FSToolkit: Adopting Software Engineering Practices for Enabling Definitions of Federated Resource Infrastructures

Christos Tranoris and Spyros Denazis

University of Patras, Greece
tranoris@ece.upatras.gr, sdena@upatras.gr

Abstract. Today organizations own resources and infrastructures (i.e. networking devices, gateways, wireless devices) that would like to either offer through the cloud model or to combine with resources of other infrastructures. Federation can be enabled by means of a resource broker that matches customer's requested services and providers' resources according to the agreed SLA. Users need ways to define complex deployments and request for resources without knowing the underlying infrastructure details. In this paper we present the Federation Scenario Toolkit (FSToolkit) that enables the definition of resource request scenarios, agnostic in term of providers. This work adopts Software Engineering practices considering the concepts of modeling and meta-modeling to define a resource broker and to specify scenarios by applying the Domain Specific Modeling (DSM) paradigm. FSToolkit is developed for experimentally driven research for validating through testing-scenarios new architectures and systems at scale and under realistic environments by enabling federation of resources.

Keywords: Federation, experimentally driven research, Resource Broker, Domain Specific Modeling.

1 Introduction

Future Internet research needs new infrastructures for supporting approaches that exploit, extend or redesign current Internet architecture and protocols. During the last few years experimentally driven research is proposed as an emerging paradigm for the Future Internet on validating through testing-scenarios new architectures and systems at scale and under realistic environments. Until recently, testbeds used in testing activities have usually a certain scope of testing capabilities. Organizations own resources and infrastructure (i.e. networking devices, gateways, wireless devices) that would like to either offer through the cloud model or to combine with resources of other infrastructures in order to enable richer and broader experimentation scenarios. Experimentally driven research addresses the need to evolve the test beds into coherent experimentation facilities. This is possible by enabling large-scale federated infrastructures of exposed organizational resources and testbed facilities. Such future experimental facilities are leaded by global efforts like GENI [1] and FIRE [2].

F. Álvarez et al. (Eds.): FIA 2012, LNCS 7281, pp. 201–212, 2012.

Federated infrastructures in experimentally driven research need models, architectures and tools to address the definition and execution/operation/control of the experiment. In our previous work [3], we presented a paradigm called Federation Computing where it deals with the aspects of defining and operating/controlling experiment scenarios or so called *Federation Scenarios.* We applied these concepts in the context of the Panlab project [4]. A *Federation Scenario* is a well-defined specification of (heterogeneous) services or resources and their configurations, offered by a diverse pool of organizations in order to form richer infrastructures for experimentally driven research. A Federation Scenario is the equivalent of an SLA required by the end-user, which is the customer of the federation. These federation scenarios represent customer needs such as i) evaluation and testing specifications of new technologies, products, services, ii) execution of network and application layer experiments, or even iii) complete commercial applications that are executed by the federation's infrastructure in a cost-effective way.

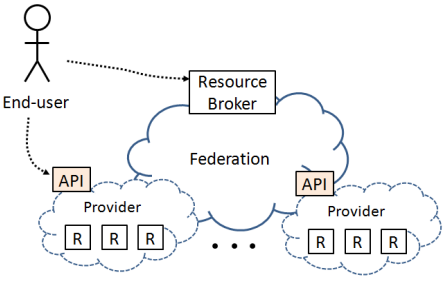

Fig. 1. Federations of Resource Providers and Brokers

A Federation Scenario describes end-user needs for services provided by resources of a federated infrastructure. At this point resource brokers play a key-role in creating and supporting federated infrastructures. A resource broker matches customer requested services and provider resources from the federation. Federation and resource brokers are well addressed by the cloud computing community in [5]: "*Federation is the act of combining data or identities across multiple systems. Federation can be done by a cloud provider or by a cloud broker. A broker has no cloud resources of its own, but matches consumers and providers based on the SLA required by the consumer. The consumer has no knowledge that the broker does not control the resources.*"

Figure 1 displays resource providers forming a federation where a resource broker is capable of exposing resources R to end-users in a uniform manner to create richer infrastructures. Resource providers must have an API that exposes resources and enables brokers to browse and manage them. The end-user can create scenarios involving resources by directly going to a resource provider or by going to a resource broker of a federation.

This work discusses how we adopted Software Engineering practices of Domain Specific Modeling (DSM), where the systematic use of textual or graphical Domain Specific Language (DSL) is involved. A DSL is defined as a specification language

that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain. For the language definition an abstract syntax (the meta-model), a concrete syntax and semantics are needed. All of these are captured in a solution workbench, which in this case is Eclipse, used both as a development but also as a deployment environment.

Having stated the above, we present a meta-model for defining a resource broker and how Domain Specific Modeling (DSM) is used to define Federation Scenarios. We implemented a meta-model that describes resource brokers offering (representing) services later mapped to resource providers. The Domain Specific Languages (DSLs) used by resource brokers, resource providers and experimenters, have the proposed meta-model as an abstract syntax. The meta-model is called Office meta-model, since it has inherited its name from the Panlab Office which is used to federate resources. However, the Office meta-model is generic enough to describe any resource broker.

A DSL called Office Description Language (OfficeDL) is used by resource brokers or resource providers to describe them. The end-user (an experimenter or customer) uses the Federation Scenario Description Language (FSDL). FSDL is a DSL to describe the needed services of an experiment over a federated infrastructure. We also discuss how we used Model-to-Model transformation between resource brokers in order to import in the language heterogeneous resources by other resource brokers or resource providers expressed with other models. Model-to-Text transformations are used to generate wrapper code for exposing resources and for targeting different provisioning engines.

The paper is structured as follows: First we present the proposed meta-model and its core entities. Then we present the OfficeDL used by resource brokers and resource providers and then we provide details of the FSDL and its concrete syntax in describing Federation Scenarios. All the languages are supported by the FSToolkit tooling which is also presented.

2 The Meta-Model Describing Resource Brokers and Federation Scenarios

A Federation Scenario describes customer needs for services over a federated infrastructure. To support this, we needed first to define a resource broker. Thus, we define a meta-model the *Office meta-model* (figure 2, level M2) which describes resource broker models (in M1) and eventually instantiations of them in Federation Scenario definitions (M0). In the *Office meta-model* the core entity *Office* is defined. An *Office* is a resource broker offering services and matching services and resources, maintains users, and in general support federation scenarios.

In our work we define *Offered Services* and *Offered Resources* in *Office* as follows:

An *Offered Service* is an abstract entity and it describes an offering along with its configuration attributes, e.g. Computing Resource with memory, disk space, etc.

An *Offered Resource* is an entity that implements an *Offered Service***.** e.g. Resource Acme.Comp1234 is a resource of the provider Acme capable of implementing a service of Computing Resource (creating Virtual Machines).

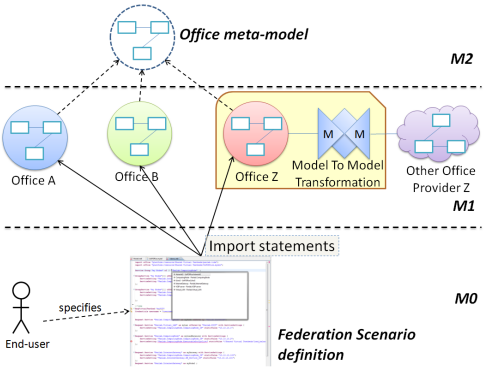

Fig. 2. The Office meta-model levels

An *Offered Resource* is supposed to be managed by Create-Read-Update-Delete operations. So an *Offered Resource* is currently a really simple entity with a few attributes exposed to the end-user. The same applies to an *Offered Service.*

The Office meta-model, is defined in Ecore: a variant of OMG's MOF [6] that has been developed in the Eclipse Modeling Framework [7] and is more or less aligned on OMG's Essential MOF (EMOF).

The Office meta-model defines related entities and their relationships, what an *Offered Service* is, what an *Offered Resource* is, how an *Offered Service* is supported by a resource of the federation, taxonomies, service compositions, SLAs, users, etc. Part of the meta-model is illustrated in figure 3, where it displays that an *Office* is an aggregation of *Offered Services*, *Users* and *Requested Federation Scenarios*. The *Office* aggregates *Requested Federation Scenarios* where an *SLA* (not shown) is created for each one of them. Since the entity *Office* describes actually a resource broker, it has an aggregation of providers offering resources. A *Resource Provider* is viewed as a user of the *Office*. A *Resource Provider* has an aggregation of *Sites* and eventually a *Site* contains the *Offered Resources*.

An *Office* matches *Offered Services* and *Offered Resources*. Having this, the *Office* maintains some contracts the *ResourceServiceContracts* (see Figure 3 right side). A contract helps the broker to match a service to a resource. From figure 3, one can see that a *ResourceServiceContract* is between an *Offered Service* and an *Offered Resource*. Some extra characteristics of the contract are described in the *Availability* of the Resource and potential *Cost*.

Figure 4 displays part of the Office meta-model which is used as the abstract syntax of the FSDL language. Classes here are instantiated later on while the end-user specifies the Federation Scenario. The *RequestedFederationScenario* contains user *Credentials*, *ScheduledPlans*, *Import* for URIs and most importantly a *ServicesRequest*. The *ServicesRequest* is a composition of *ServiceRequest*, the services that the end-user wants for his scenario. Each *ServicesRequest* references an *(Offered) Service* and contains some requested *ServiseSettingInstances.*

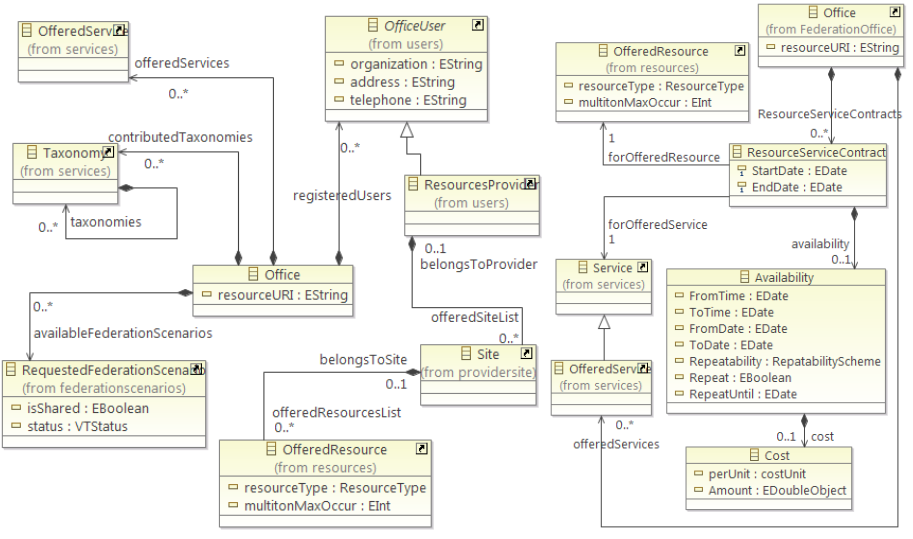

Fig. 3. Part of the Office meta-model and ResourceServiceContract

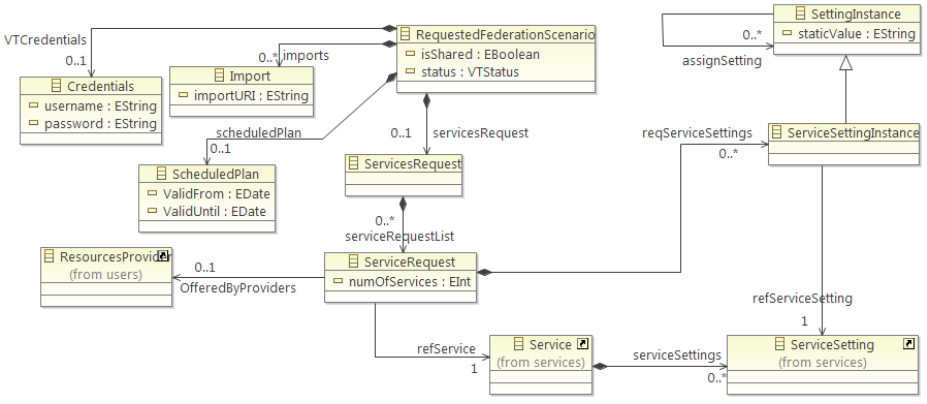

Fig. 4. A view of the RequestedFederationScenario

3 The OfficeDL: Describing Resource Brokers and Resource Providers

To enable rapid prototyping of the *Office* meta-model and to enable easy instantiations of the meta-model we developed a DSL called OfficeDL. OfficeDL has as abstract syntax the meta-model. The concrete syntax is based on the textual modeling framework (TMF) of Eclipse [8] and specifically the Xtext framework [9. An example of describing an *Office* called myBroker is given below:

```
Office "myBroker" {

registeredUsers {
 OfficeCustomer "Tranoris" {
  address "Address"
   hasAccount Account "Name" {
    password "Password" username "Username"
   }
  },
 ResourcesProvider "ProviderA"{
   offeredSiteList {
    Site "WCL" {
     ptm PTM "uopPTM" { IP "150.140.184.234" }
     igwlist { IGW "uopIIW" { IP "150.140.184.231" } }

     locatedAt SiteLocation "loc" {
      address "Rion, Patras"
      geocoords "5435345.00, 325345.00"
     }
     offeredResourcesList {
      OfferedResource "UpatMI" {},
      OfferedResource "VM_STANDARD" {    },
      OfferedResource "VM_MEDIUM" {},
      OfferedResource "VM_LARGE" {}
     }
    }
   }
 },
```

The language tokens are with bold fonts and variables with other fonts. Having this, while someone uses the language, he creates a model of his own office, defining: users, offered services, resource providers, offered resources, etc… Some benefits of creating such a DSL: there is a way to quickly check the meta-model for its correctness; tools can import the instantiated models which are validated from the framework; resource brokers can use it to describe their users, offered services, providers and contracts; finally, resource providers may use it for describing only their own organization resources for local usage and offer all the available tooling to their users.

It is expected that OfficeDL will be used for small to medium broker and provider descriptions. For large organizations a permanent repository supporting the model is more adequate. These descriptions though will be useful later on, when end-users use them for defining their federation scenarios.

4 FSDL: A DSL for the End-User

The previous section discussed the OfficeDL which is used by a resource broker and resource providers for describing federation entities. We have created another DSL

for enabling the end-user describing federation scenarios. The language is called Federation Scenario Description Language (FSDL). In the simplest usage an FSDL definition starts with the keyword *RequestedFederationScenario* followed by a name. A set of import office statements that contain definitions of the offices (the resource brokers, services and resources) may follow. Next, one can define either a resource agnostic scenario request or specific resources of providers. To illustrate the above we will discuss some examples.

The following, discusses a resource agnostic scenario request example (with a request for offered services). The request is towards a broker *brokerOfficeXYZ*. We would like to use an echo service that the *brokerOfficeXYZ* provides. The request is described in the following FSDL :

```
RequestedFederationScenario myScenarioName

import office "http://brokerOfficeXYZ.org/myresourcedef.office";

RequestServices{
 Service "brokerOfficeXYZ.echo" as myecho settings {//An echo
resource. Write something in input. Read the output to get it
  Setting "input" : myinput = "Hello"  //An input text for echo
  Setting "sleeptime_ms" : mysleeptime_ms = "3000"  //delay of echo
in msecs
  }
}
```

Inside the *RequestServices* section we describe the request for services and their initial settings. The keyword *Service* declares a new service request followed by the name of the requested service. In the presented example we request the echo service *echo*. After the *as* keyword we define an alias of the service (i.e. myecho). After the settings keyword follows the section with the initial settings of the requested service. In our example we define the two settings input (the input setting will be the output of the echo service) and sleeptime_ms (delay of the message).

In the next example we present the case of selecting resources from specific providers, where we use a slightly different syntax of the language. In this case, we would like to use an echo resource that provider *ProviderAcme* offers. We have two ways to express this request in FSDL. The first:

```
RequestedFederationScenario myScenarioName

import office "http://brokerOfficeXYZ.org/myresourcedef.office";

RequestServices{
  Service "brokerOfficeXYZ.echo" as myecho offered by
"brokerOfficeXYZ.ProviderAcme" settings{
    Setting "input" : input = "Hello"  //An input text for echo
    Setting "sleeptime_ms" : sleeptime_ms = "3000"  //delay of echo
in msecs
  }
}
```

The keyword *offered by* is used to define that the end-user wants to request the resource by the *ProviderAcme* provide. Another way for expressing this request is as follows:

```
RequestedFederationScenario myScenarioName

import office "http://brokerOfficeXYZ.org/myresourcedef.office";

RequestInfrastructure {
 Resource "brokerOfficeXYZ.ProviderAcme.site.echo_rp12_or10782" as
myecho settings {
   Setting "output" : output = ""  //
   Setting "input" : input = "Hello"  //
   Setting "sleeptime_ms" : sleeptime_ms = "2000"  //
 }
}
```

The *RequestInfrastructure* is used to describe a concrete infrastructure of resources and their attributes by specific resource providers. Both approaches could be used for different needs. Usually service definitions are more generic and contain generic settings that all resource providers supply. However it is possible that a resource can have more settings than the offered service it matches. The latter description of describing the infrastructure is submitted for provisioning. In general, the section *ServicesRequest* contains a list of *ServiceRequest* entities. The user creates instances of *ServiceRequests* in the language referenced by the imported model. The syntax for requesting an Offered Service is as follows:

```
Service "NAME_OF_SERVICE"  as nameAlias([1.. numOfServices
])?(offered by "ResourcesProvider"  (optional)?   )? settings {
  Setting "NAME_OF_SETTING":settingNameAlias (= staticValue)?
(assign +=SettingInstance|STRING]  ( , SettingInstance  )?
  Setting "NAME_OF_SETTING":settingNameAlias (= staticValue)?
(assign +=SettingInstance|STRING]  ( , SettingInstance  )?
     ...
     ...
}
```

Where:

- NAME_OF_SERVICE: a full qualified name of the service
- nameAlias: a user chosen value to name the service followed optionally by how many services he wants
- *offered by* is optionally to indicate to the broker that we need the specific provider.
- the optional keyword says to the broker to try to match the selected provider if possible
- NAME_OF_SETTING: the name of an attribute of an offered service

- settingNameAlias: a user chosen value to name the setting. If after the alias there is a = then the setting can have a static value. If there is the keyword *assign* the user can assign the value of another setting.

A more complex example to illustrate FSDL is the following: An end-user wants to deploy a XEN VM image to 15 machines. The resource broker *brokerOfficeXYZ* will allocate these later to his resource providers. The FSDL specification is as follows:

```
RequestedFederationScenario deployingAXenImage

import office "http://brokerOfficeXYZ.org/myresourcedef.office";

RequestServices{
 Service "brokerOfficeXYZ.xenimagestore" as myXENImageP2ner settings{
  Setting "Name":imgname = "myXENImageP2ner"
  Setting "InputURL":inputurl
="http://196.140.184.233/myxenimage.img"//The url to copy from
  Setting "OutputURL":outputurl //holds the location of the stored
image, to be used by testbed's resources
  }

 Service "brokerOfficeXYZ.xenvmdeploy" as clients[1..15] settings{
  Setting "CAP": cap = "50"
  Setting "MEM": mem = "512"
  Setting "URL": url assign "myXENImageP2ner.outputurl"
  Setting "NAME": name = "client"
 }
}
```

The user wants 2 services. The *xenimagestore* is used to move a XEN VM image to be used by a XEN host, where the *InputURL* setting defines the source of the image. The *xenvmdeploy* service is responsible for deploying the XEN image to a computing resource. Some parameters are depicted in the example. Also the keyword **assign** is used when we want to assign as input to this setting the value of another setting by another offered service. The clients are declared as a group of services (`clients[1..15]`). This gives the end-user flexibility when later runs the scenario to execute commands on all the services (and eventually the resources) of the group. Each *ServiceRequest* contains a list of settings that the user can define for the scenario. The end-user can either define values for each setting (eg. an integer or a string) or can assign output values from other resources of the scenario.

To help the end-user with the syntax and protect from syntax errors the FSToolkit [15] environment has a specific FSDL editor. The editor is based again on the textual modeling framework (TMF) of Eclipse and is installed by the end-user as Eclipse plugins. Figure 5 displays an overview of the FSToolkit with installed FSDL plugins. It contains views that help the end-user during the description of a scenario. On the left hand side there are views to see user projects, Offered Services from available Offices and stored scenarios on those offices. On the middle we depict the editor of FSDL files. The editor is capable of making syntax validation and the context-assist

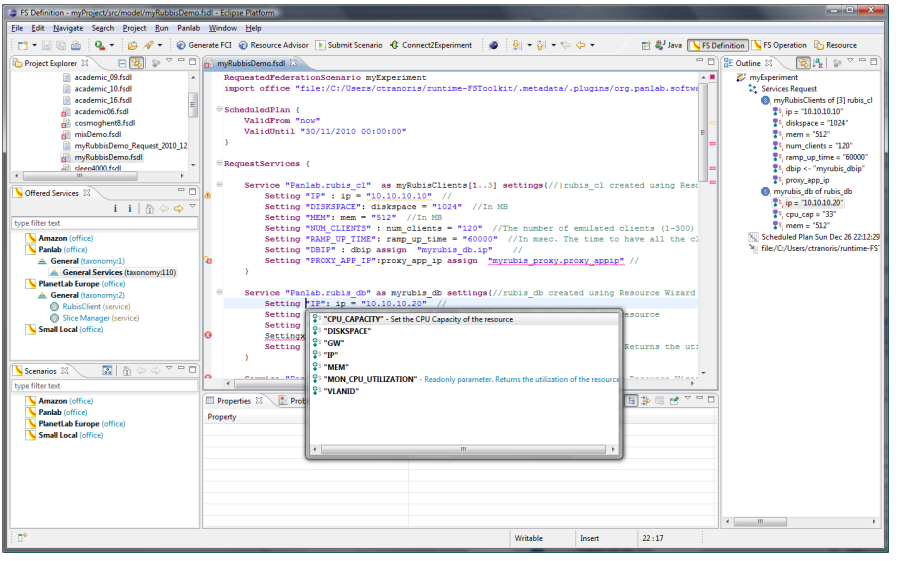

Fig. 5. An overview of the FSToolkit Federation Scenario editor

utility helps the end-user with the correct syntax by suggesting commands, keywords and variables. Moreover, double clicking on offered services triggers automatic text injection in the scenario description.

5 Towards Provisioning of Resources

Provisioning resources in a federated infrastructure is also a necessary step during a scenario's lifecycle. As discussed, our model assumes that resource brokers offer services that later on are matched to resources via contracts under certain availability, cost, policy, etc. Using FSDL, the end-user requests offered services from resource brokers.

All this "contract-oriented" information is used by a module called *Resource Advisor*, which transforms the Federation Scenario into a detailed list of requirements for specific resources. The Resource Advisor proposes to the Federation Scenario developer different *Implementation Plans* to continue, under certain cost and availability of the resources. In this way we have created a model of an SLA for federation scenarios in order to assign responsibilities to a certain resource for every item contained in an SLA. An SLA aggregates contracts for each requested service. To this end, a provider's resource is responsible for a specific requirement of the SLA. This approach of contracts and responsibilities of resources helps also towards monitoring an SLA for different aspects (ie metering, service quality, security, etc). The Resource Advisor module is a plugin in the FSToolkit environment.

The provisioning workflows invoke RESTful commands towards Broker Gateways (BGW) and eventually provision provider resources. A similar process is followed for tear down and releasing the resources.

6 Provisioning/Controlling Resources of Federation Scenarios: The Federation Computing Interface

What is critical with the operational part of a federation scenario is the proper and valid configuration of the participating resources. While the scenario is operated by a customer (i.e. during an application deployment or during an experiment on the federated infrastructure), the federation must ensure that all SLA terms are fulfilled and nothing is violated or falls out of the scope of the SLA. To this end, the SLA must be constantly monitored for different aspects (i.e. metering, service quality, security, etc). In [3] we presented some initial aspects of such an API, which is called Federation Computing Interface (FCI) [14].

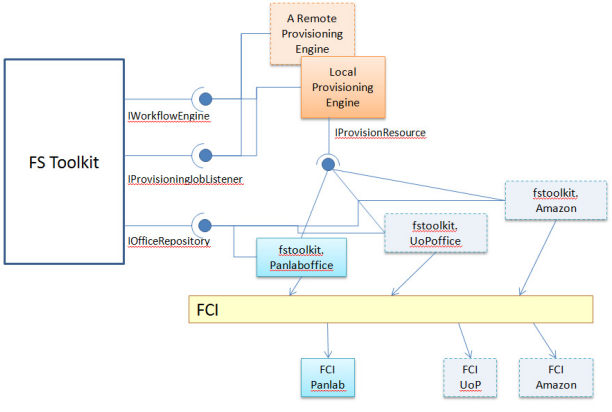

Fig. 6. A plugin based architecture of FSToolkit and Extension Points

7 Extending the FSToolkit via Extension Points

FSToolkit is based on the Eclipse platform and is being deployed to end-users as a set of plugins. Additionally, FSToolkit can be extended through defined Extension Points. Figure 6 shows this concept. There are three main Extension Points in FSToolkit. The IWoklflowEngine and the IProvisioningJobListener are used by plugins that are capable of handling provisioning of resources. The third extension point, the IOfficeRepository, can be used by resource providers and brokers to expose resources to the end-users, in order to create federation scenarios. A provider in order to support provisioning of resources, the extension point IProvisionResource of the Provisioning Engines must be implemented.

8 Conclusions and Future Work

This paper discusses how we applied Software Engineering practices and especially the Domain Specific Modeling paradigm, for defining federation scenarios. A meta-model for resource brokers and resource providers was presented. Moreover we developed a

family of DSLs targeting brokers, providers and end-users having the meta-model as abstract syntax. All appropriate tooling supporting is given through FSToolkit. All presented tools are licensed under the Apache License, Version 2.0. The meta-model can be downloaded from http://svn.panlab.net/PII/repos/Software/sources/FCI/org.panlab.software.office.model/model/. More details, instructions, source code and downloads are available also at our web site http://nam.ece.upatras.gr/fstoolkit.

Acknowledgments. The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) from project PII- Panlab and under grant agreement n° 287581 – OpenLab.

References

1. National Science Foundation, GENI, `http://www.geni.net` (last accessed February 12, 2012)
2. European Commission, FIRE website, `http://cordis.europa.eu/fp7/ict/fire` (last accessed February 12, 2012)
3. Tranoris, C., Denazis, S.: Federation Computing: A pragmatic approach for the Future Internet. In: 6th IEEE International Conference on Network and Service Management (CNSM 2010), Niagara Falls, Canada (2010)
4. Website of Panlab and PII European projects, supported by the European Commission in its both framework programmes FP6 (2001-2006) and FP7 (2007-2013), `http://www.panlab.net`
5. Opencloudmanifesto, Cloud Computing Use Cases White Paper, `http://www.opencloudmanifesto.org/` (last accessed February 12, 2012)
6. OMG website. Catalog of OMG Modeling and Metadata Specifications, `http://www.omg.org/technology/documents/modeling_spec_catalog.htm` (last accessed February 12, 2012)
7. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF eclipse modeling framework, 2nd edn. Addison Wesley (2008)
8. Eclipse Foundation website, `http://www.eclipse.org` (last accessed March 27, 2011)
9. Xtext framework website, `http://www.eclipse.org/Xtext/` (last accessed February 12, 2012)
10. RADL, Panlab wiki website, `http://trac.panlab.net/trac/wiki/RADL` (last accessed February 12, 2012)
11. Teagle, `http://www.fire-teagle.org` (last accessed February 12, 2012)
12. Specification Business Process Execution Language for Web Services, Version 1.1, `ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf` (last accessed February 12, 2012)
13. Belaunde, M., Falcarin, P.: Realizing an MDA and SOA Marriage for the Development of Mobile Services. In: ECMFA: European Conference on Modelling Foundations and Applications, pp. 393–405 (2008)
14. Federation Computing Interface (FCI), Panlab wiki website, `http://trac.panlab.net/trac/wiki/FCI` (last accessed February 12, 2012)
15. Federation Scenario Toolkit (FSToolkit) web site, `http://nam.ece.upatras.gr/fstoolkit` (last accessed February 12, 2012)