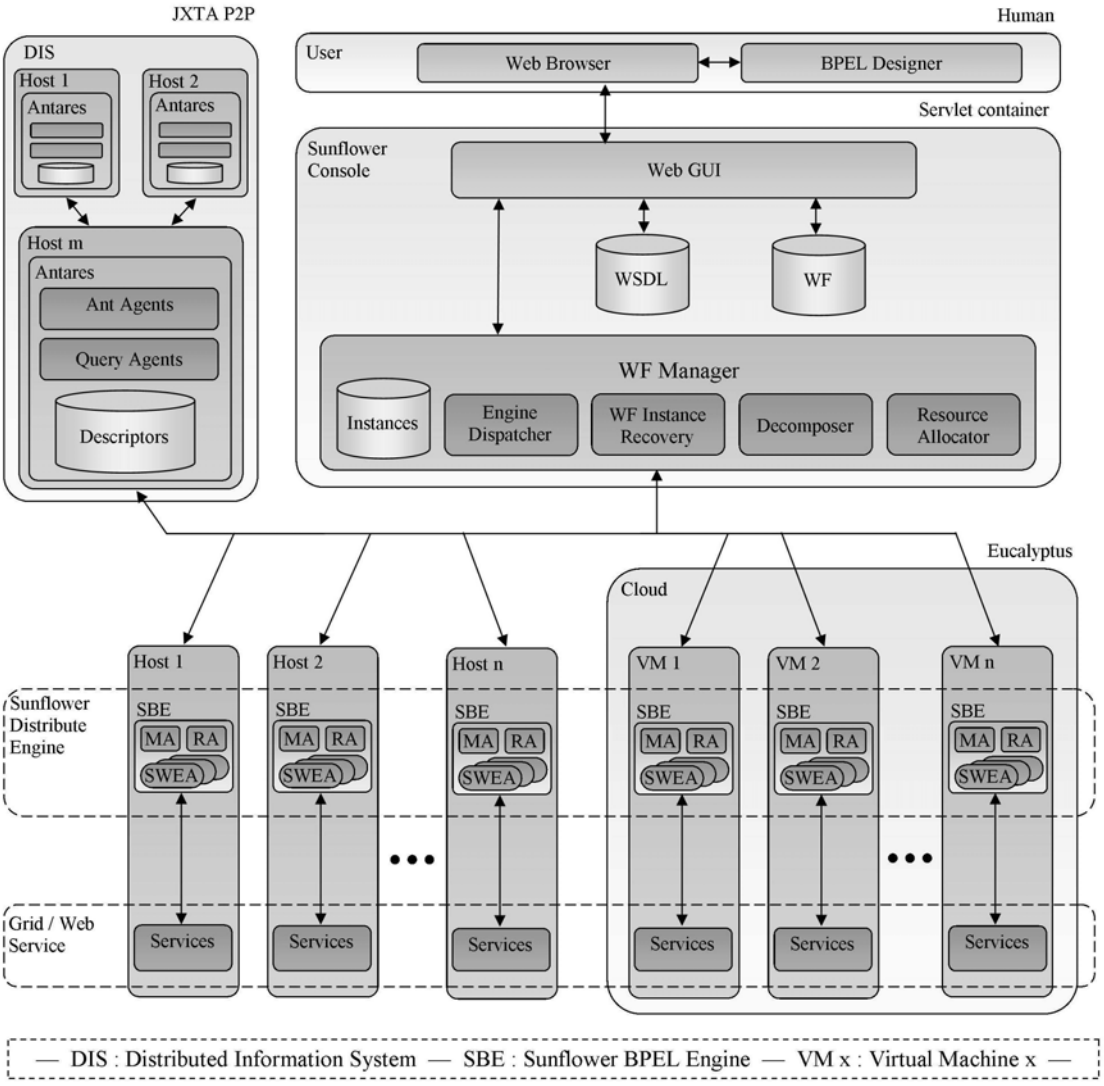

Figure 2. The Cloud-enabled Sunflower architecture.

The Sunflower Console provides the user's interface to model a workflow using the Eclipse BPEL Designer and generate the XML-BPEL code. Through the *decomposer* module the XML-BPEL code is partitioned into sub-workflow schema that are associated to the SWEA agents. The Sunflower Console also enables the deployment of the workflow according the initial configuration on the nodes of the Grid. During this phase the SWEA agents are queued on SBE nodes that contain theWeb services to invoke.

In order to mask the complexity of the underlying infrastructure two different environments must be installed. A standard Sunflower environment is installed on the Grid nodes. On the Cloud, Sunflower uses a Software as a Service (SaaS) delivery model which provides different customers the functionality of an application that is completely hosted in the cloud. The user does not have to worry about the required hardware resources, software components, deployment of the application, etc. Sunflower environment is installed on a running virtual machine (VM) and then the VM is

saved as an image that could be used to deploy future VMs. A convenient benefit of using the cloud is that once an image is developed, it is not necessary to reinstall software each time a new VM is launched. The configuration that is set up while a VM is running can be saved, and the image can be used to launch multiple copies of the original VM.

During the runtime, if the overall load is normal, Sunflower dynamically discovers the availableWeb/Grid services and performs a dynamic mapping of the SWEA agents to nodes of the Grid where are available the requested services with the suitable QoS. The SWEA agents migrate from one SBE queue to another SBE run queue if the current host providesaWeb/Grid service with a degraded QoS.

The Antares distributed information system manages all information about services available on the Grid. In Antares the use of shared metadata provides direct support for self-organizing systems. Metadata are permanently acquired, updated and monitored at runtime by both the system's components and the supporting infrastructure. Metadata is information about the running system. It is distinct from the code that implements component services. Metadata contain functional information and non-functional information (QoS, etc.)

When a peak load situation occurs the scheduling decision might involve starting new virtual machines by calling the provisioner that encapsulates interfaces to manage virtual machines in on-demand infrastructure likeAmazon EC2. The provisioner provide general abstractions to upload virtual machine images, download, modify, delete, or save copies of preexisting images, deployimages as virtual machine.

Sunflower uses the scheduling algorithm outlined in figure 3, executed by RA agents, to make these choices, using information provided by Antares. For eachWeb/Grid service, the RA agent schedules the SWEA agents queued in the local SBE applying a dynamic mechanism that describes a very simple pattern that allows a dynamic adaptation in response to component availability falling below a specified QoS threshold.

When the QoS of the service relied on a node of the grid is less than that required a request is sent to the Antares registry service to search for an equivalent service to replace. If the service exists and is available, the reference to the service is returned to the agent RA that uses to migrate a SWEA agent on the node where the service is localized. Otherwise, if there are not services available an activation request of a new virtual machine is sent to the provisioner. Before that, a new VM is started the provisioner checks if one VM with the QOS requested already exists else a new VM is started. The VM continues the execution of the workflow and before toinvokenextWeb/Grid service on the Cloud the RA agent checks, by querying Antares, if an equivalent service is available on the Grid. In case affermative, the activation token with the status information is transferred to the SBE node on the grid that has the next service to invoke.

```
1.  foreach SWEA agents in queue on SBE
2.   {
3.       if  (SWEA.ReqService.QoS > SBE.CurrentQoS)
4.        {
5.          AlterSBE = Antares.SearchQuery(SWEA.ReqService.ServiceType,
                          SWEA.RequiredService.QoS);
6.              if (AlterSBE != null)
7.               {
8.                   SBE.Routing(SWEA, AlterSBE);
9.                }
10.             else
11.             {
12.           AlterSBE = ec2-describe-instances(SWEA.ReqService.VMDiskImage,
                              SWEA.ReqService.QoS)
13.             if  (AlterSBE == null)
14.             {
15.           AlterSBE = ec2-run-instances(SWEA.ReqService.VMDiskImage,
                                  SWEA.RequiredService.QoS)
16.              }
17.              SBE.Routing(SWEA, AlterSBE);
18.         }
19.  }

20. }
```

Figure 3. The RA scheduling algorithm.

4 Experimental results

We conducted the scalability experiments using 5VMs with different size and we evaluated the overhead introduced by Cloud infrastructure to execute a workflow in Sunflower. The table 1 describes in detail the VMs defined. We used 2VMs with 1 CPU, 2 VMs with 2CPUs and 1VM with 4 CPUs.

VM name	CPUs	Memory (MB)	Disk space(GB)
m1.small	1	128	2
c1.medium	1	256	5
m1.large	2	512	10
m1.xlarge	2	1024	20
c1.xlarge	4	2048	20

Table 1. The type of VM used in the study.

The performance of our load balancing algorithm has been evaluated using a geoworkflow which is apt to simulate many complex real world geo-hazard cellular automata (CA) models as landslide evolution, lava flows, floods, etc. Geo-workflows greatly simplify the process of conducting geophysical analyses and forecasts. Geo-workflows are tools for designing and conducting computational experiments. Scientist need to be able to run analysis processes over collected data. Often these analysis processes are single computations and often they are complex composed scenarios of preprocessing, analysis, post processing and visualization. These experimental geo-workflows are often repeated hundreds of times with slightly different parameter settings or input data. The user can decrease the execution times if more geo-workflows can be launched in parallel.

Multiple geo-workflows, up to seven, has been run concurrently. New virtual machines were booted, due to high load of existing machines, when the number of concurrent workflows was greater than three. The Sunflower workflow engine and all virtual machines are hosted by Eucalyptus.

Figure 4 shows the execution time when the visualization service is invoked from a workflow. Note that the scheduling algorithm allocates to the workflow the service on the machine with the higher performance. When more of three workflows are instantiated a new virtual machine is booted to handle the peak load. For this measurement, we used the medium host type available on the Cloud infrastructure. Such a "medium instance" has 1 CPU, 256MB of RAM, 5GB hard disk and runs a 32 Bit Linux.

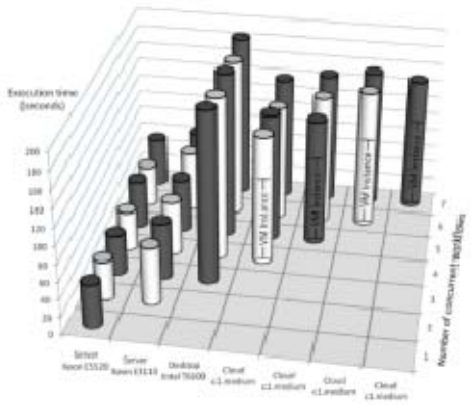

Figure 4. Execution times for the execution of multiple workflow instances.

Multiple workflows has been run more than 30 times. The average runtime measurements including the startup of new virtual machines are shown in table 2.

peer	xeon E5520	xeon E3110	intel t6600	c1.medium	c1.medium	c1.medium	c1.medium
1	49.703	-	-	-	-	-	-
2	45.828	66.578	-	-	-	-	-
3	46.047	65.39	195,953	-	-	-	-
4	45.781	62.375	186.313	146.828	-	-	-
5	54.984	61.438	189.859	146.344	143.578	-	-
6	46.109	67.531	180.25	130.546	146.578	156.578	-
7	54.954	66.484	185.781	140.546	149.39	156.438	150.578

Table 2. Execution times of multiple workflows including the startup of new virtual machines.

5 Conclusions

In this paper, we have presented a decentralized and cooperative strategy for the execution of autonomic workflows on heterogeneous distributed platform, such as Grids and Clouds.Abio-inspired cross-layer approachisusedto supportthe self-adaptationofthe workflow enactment engines to changes in the Grid and fulfill QoS requirements for Web/Grid services.To handle peak loads, Sunflower integrates the scheduling algorithm with a provisioning component that can dynamically launch virtual machines in Eucalyptus's Cloud infrastructure and deploys the required middleware components on-thefly. The overhead introduced from the Cloud infrastructure have been evaluated for the execution of a geo-workflow.

References

[1] M. Rahman and R. Buyya : An Autonomic Workflow Management System for Global Grids, *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008)*, IEEE CS Press, pp.578-583, 2008.

[2] Bonabeau E., Dorigo M., Theraulaz G. : Swarm Intelligence: From Natural to Artificial Systems New York, *NY: Oxford University Press*, Santa Fe Institute Studies in the Sciences of Complexity 1999.

[3] Forestiero A., Mastroianni C., and Spezzano G. : So-Grid: A Self-organizing Grid Featuring Bio-inspired Algorithms, *ACM Transactions on Autonomous and Adaptive Systems*, vol. 3 n. 2, 2008.

[4] Brazier, F.M.T., Kephart, J.O., Van Dyke Parunak, H. Huhns M.N. : Agents and Service-Oriented Computing for Autonomic Computing: A Research Agenda, *IEEE Internet Computing*, vol. 13 n. 3, pp. 82-87 2009.

[5] Vidal, J.M., Buhler, P., Stahl, C.: Multiagent systems with workflows, *IEEE Internet Computing*, vol. 8 n. 1, pp. 76-82 2004.

[6] Kephart Jeffrey O., Chess David M. : The Vision on Autonomic Computing, IEEE Computer, vol. 36, n. 1, pp.41-50, 2003.

[7] Web Services Business Process Execution Language Version 2.0 - Committe Specification. Technical Report, OASIS, Jan. 2007.

[8] W. van der Aalst and K.M. van Hee: Workflow Management: Models, Methods, and Systems, *MIT Press*, Cambridge, MA, 2002.

6 Appendice – Manuale Sunflower

6.1 *Installazione della Sunflower Console e del Sunflower Distribute Engine*

La Sunflower Console (SC) consente di gestire l'intero ciclo di vita delle istanze di workflow, dalla fase di creazione fino alla visualizzazione del risultato. Mentre il Sunflower Distribute Engine (SDE) si occupa dei eseguire le istanze di workflow in coreografia.
Prima di installare la SC su un PC è necessario soddisfare i seguenti prerequisiti:

1. Installazione della Java JDK 6 nella cartella /var/lib/jdk1.6.X_XX e settaggio delle relative variabili d'ambiente JAVA_HOME e PATH .
   (http://www.oracle.com/technetwork/java/javase/downloads/index.html)
2. Installazione di Apache Tomcat 6.0 nella cartella /var/lib/apache-tomcat-6.0.XX e settaggio della relativa variabile d'ambiente CATALINA_HOME .
   (http://tomcat.apache.org/)
3. Installazione di MySQL da package scaricato dal sito MySQL o con il gestore dei pacchetti APT o YUM .
   (http://www.mysql.com/)

Per installare la SC:

4. Scaricare il file Sunflower_1.0.zip dalla sezione download e copiare il file "SunflowerConsole.war" in esso contenuto nella cartella "$CATALINA_HOME/webapps". Dopo pochi secondi inizierà il deploy automatico del file *.war, nel caso ciò non accada riavviare il server Apache Tomcat.

Terminata la fase di installazione della SC è possibile accedervi browser internet all'indirizzo : http://localhost:8080/SunflowerConsole/ . La figura 1 mostra la GUI della SC attraverso cui è possibile effettuare le successive fasi di installazione e configurazione del SDE.

Figura 1: GUI di configurazione di Sunflower.

Selezionando la scheda "Download SDE" dalla SC è possibile scaricare il software e la documentazione necessaria per installare il SDE. In questa scheda inizialmente troveremo abilitato per il download solo il link "Sunflower Distribute Engine For Sunflower Console" (SDE4SC), mentre il secondo link "SunflowerDistributeEngine4Services.zip" (SDE4S) risulterà disabilitato fino a quando non verrà installato e configurato SDE4SC sulla macchina dove è presente la Sunflower Console. Nella scheda sono indicati i passi di installazione per il SDE4SC:

- Passo 1:
  Scaricare il "SunflowerDistributeEngine4SunflowerConsole.zip" in una cartella temporanea.

- Passo 2:
  Estrarre il contenuto del "SunflowerDistributeEngine4SunflowerConsole.zip" in "/var/lib/SunflowerDistributedEngine" ed eseguire "startup.sh" in essa contenuta.

SDE4SC come prerequisito richiede Java JDK 6, ma essendo installato sulla stessa macchina dove è presente la SC questo risulta già soddisfatto. La prima volta che si avvia il SDE4SC, questo creerà automaticamente i file di configurazione necessari per attivare il download del SDE4S.

Il SDE4SC fungerà da bridge di comunicazione tra la SC e i SDE4S che verranno installati successivamente, per questa ragione è necessario caricare prima di attivare il download del SDE4S i dati relativi all'end-point del SDE4SC.

Ritornando nella scheda "Download SDE" sarà ora possibile scaricare il "Sunflower Distributed Engine 4 Service". Questo va installato sui PC dove sono presenti i Web/Grid Service che si vuole utilizzare nell'esecuzione dei workflow. Nella scheda sono indicati i passi di installazione per il SDE4S (prerequisito, installazione della Java JDK 6 come al passo 1 dei prerequisiti della SC):

- Passo 1:
  Scaricare o copiare il file "SunflowerDistributeEngine4Service.zip" in una cartella temporanea.

- Passo 2 : (per sistemi linux)
  Estrarre il contenuto del "SunflowerDistributeEngine4Services.zip" in "/var/lib/SunflowerDistributedEngine/" ed eseguire "startup.sh" in essa contenuta.

- Passo 2 : (per sistemi windows)
  Estrarre il contenuto del "SunflowerDistributeEngine4Services.zip" in "C:\SunflowerDistributeEngine\" ed eseguire "startup.bat" in essa contenuta.

Il SDE è basato sul piattaforma P2P JXTA, in caso di problemi di configurazione e firewall fare riferimento alla documentazione contenuta al seguente indirizzo https://jxta.dev.java.net .

Al termine dell'installazione, della SC e del SDE4SC, sarà possibile eseguire alcuni workflow di test presenti nella scheda "Execute Workflow" della SC. Questi workflow di prova non utilizzano alcun servizio ma servono solo a testare la corretta installazione del sistema. A questo punto per poter utilizzare workflow più complessi che richiedo l'impiego di servizi, è necessario come primo passo associare tali servizi al framework Sunflower. Per far ciò è necessario che i WSDL dei servizi, che useremo per la creazione dei workflow astratti con "Eclipse BPEL Designer", siano presenti nel WSDL Repository della SC. La prossima sezione mostrerà i passi da eseguire per effettuare la registrazione dei servizi reali come WSDL astratti nel WSDL Repository.

N.B.: Nel primo accesso alla scheda "Execute Workflow" verrà richiesto di inserire la password dell'utente "root" del database "MySQL", dopo averla inserita la SC configurerà automaticamente MySQL con tutti i database e tabelle di cui necessita.

6.2 *Deposito dei WSDL astratti*

L'idea di base del WSDL Repository è quelle di fornire gli strumenti necessari per gestire, in modo semplice e intuitivo, un UDDI locale di servizi per Sunflower. Attraverso la scheda "WSDL Repository" della SC riportata in figura 2, gli utenti possono riorganizzare in categorie e registrare i prototipi dei servizi che intendono usare nella creazione dei workflow BPEL.

Per aumentare il livello di astrazione rispetto ad un classico UDDI si è scelto di utilizzare il paradigma della *struttura a directory*, così da utilizzare la tipica struttura gerarchica ad albero per facilitare l'organizzare in categorie dei prototipi dei servizi. Attraverso questa visione, il nome di ogni directory esprime un concetto, che diventa sempre più specifico quanto più si avvicina ad una foglia. Mentre il nome di ogni file rappresenta un concetto finale, non scomponibile ulteriormente, concretizzato come prototipo di servizio alias WSDL astratto. Infine, nella nostra visione, il contenuto di un file foglia è il descrittore del servizio in *formato WSDL* secondo lo standard W3C (http://www.w3.org/TR/wsdl).
Nel WSDL Repository una volta prescelto un ramo/directory/concetto o una foglia/file/WSDL, su di esso si potranno eseguire le quattro operazioni fondamentali di creazione e modifica:

1. "Add Dir", aggiunge una directory/concetto nella posizione corrente.
2. "Add WSDL", aggiunge una foglia/WSDL astratto nella posizione corrente.
3. "Del Dir", cancella la directory corrente.
4. "Del WSDL", cancella il file corrente.

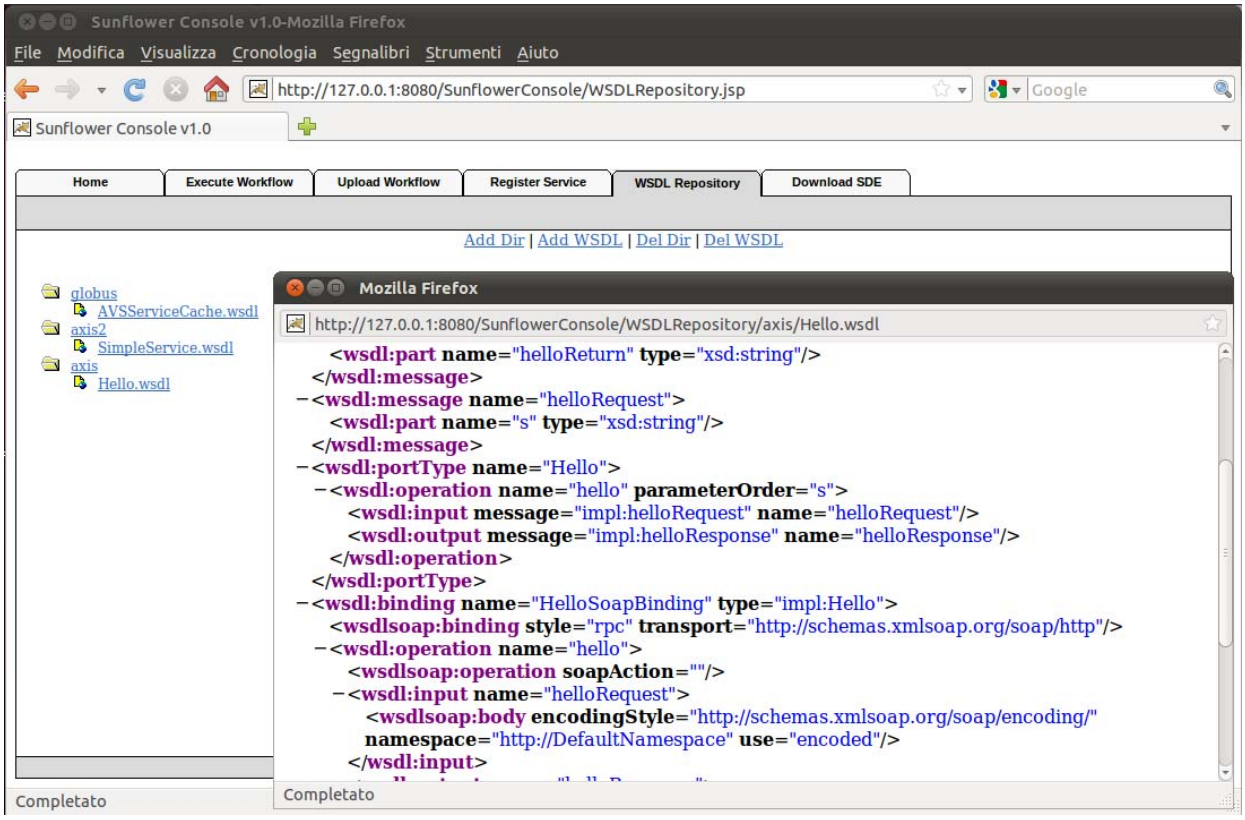

Figura 2: WSDL Repository.

La struttura dati così creata non fa riferimento a nessun servizio realmente implementato, ma crea la struttura dove andare a classificare e registrare i servizi creati. Un utente che vuole creare un servizio, deve prima di tutto selezionare una foglia da cui ricavare il WSDL, in secondo luogo implementare il servizio in base alle specifiche del WSDL, in fine registrare il servizio creato alla relativa classe di servizio. Per semplificare le operazioni di creazione del WSDL astratto il metodo *Add WSDL* importa automaticamente il *WSDL di un servizio reale* come *WSDL astratto*, e richiede

come parametri l'URL del WSDL servizio reale da importare e il nome da assegnare alla foglia nell'ontologia a directory. Questo passaggio consente di creare, partendo da un WSDL di servizio reale, un prototipo di WSDL di servizio astratto. Questo prototipo di servizio verrà successivamente usato per registrare il servizio reale da cui è stato generato e tutti servizi ad esso equivalenti al medesimo WSDL astratto. Inoltre, nella fase di creazione di un workflow BPEL attraverso Eclipse BPEL Designer, questi WSDL astratti verranno usati come sostituti dei servizi reali così da creare dei *workflow astratti* basati su servizi astratti.

6.3 *Registrazione di un servizio reale*

I WSDL astratti contenuti nel Repository consentono la creazione di workflow astratti, tuttavia tali informazioni non sono sufficienti per l'esecuzione del workflow. Nelle varie fasi del ciclo di vita di un workflow, infatti, sono necessarie informazioni riguardanti servizi reali e peer.
La registrazione di un servizio reale e del suo nodo SDE di competenza è completamente automatizzata, e richiede semplicemente la compilazione dei campi di una Form accessibile nella scheda "Register service" della SC (vedi figura 3).

La Form contiene tre campi che vanno a formare il contenuto di una tupla che lega assieme un servizio reale, un servizio astratto e un nodo SDE.

- URL del WSDL relativo al servizio reale da registrare
  (es.: http://150.145.63.130:8080/axis/DataServerCache.jws?wsdl).
- URL del WSDL astratto nel WSDL Repository, da cui è stato generato il sevizio reale
  (es.:
  http://127.0.0.1:8080/SunflowerConsole/WSDLRepository/Dati/DataServerCache.wsdl).
- Locazione del file "client.adv", generato al primo avvio del nodo SDE sul PC su cui risiede il servzio reale
  (es.: /var/lib/SunflowerDistributeEngine/client.adv per linux,
  C:\ SunflowerDistributeEngine \client.adv per windows).

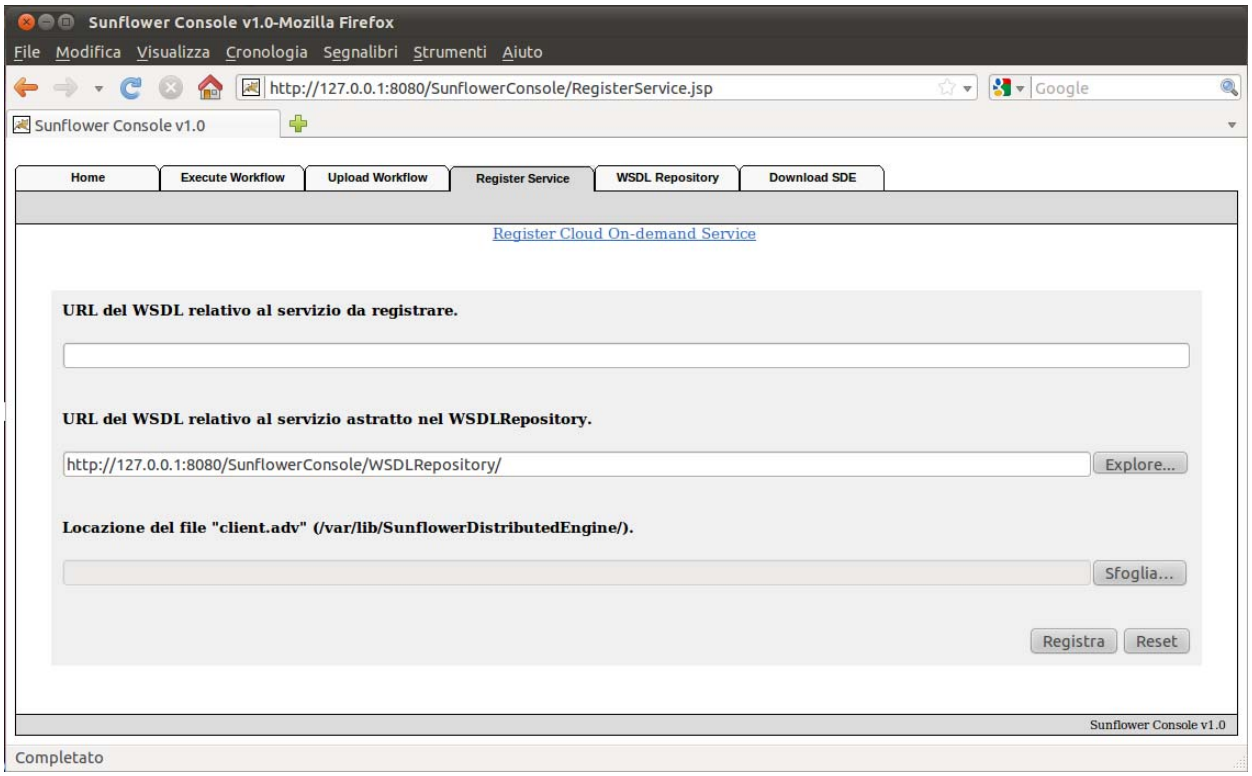

Figura 3: Registrazione dei servizi reali e associazione al relativo servizio astratto e Peer.

Naturalmente il nodo del SDE da associare al servizio reale è quello presente sul PC dove risiede il servizio, in modo da garantire al SDE il monitoraggio del sistema e il rispetto della QoS imposta dall'utente. Si possono verificare però casi particolari in cui il nodo SDE registrato non sia quello dove risiede il servizio, in questi casi il sistema non può garantire il monitoraggio e la QoS ma garantisce solo l'esecuzione. Per maggiori dettagli su questi casi particola far riferimento alla documentazione.
N.B.: Il contenuto del link "Register Cloud On-demamd Service" verrà discusso in seguito.

6.4 *Creazione e deploy di un workflow BPEL*

La creazione dei workflow BPEL avviene mediante l'impiego del plug-in "Eclipse BPEL Desiger", sviluppato per Eclipse nel progetto "BPEL Designer Project". Grazie a questo designer è possibile creare codice in base allo standard XML-BPEL senza dover toccare direttamente codice XML, componendolo attraverso operazioni di drag & drop e configurando le singole attività mediante wizzard.

Prima di procedere all' installazione dell'Eclipse BPEL Designer è necessario soddisfare i seguenti prerequisiti:

1. Installazione della Java JDK 6 nella cartella /var/lib/jdk1.6.X_XX e settaggio delle relative variabili d'ambiente JAVA_HOME e PATH .
   (http://www.oracle.com/technetwork/java/javase/downloads/index.html)
2. Installazione di Eclipse Classic 3.6.X nella cartella /home/$USER/eclipse .
   (http://www.eclipse.org/downloads/)

Per installare Eclipse BPEL Designer 0.5 :

3. Avviare Eclipse e selezionare "Help" → "Install New Software" , nella finestra di dialogo inserire nella casella "Work with" il seguente URL http://download.eclipse.org/technology/bpel/update-site , selezionare il plug-in BPEL per l'installazione e cliccare su "Next", per completare l'installazione seguire le istruzioni della finestra di dialogo.

Terminata la fase d'installazione, Eclipse con il plug-in BPEL Designer si presenterà con l'interfaccia grafica (perspective) riportata in figura 4.

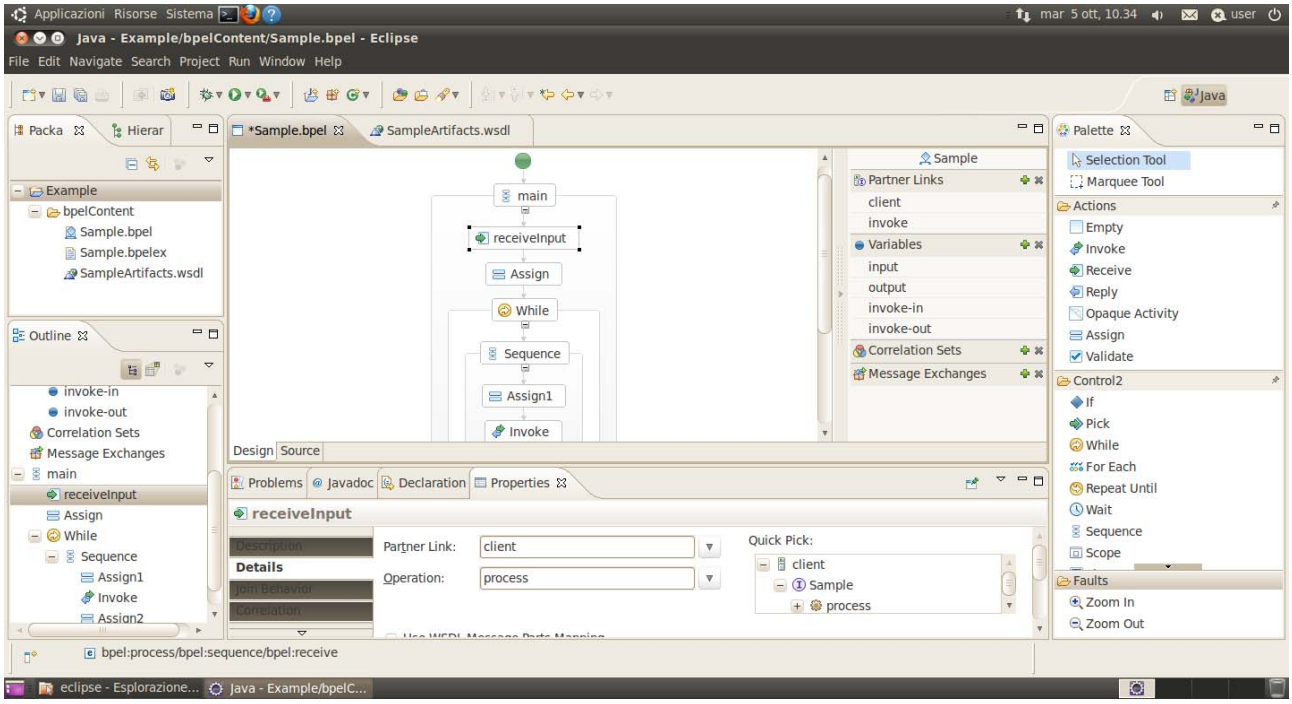

Figura 4: Eclipse BPEL Designer.

All'URL http://www.eclipse.org/bpel/ sono disponibili la documentazione, il tutorial e i video tutorial per la creazione di workflow BPEL. Per rendere il workflow creato compatibile con Sunflower, bisogna discostarsi leggermente da tale documentazione, più precisamente nel punto raggiungibile al seguente link http://www.eclipse.org/bpel/users/howto/wsdl.php o selezionando "Users" dal menu di sinistra e poi "Add a WSDL" dalla scheda dei contenuti. Nel punto evidenziato in figura 5, è possibile importare un qualsiasi WSDL di servizio per la creazione di "Partner Link".

Nel nostro caso i WSDL dei servizi da usare nei workflow BPEL devono essere selezionati tra quelli disponibili nel WSDL Repository che contenente i WSDL dei servizi astratti, accessibile all'indirizzo http://localhost:8080/SunflowerConsole/WSDLRepository.jsp .

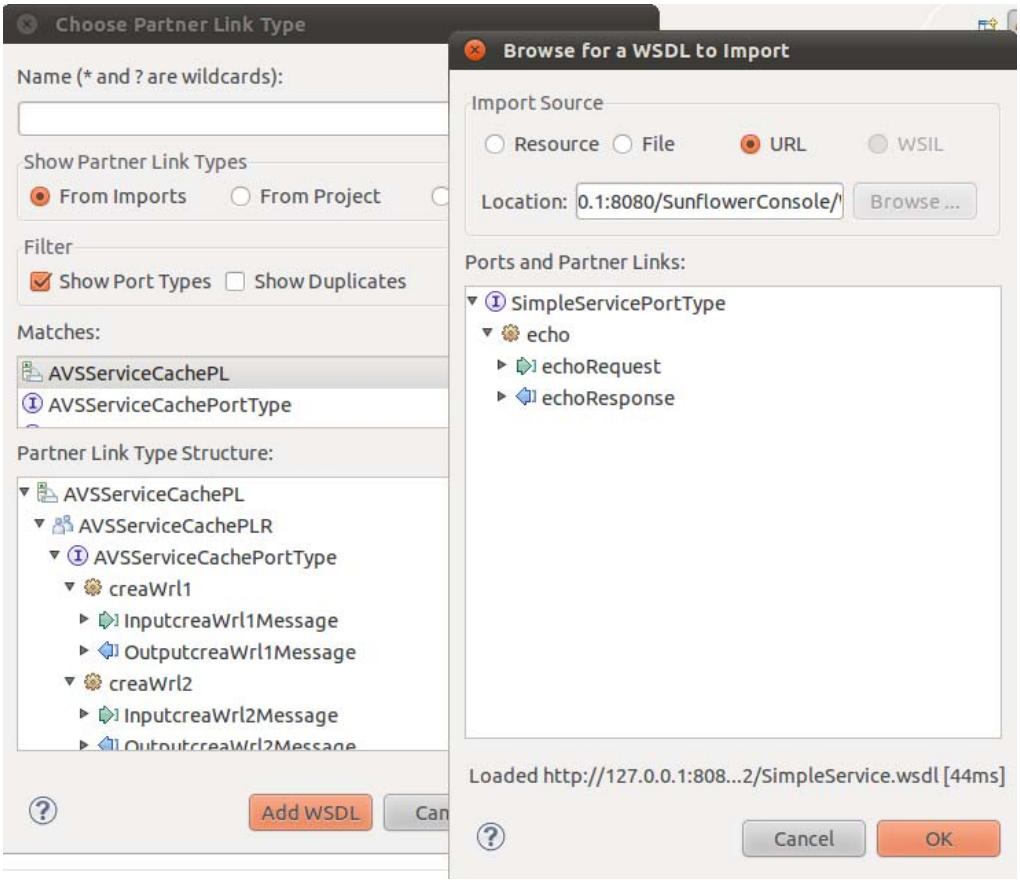

Figura 5: Creazione di un Partner Link che utilizza un WSDL astratto.

Per poter eseguire il deploy del workflow, creato con Eclipse BPEL Designer, questo va opportunamente inserito in un package per predisporlo alla fase di upload. Per far ciò in Eclipse si seleziona "File" → "Export" → "Archive file" → selezionare la cartella del progetto BPEL contenete il workflow creato → selezionare un nome per l'archivio da esportare → "Save in zip format".

Dopo aver creato il package contenete il workflow BPEL è necessario eseguire il deploy in modo renderlo disponibile per l'esecuzione nell'elenco di "Execute Workflow", ciò può essere fatto accedendo alla scheda "Deploy Workflow " e seguendo le istruzioni in essa contenute (figura 6).

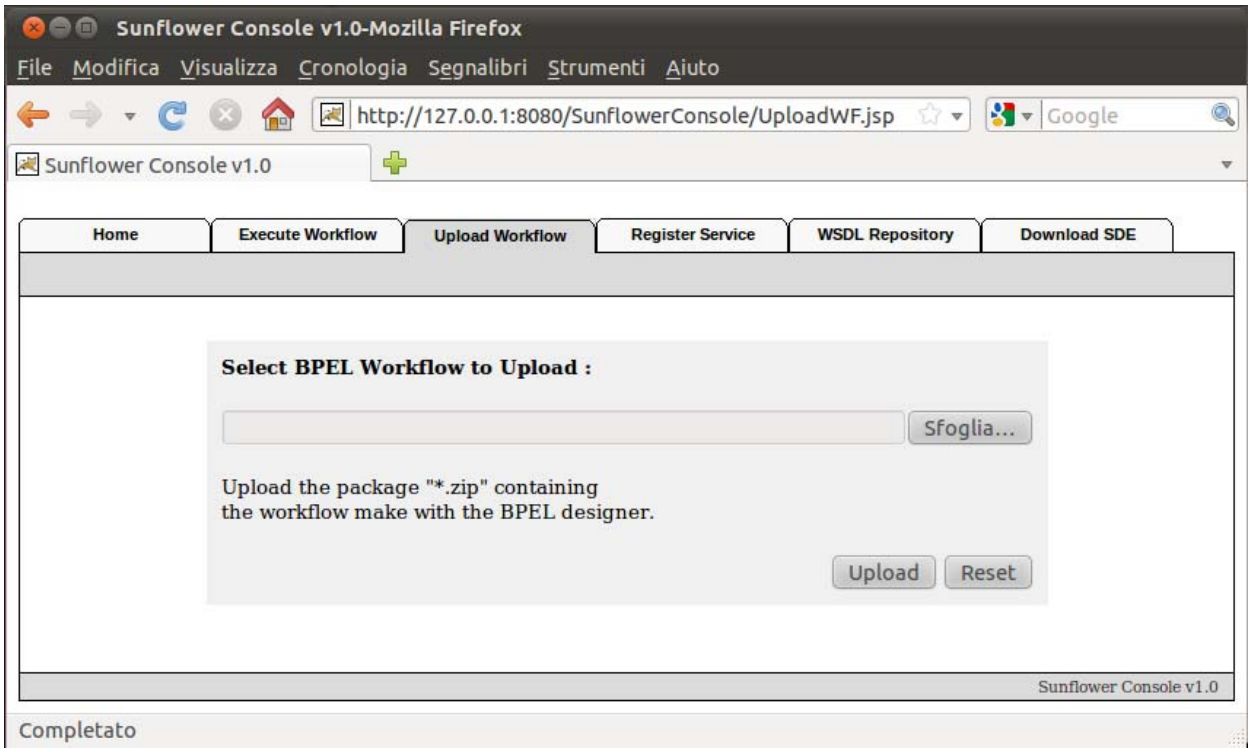

Figura 6: Upload e Deploy di un workflow BPEL.

Nella fase di upload il package contenete il workflow è inviato alla SC, mentre nella fase di deploy si effettua la verifica il contenuto del package. Se questo contiene un workflow BPEL, verrà inserito nell'elenco dei workflow disponibili, altrimenti sarà restituito l'errore.

6.5 *Inizializzazione ed esecuzione di un workflow BPEL*

Tutti i workflow BPEL per i quali la fase di Upload e Deploy è andata a buon fine, vengono aggiunti alla lista presente nella scheda "Execute Workflow"(figura 7). Qui è possibile selezionare il workflow da eseguire in base al nome assegnato al package esportato da Eclipse.

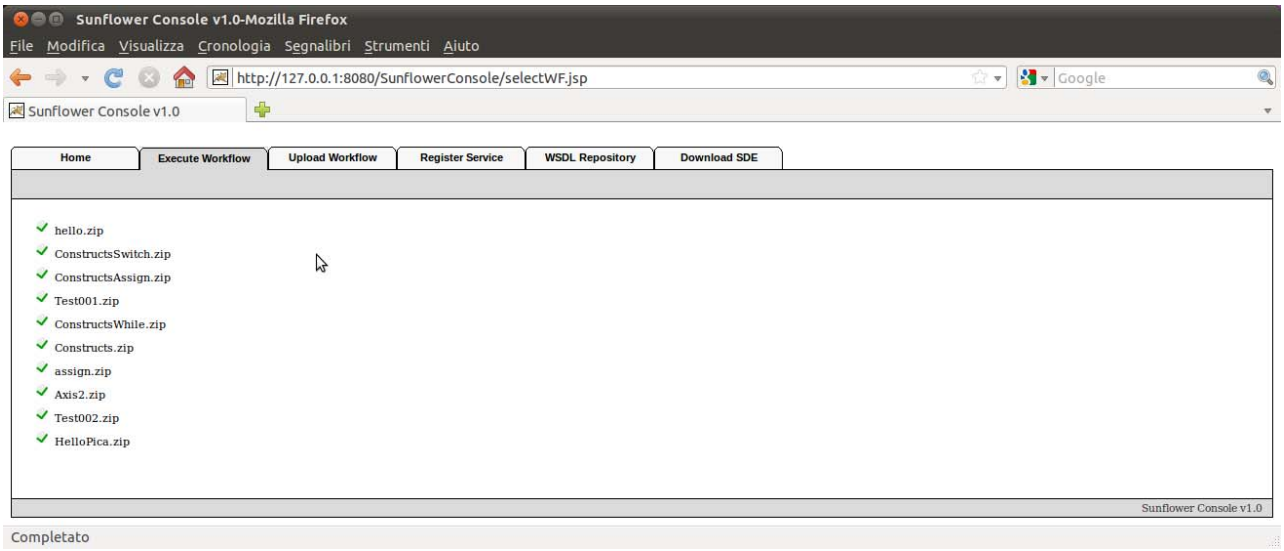

Figura 7: Elenco dei workflow BPEL disponibili per l'esecuzione.

Selezionato un workflow BPEL l'inizializzazione, dei parametri relativi alle variabili di input e agli attributi che descrivono la QoS, può essere eseguita attraverso la compilazione della form mostrata in figura 8. Nella parte superiore della form sono presenti le caselle relative alle variabili di ingresso del workflow che devono essere inizializzate con i valori che si vogliono assegnare alle variabili. In basso è presente una tabella che riporta per riga i servizi coinvolti nel workflow e per colonna i vincoli di QoS da rispettare. Gli attributi attualmente presenti per descrivere la QoS sono: frequenza della CPU, percentuale di utilizzo della CPU, memoria fisica disponibile e memoria libera disponibile. Tali attributi riportano per ogni servizio un valore di default, che può essere modificato in base alle esigenze di esecuzione con valori scelti dall'utente, ma sempre con valori compatibili alle caratteristiche presenti nelle macchine disponibili. Associato ad ogni servizio è disponibile una check-box che permette di rendere il servizio vincolato o non vincolato ad un nodo SDE (Switch/No Switch). Nel caso di nodo vincolato è possibile usare solo quel servizio e non copie replicate, quindi la sostituzione del servizio in caso di QoS bassa è disabilitata. In tal caso il workflow userà il servizio anche se la sua QoS non è rispettata. Questa caratteristica rende il sistema compatibile con servizi state-full, dove parte dell'elaborazione parziale è memorizzata nel servizio stesso. Da notare che solitamente il risultato parziale del workflow è conservato nel Token e nelle cache dei servizi.

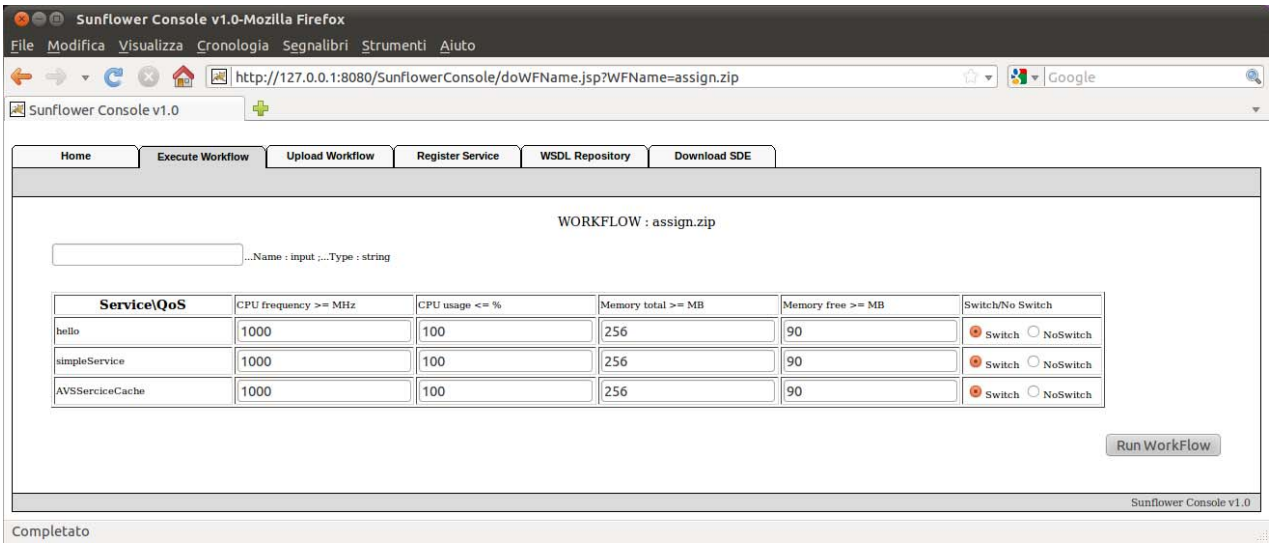

Figura 8: Form per l'inserimento dei parametri relativi alle variabili e alla QoS.

Terminata la fase d'inizializzazione del workflow è possibile avviarne l'esecuzione mediante il bottone "Run workflow". Prima di iniziare l'esecuzione vera e propria, il codice XML-BPEL presente nel package è tradotto in base al modello a coreografia basato su Rete di Petri. Successivamente, ogni frammento di codice BPEL è assegnato a un nodo SDE in base al servizio reale scelto per concretizzare del workflow astratto, ed il Token è inizializzato con i parametri di input del workflow. In fine, Token e frammenti XML-BPEL sono inviata ai rispettivi nodi SDE per l'esecuzione.

6.6 *Monitoring e visualizzazione del risultato*

Sunflower fornisce un'interfaccia grafica che consente di monitorare l'esecuzione del workflow attraverso due rappresentazioni grafiche, il primo è quello definito con Eclipse BPEL Designer e il secondo quello è quello risultante dalla decomposizione per l'esecuzione in coreografia (vedi figura 9). Nel riquadro di sinistra è riportato il workflow completo generato dal codice BPEL classico. Nel riquadro di destra sono mostrati i nodi con i relativi frammenti di codice BPEL che devono eseguire in coreografia. Il flusso di esecuzione del workflow può essere seguito attraverso i simboli che appaiono di volta in volta sui vari costrutti BPEL, il cui significato è riportato di seguito:

- Il pallino arancione, rappresenta il Token ed indica l'ultima operazione eseguita correttamente.
- Il pallino rosso, indica un errore irreversibile che richiede la procedura di "recovery del workflow", dopo pochi secondi il monitoring verrà ridiretto sulla nuova istanza di workflow per continuare l'esecuzione.
- Una "R" rossa sui costrutti di "invoke", indica che il servizio associato all'invoke non era in grado di rispettare la QoS richiesta ed è stata eseguita con successo la procedura di Routing per sostituirlo.

- Le lettere “RC” rosse sul costrutto di “invoke”, indica che il servizio associato all’invoke non era in grado di rispettare la QoS richiesta ed è stata eseguita con successo la procedura di Routing per sostituirlo, ma l’unico servizio disponibile era sul “Cloud”.

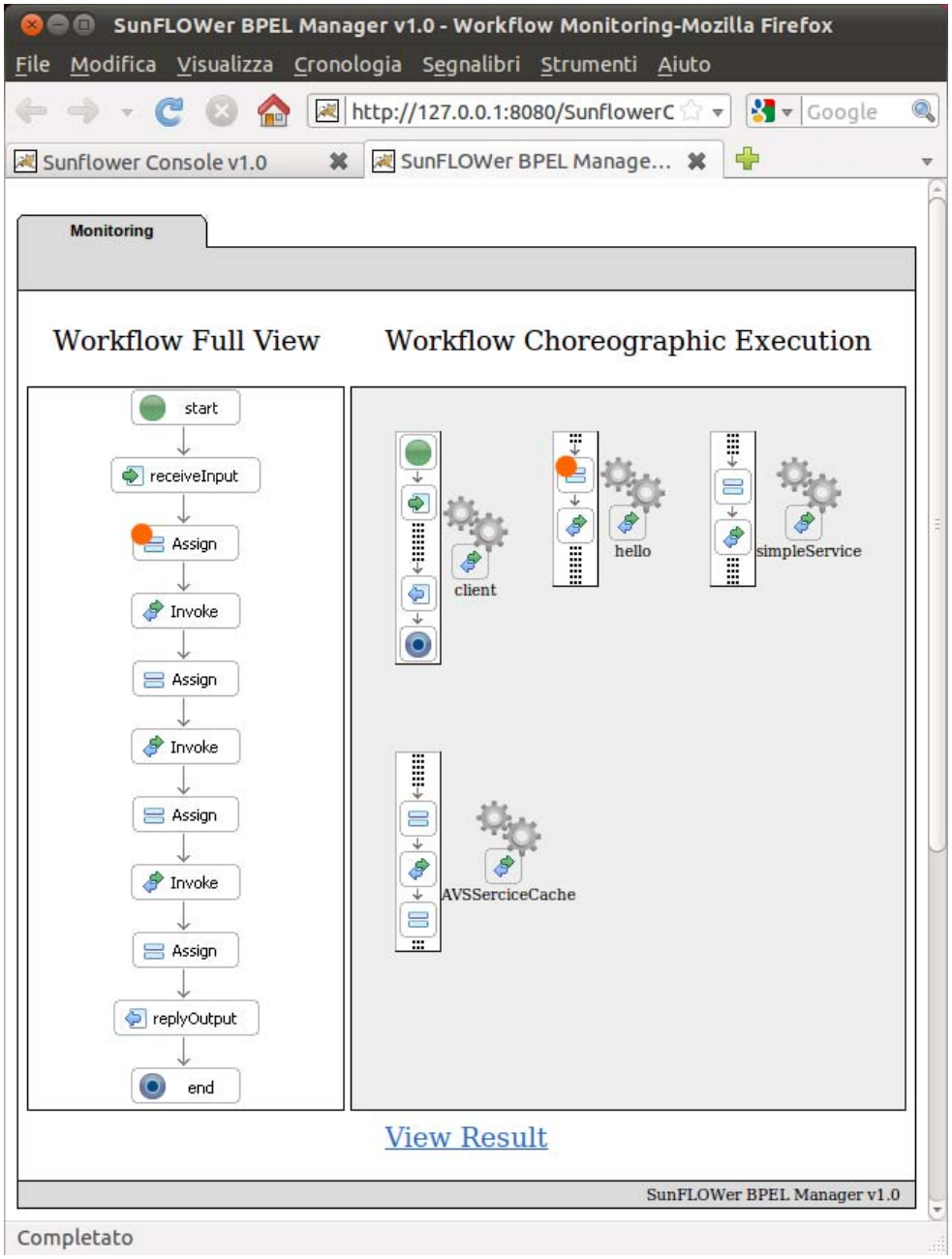

Figura 9: Monitoring di un workflow in Sunflower.

Quando il Token raggiunge il costrutto “end”, l’esecuzione del workflow BPEL è completa ed è possibile visualizzare il risultato cliccando sul link “View Result”. Il risultato verrà visualizzato in base alle impostazioni predefinite del browser, mentre nel caso in cui il risultato sia un URL il

monitoring ridirigerà la connessione verso URL remoto per poi visualizzarlo sempre in base alle impostazioni del brower.

6.7 *Sunflower & Eucalyptus Cloud*

Sunflower, grazie ad algoritmi di scheduling statico e dinamico basati su QoS è in grado di bilanciare il carico di lavoro dei workflow BPEL distribuendolo sui vari SDE. Sfortunatamente, per quanto si possa ottimizzare l'allocazione sui servizi reali, nel caso di colli di bottiglia e picchi di carico se il numero di servizi non è sufficiente per soddisfare le richieste, il sistema comincerà a rifiutare ulteriori workflow e a terminare con errore quelli già in esecuzione. Per superare questi picchi di carico e colli di bottiglia in tempi rapidi e a basso costo, si è pensato di integrare in Sunflower un meccanismo di "service on-demand" basato su "Eucaluptus Cloud" (http://www.eucalyptus.com/).

Prima di creare un service on-demand per Sunflower è necessario soddisfare i seguenti prerequisiti:

1. Installazione di Eucaliptus 2.0.2 e Euca2ools 1.3.1 .
   (http://open.eucalyptus.com/downloads)
2. Studio del tutorial "Eucalyptus Getting Started" e "Eucalyptus Image Management" per acquisire i concetti e la nomenclatura usati successivamente in questo documento.
   (http://open.eucalyptus.com/wiki/EucalyptusGettingStarted_v2.0
   http://open.eucalyptus.com/wiki/EucalyptusImageManagement_v2.0)

Per creare un service on-demand per Sunflower bisogna:

3. Selezionare e installare tra le "Images" di virtual machine (VM) presenti nello "Store" quella più adatta al tipo di servizio che si deve creare.
4. Recuperare l'ID della VM Image installata (emi-12345678) e creare un'istanza da essa (euca-run-instances –k … -t … emi-12345678).
5. Accede con ssh all'istanza di VM creata ed eseguire tutti i passi necessari per installare il servizio da utilizzare in Sunflower (ssh –X –i … root@ip_instance).
6. Salvare l'immagine del disco virtuale modificato e aggiungerlo come indicato in Managing Eucalyptus Images alle VM Images, recuperando l'ID della VM Image creata.

Per registrare un service on-demand su Sunflower, bisogna accedere alla scheda Register Service della SC e cliccare su "Register Cloud On-demand Service" (vedi figura 10). La Form in essa contenuta ha vari campi, che vanno a formare il contenuto di una tupla che lega assieme un servizio on-demand, un servizio astratto e un certificato per Euca2ool.

- URL del WSDL relativo al servizio on-demand da registrare
    1. Protocollo usato dal servizio (http / https)
    2. ID della WM Image contenete il servizio (emi-XXXXXXXX)
    3. Porta usata dal servizio (80 / 443 / 8080)
    4. Path del WSDL servizio creato

(es.: http://emi-12345678:8080/axis/Hello.jws?wsdl).

- URL del WSDL astratto nel WSDL Repository, da cui è stato generato il sevizio installato nel disco immagine della VM
  (es.: http://127.0.0.1:8080/SunflowerConsole/WSDLRepository/axis/Hello.wsdl).
- Locazione del file "euca2-admin-x509.zip", scaricabile dall'interfaccia web di Eucalyptus o tramite il comando "euca_conf --get-credentials euca2-admin-x509.zip", contenete le credenziali per accedere a Eucalyptus mediante Euca2ools.

N.B.: L' "URL del WSDL relativo al servizio on-demand da registrare" è identico all' "URL del WSDL relativo al servizio reale da registrare" in cui l'IP è sostituito dall'ID della VM Image che contiene il servizio.

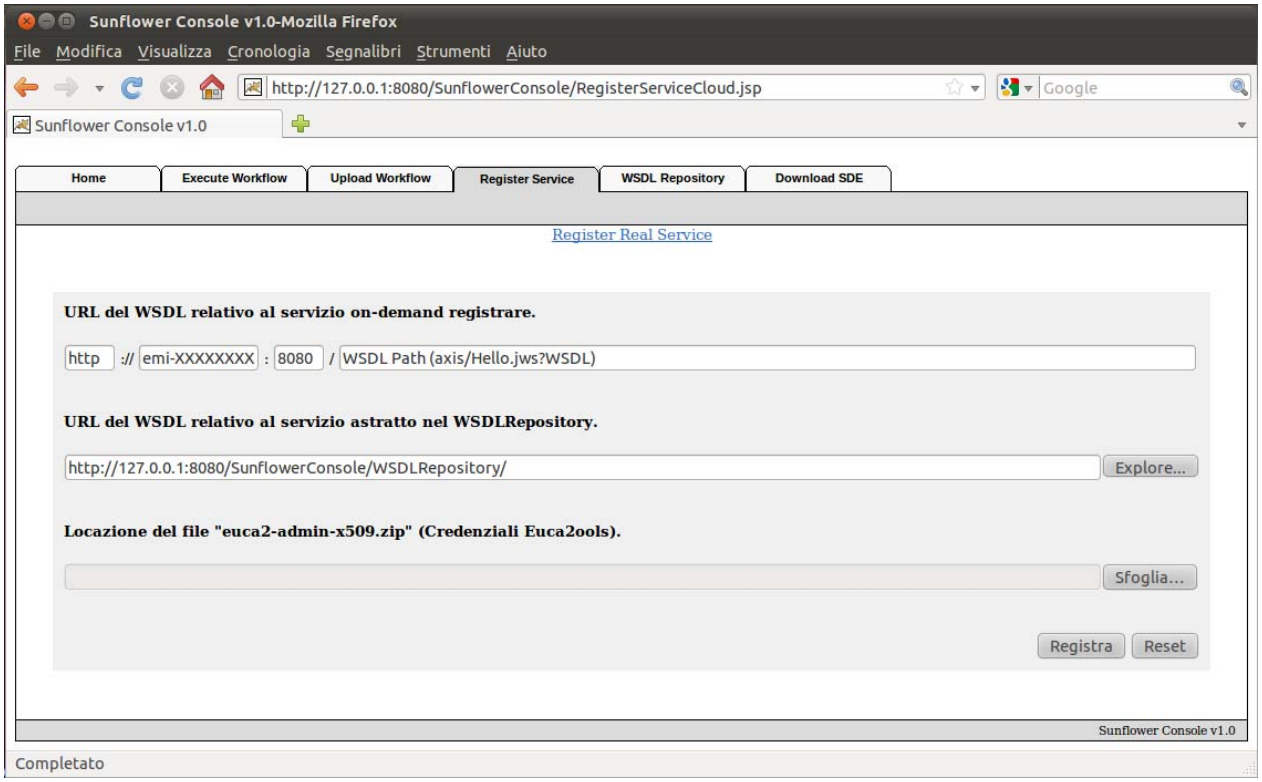

Figura 10: Registrazione di un servizio on-demand con il relativo servizio astratto e certificato.

Come si può notare, la registrazione di un servizio on-demand Eucalyptus è molto simile alla registrazione di un servizio reale, mentre la logica di utilizzo è fortemente differente. Analizzando la Form si vede che l'URL del WSDL del servizio non è più un indirizzo reale, ma un URL che in fase di utilizzo andrà opportunamente risolto per farlo puntare al servizio on-demand. Vediamo i passi di come questa risoluzione avviene:

1. Quando non sono più disponibili servizi reali associati al servizio astratto richiesto, si verifica se esiste un servizio on-demand del tipo richiesto.

2. Se esiste, si recupera L'ID della VM Image e la SC tramite le librerie Euca2ool provvedere a creare un'istanza di VM.
3. Si recupera l'IP dell'istanza di VM attiva, e si sostituisce nell'URL del WSDL del servizio on-demand al posto dell'ID della VM image.
4. L'URL del servizio on-demand creato è restituito e utilizzato come un normale servizio reale.

Un'altra importante differenza sta nel fatto che il servizio on-demand non è gestito direttamente da un suo SDE, ma sfrutta altri SDE per l'esecuzione e Eucalyptus per la gestione della QoS. Infatti, se per i servizi reali bisogna registrare il SDE di competenza, per i servizi on-demand bisogna registrare le credenziali Euca2ools per l'accesso ad Eucalyptus. Grazie alle credenziali di accesso, Euca2ools è in grado di istanziare una VM con la QoS richiesta dall'utente per il servizio, ed Eucalyptus provvederà a mantenere tale QoS per tutto il ciclo di vita dell'istanza di VM. Venuta a mancare la necessità di monitorare direttamente la QoS del servizio da parte del SDE, i frammenti di codice BPEL relativi al servizio on-demand possono essere tranquillamente accorpati con quelli associati ad un altro servizio reale, per poi invocare il servizio on-demand non più come servizio locale ma come servizio remoto. Tale strategia per i servizi on-demand presenta due vantaggi, il primo è quello di mantenere l'istanza di VM più leggera possibile, la seconda è quella di evitare problemi di comunicazione tra gli SDE dei servizi reali e un eventuale SDE installato in una istanza di VM.

Il tempo di vita di un'istanza di VM è determinato dalla durata del workflow, infatti quando questo termina lancia un messaggio di garbage che distrugge l'istanza del workflow eseguito, e nel caso di servizi on-demand creati durante la sua esecuzione provvede a terminarne l'istanza di VM (euca-terminate-instance i-xxxxxxxx).