On-demand VM provisioning and dynamic application deployment using SmartFrog

Antonis Papaioannou, Damianos Metallidis and Kostas Magoutis
Institute of Computer Science (ICS)
Foundation for Research and Technology – Hellas (FORTH)
Heraklion 70013, Greece
Email: {papaioan,metal,magoutis}@ics.forth.gr

I. Purpose of document

The purpose of this document is to present in more detail how we extend the applicability of SmartFrog to on-demand provisioning of cloud VMs and dynamic deployment of software components on them as described in [refenrece paper]. We model VMs as SmartFrog component objects, similar to any other application component. Each of these VM objects has its own lifecycle, state, configuration parameters, and management events.

II. SmartFrog code and runtime support

We have SmartFrog-enabled the software stack of a representative enterprise application, the SPEC jEnterprise2010 [1] benchmark. The software stack includes the JBoss application server, MySQL database server, application logic packaged in an enterprise archive file (specj.ear), and a load balancer interposed between load-generating clients and application servers (the inclusion of a load balancer is an enhancement over the standard version of SPEC jEnterprise, which supports a single application server instance). The SmartFrog version of SPEC jEnterprise2010 operates as follows: Each component (JBoss, MySQL, load balancer) includes a sfStart() method that triggers the installation and deployment of the respective component. Components terminate via the sfStop() function.

The SmartFrog management code for SPEC jEnterprise2010 comprises the following four main components

- Virtual machine, dynamically provisioned in a public, private or hybrid cloud
- Database server
- Application server
- Load balancer, responsible for spreading client load over multiple application servers

Listing 1 shows SmartFrog declarations of abstract components that will be extended and referenced later in *.sf* files. A key abstract component is *Public_Cloud_VM*, which represents the provision of VMs of a specific type. The attributes of *Public_Cloud_VM* include the cloud provider the VM is going be deployed on, the type of the VM (e.g small, medium, etc), an indication of which software component will be deployed on it (taking values in LB (load balancer), AS (application server), DB (database), etc). An important attribute in *Public_Cloud_VM* is *VM_IP* whose value is late bound to the IP address of the VM when the latter is provisioned and started. Late bound variables in Listing 1 are declared using SmartFrog's TBD keyword. The binding is performed in the *sfDeployWith()* function of any concrete VM class that extends *Public_Cloud_VM* (see Listing 2). *LB* and *ReconfigureLB* are the SmartFrog components responsible for deploying and reconfiguring the load balancer during a migration. The *LB* component consists of two attributes, the *IP* address of the load balancer, and the IP of the provisioned application server (*Worker_IP*). The SmartFrog declarations for the initial deployment of a full SPEC jEnterprise2010 software stack are shown in Listing 2.

Listing 1: Abstract SmartFrog code

```
Public_Cloud_VM extends Prim{
        sfClass              "...";
        CloudProvider       TBD;
      VM_IP               TBD;
      typeOf              TBD;
      SoftwareTypeVM    TBD;
}
LB extends Prim{
        sfClass              "...";
      LB_IP               TBD;
        Worker_IP           TBD;
}
ReconfigureLB TBD;
```

To respect provisioning and deployment dependencies we use SmartFrog's Compound statement to link the lifecycles of grouped components. Within a Compound (Listing 2) components are first deployed in the stated sequence prior to being started in the same sequence, as shown graphically in Figure 1.

Listing 2 shows a Compound deployment structure where the VMs that the application server and database server are going to be deployed on are provisioned first. The code that performs the actual provisioning is pointed at by the *sfClass* attribute (details omitted for clarity). Then component *Application_Instance* deploys the application logic (expressed in *specj.ear*) and connects to the IP addresses of the provisioned database and application servers. Finally a load balancer instance is provisioned and deployed. The *LoadBalancer* component expresses the load balancer reconfiguration logic. Note that components expressing middleware to be deployed on previously provisioned VMs in Listing 2 (such as *Application_Instance*) are late-bound and cross reference the *VM_IP* attribute of their underlying VM. Similarly components at higher levels (such as *LB_Instance*) cross reference attributes of their underlying middleware components.

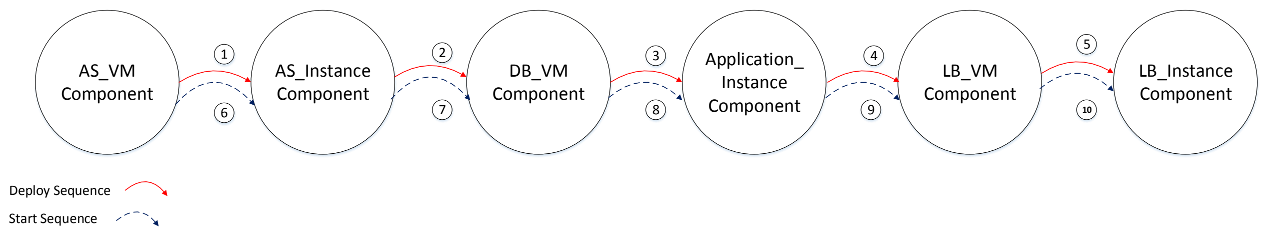

Fig. 1: Solid (red) arrows denote deployment sequence (”happens before”); Dashed (blue) arrows denote start action sequence.

Listing 2: Initial Deployment SmartFrog code

```
InitialDeployment extends Compound{
   AS_VM extends Public_Cloud_VM{
      CloudProvider   ”Amazon”;
      VM_IP           ”NotKnownYet”;
      typeOf          ”m1.small”;
      SoftwareTypeVM  ”AS”;
   }
   AS_Instance extends Prim{
      sfClass         ”...”;
      AS_Name         ”JBoss−1”;
      AS_IP    LAZY   AS_VM:VM_IP;
   }
   DB_VM extends Public_Cloud_VM{
      CloudProvider   ”Amazon”;
      VM_IP           ”NotKnownYet”;
      typeOf          ”m1.small”;
      SoftwareTypeVM  ”DB”;
   }
   Application_Instance extends Prim{
      sfClass         ”...”;
      DB_Name         ”MySQL−1”;
      AS_IP    LAZY   AS_vm:VM_IP;
      DB_IP    LAZY   DB_VM:VM_IP;
   }
   LB_VM extends extends Public_Cloud_VM{
      CloudProvider   ”Amazon”;
      VM_IP           ”NotKnownYet”;
      typeOf          ”m1.small”;
      SoftwareTypeVM  ”LB”;
   }
   LB_Instance extends LB {
      LB_Name         ”mod_jk−1”;
        LB_IP            LAZY LB_VM:VM_IP;
        Worker_IP        LAZY AS_VM:VM_IP;
   }
}
```

Listing 3 shows the onEvent component (part of SmartFrog’s workflow [2] architecture) that handles the adaptation event. The *Load_Balancer* component handles the arrival of one or more events describing conditions that require adaptation. The *singleEvent* attribute declares whether the onEvent component should respond once or multiple times to the arrival of a potential stream of events. *sfProcessComponentName* declares the name of the component for resolution purposes.

Listing 3: onEvent SmartFrog code

```
LoadBalancer extends OnEvent{
singleEvent false;
sfProcessComponentName ”LoadBalancer”;
  ReconfigureLB extends LAZY LB{
     sfClass     ”...”;
     WorkerIP   LAZY HOST Migration_Provisioner_IP:
          AS_IP:VM_IP;
     LB_IP      LAZY HOST Initial_Provisioner_IP:
          VM_LB:VM_IP;
     }
}
```

Listing 4 shows a sendEvent component (also part of SmartFrog’s workflow architecture) that triggers the event called *ReconfigureLB* on a specific host and software component via specific references, in this case the *LoadBalancer* component (Listing 3).

Listing 4: sendEvent SmartFrog code

```
LoadBalancerEvent extends EventSend{
      sendTo:a  LAZY HOST Initial_Provisioner_IP:
           LoadBalancer;
      event      ”ReconfigureLB”;
    }
```

References

[1] “SPEC jEnterprise2010 Benchmark,” Accessed 5/2014. [Online]. Available: http://www.spec.org/jEnterprise2010/

[2] “SmartFrog Workflow,” Accessed 3/2014. [Online]. Available: http://www.hpl.hp.com/research/smartfrog/releasedocs/smartfrogdoc/sfWorkflow.html