A distributed Information System for Service Oriented e-Infrastructures

Manuele Simi
Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo"
Consiglio Nazionale delle Ricerche
Area della Ricerca CNR di Pisa
Via G. Moruzzi, 1 - 56124 PISA - Italy
manuele.simi@isti.cnr.it

Abstract

The report illustrates the architecture of the Information System (IS) successfully adopted in the D4Science stream of EU-funded projects. The IS is part of the enabling layer of the gCube framework and offers a unique support for the dynamic deployment capabilities of the system. From the architectural point of view, it is composed by a set of cooperating Web Services developed in Java. These services can be distributed and replicated as needed to optimize the publication and discovery phase and to provide fault tolerance by avoiding single points of failures. In addition, a set client libraries have been developed to offer to clients the better experience possible when dealing with the IS. These libraries abstract over the details of the interaction with the services and their actual distribution and physical location.

Contents

1 Information System Architecture **3**
1.1 Standards . . . 4

2 IS-InformationCollector **4**
2.1 Role . . . 4
2.2 Documents . . . 5
2.3 XML Indexing . . . 6
2.4 Design and Interface . . . 7
2.4.1 XMLCollectionAccess port-type . . . 8
2.4.2 XQueryAccess port-type . . . 8
2.4.3 Sink port-type . . . 9
2.4.4 XMLStorageAccess port-type . . . 9

3 IS-Registry **9**
3.1 Role . . . 9
3.2 Design and Interface . . . 9
3.2.1 ResourceRegistration port-type . . . 10
3.2.2 Factory port-type . . . 10
3.2.3 Factory Resource . . . 11

4 IS-Publisher **12**
4.1 Role . . . 12
4.2 Design . . . 12
4.3 Registering Resources . . . 14
4.3.1 Instance State . . . 14
4.3.2 GCUBEResource . . . 15
4.3.3 XML Document . . . 15
4.4 Library Implementation Notes . . . 15
4.4.1 Configuration . . . 16
4.4.2 Bulked Publications . . . 16

5 IS-Client **16**
5.1 Role . . . 16
5.2 Design . . . 16
5.3 Queries . . . 17
5.3.1 Pre-defined Queries: Querying GCUBEResources . . . 17
5.3.2 Pre-defined Queries: Querying WS-ResourceProperties Documents . . . 18
5.3.3 Named queries . . . 18
5.3.4 Caller-defined queries . . . 18

6 In Conclusion **19**

1 Information System Architecture

The gCube Information System (shortly, IS) plays a central role in a gCube [3] Infrastructure: it delivers functionalities for publishing, discovering and monitoring the set of resources forming the infrastructure. It acts as the registry of the infrastructure, i.e. all the resources are registered in the IS and every service partaking in the infrastructure must refer to it to dynamically discover the other infrastructure constituents.

The approach provided by the IS is of great support for the dynamic deployment capabilities [12], [11], [14] and the interoperability solutions [13] offered by the gCube Framework.

In this context, a resource can be:

- a gCube resource, supporting the deployment and operation of a gCube infrastructure (typical gCube resources are nodes, instance of services, deployable packages, collections of data, etc.);
- an instance state, characterizing the operational state of an instance of a gCube service (more specifically, a stateful WS-Resource [7])
- a generic resource, any piece of well-formed XML information

To deliver the quality of service and performances and to handle growing amounts of information (scalability), the Information System is composed by a set of Web Services and client libraries. Figure 1 presents these components of the Information System and their main interactions:

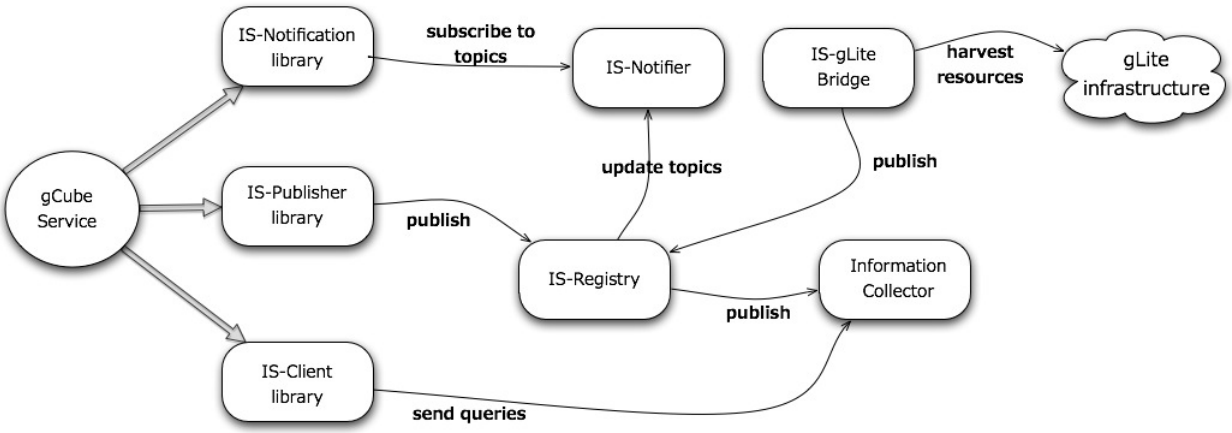

Figure 1: Information System Architecture and Main Interactions

They globally deliver the following functionalities with respect to the information handled:

- production and publication
- collection and storage
- discovery and consumption

In the following, this report focuses on the components dedicated to the publication, storage and discovery phases, leaving to other reports the description of the Notification paradigm.

1.1 Standards

The IS has been conceived to be as much open as possible in order to be exploited also by non-gCube Services. Thus, it adopts standards in its key features.

Earlier versions were deeply bound to the WS-Resource Framework [7], WS-ServiceGroup [9] and WS-ResourceProperty [8] specifications were used for the production and publication phase. Starting from version 2.0 (released in Feb 2011), the IS has been re-designed around the WS-DAIX [5] specification for publishing data.

As for the discovery phase, since the first version released, the IS accepts and processes queries compliant with the XQuery [10] language.

The IS-Notifier service in conjunction with the IS-Notification library allows event driven programming between gCube services by building on top of WS-Notifications [6].

Finally, worthy to mention, for the client libraries delivered with the IS, the principle *program to an interface, not an implementation* drove their design and their implementation. In this part, we tried to maintain the IS consumers and producers as much as possible decoupled from the IS internal logic, details of the interaction and actual distribution and replication of the IS services. More concretely, a gCube service has to know only the IS-Client, IS-Notifier and IS-Publisher interfaces and that's all. It does not need to care about their implementation (mechanisms to dynamically load the IS-Client, IS-Notifier and IS-Publisher at runtime have been put in place) nor the actual IS deployment scenario (completely abstracted by the IS client libraries).

2 IS-InformationCollector

2.1 Role

The IS-InformationCollector is a gCube service in charge of aggregating the information published by the gCube services belonging an infrastructure or a subset of it (this depends on the infrastructure configuration).

Major features of the service are:

- storage, indexing, and management of gCube Resources profiles
- storage, indexing, and management of instances' states in the form of WS-ResourceProperties documents
- storage, indexing, and management of well-formed XML documents
- full XQuery 1.0 support over the collected information
- remote management of the underlying XMLStorage

It plays a crucial role on an infrastructure, since it provides to clients a continuously updated picture of the infrastructure and its state. Responsiveness is also a key point of the service: many queries are sent to the InformationCollector during online operations and any delay introduces by the query processing is immediately reflected on any aspect of the system.

Historically, the service was conceived as an aggregator service, able to create Aggregator Sinks that query remote Aggregator Sources (in particular QueryAggregatorSources, as those created by the IS-Publisher) to harvest resource properties.

Version 3.0 features a new WS-DAIX compliant interface proving a more general approach to the feeding phase.

2.2 Documents

As wrote, IS-InformationCollector is capable to handle three class of documents. Two of them, gCube Resource profiles and instance's states, have a specific semantic in the infrastructure since:

- gCube Resource profiles are the manifestation of gCube Resource and allow interested client to discover such resources
- instance's states, in the form of WS-ResourceProperties document, are views on the state of instance of gCube services

Because of this semantic, the two classes of resource require some extra-information for their correct management. In particular, the to-be-stored documents has to come along with a *metadata record* reporting information about the publisher and the resource lifetime. The record must have the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<Metadata>
  <Type>Profile|InstanceState</Type>
  <Source>URI</Source>
  <TimeToLive>600</TimeToLive>
  <GroupKey>MyGroupKey</GroupKey>
  <EntryKey>MyEntryKey</EntryKey>
  <Key>MyKey</Key>
  <Namespace>URI </Namespace>
  <PublicationMode>push|pull<7PublicationMode>
</Metadata>
```

It gives to the service the information about:

- the type of the resource (*Profile* or *Instance State*)
- the URL of the publisher service (*Source*)
- the lifetime (*TimeToLive*) of the resource in seconds, which is added to the time the service receives the document (this is valid only for instance states published in push mode)
- the service group resource, identified by the *GroupKey* and *EntryKey* element (this is valid only for instance states published through the Sink port-type)
- the *namespace* of the WSDL in which the resource was defined (this is valid only for instance states)
- the *publication mode*, *push* if the resource is pushed at each change, *pull* if it is periodically sent to the service (this is valid only for instance states)

This information are then wrapped around the source document and indexed.
The resulting document looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<Document>
  <ID>3bb6e850-94d2-11df-8d06-8e825c7c7b8d</ID>
  <Source>http://source</Source>
  <SourceKey>MyKey</SourceKey>
  <CompleteSourceKey/>
```

```
  <EntryKey>MyEntryKey</EntryKey>
  <GroupKey>MyGroupKey</GroupKey>
  <TerminationTime>1288103198680</TerminationTime>
  <TerminationTimeHuman>Tue Oct 26 15:26:38 GMT+01:00 2010</TerminationTimeHuman>
  <LastUpdateMs>1288102598680</LastUpdateMs>
  <LastUpdateHuman>Tue Oct 26 15:16:38 GMT+01:00 2010</LastUpdateHuman>
  <Data>
    <Resource xmlns="" version="0.4.x">
      <ID>...</ID>
      <Type>...</Type>
      <Scopes>
        <Scope>...</Scope>
      </Scopes>
      <Profile>
        <!-- the profile section is here -->
      </Profile>
    </Resource>
  </Data>
</Document>
```

This allows to:

1. have more enriched queries based also on the publisher's data
2. manage the termination time of the resource by relying on the time machine of the node hosting the Collector instance (the TimeToLive in seconds is added to the last update time in order to obtain an absolute expiration time of the document), by avoiding problems due to different timezones in the infrastructure.

About the target collection in which the document will be stored:

- the collection is automatically derived (and created, if needed) starting from the document type and the information included in the document itself
- the collection name reported passed to the AddDocuments operation is therefore ignored.

If a document comes alone, i.e. without a metadata record, it is treated as a generic XML documents and stored in the target collection indicated in the addDocuments invocation.

2.3 XML Indexing

The IS-InformationCollector uses an embedded instance of eXist 1.2 [1] to index XML data according to the XML data model and offers efficient, index-based XQuery processing. The documents are stored in collections following this structure:

```
db
|
|- Profiles
|    |-<type1>
|    |-<type2>
|    |- ..
|    |-<typeN>
|
```

```
|- Properties
|
|- <User defined collection(s)>
|    |- <User defined sub-collection(s)>
```

Profiles and Properties collections are reserved to store respectively gCube resource's profiles and instance's states. Under the Profiles collection, a sub-collection with the resource type name is created whenever a new type is detected in a to-be-stored profile. Moreover, a client may define his own structure of collection and sub-collections and store, index and query there his documents via the XMLCollectionAccess port-type. This hierarchy of collections and sub-collections must be kept in mind when constructing a query expression to execute via the XMLQueryAccess.

Periodic exports (as zipped archives) of the XML database content are performed for backup purposes. The behavior of this activity can be configured in the service's JNDI file. The following section of the JNDI file shows the parameters that may be used to define where, when and how many backups are managed:

```
<service name="gcube/informationsystem/collector">

    <!-- ... -->

      <environment name="backupDir" value="existICBackups" type="java.lang.String"
      override="false" />

      <environment name="maxBackups" value="10" type="java.lang.String"
      override="false" />

      <environment name="scheduledBackupInHours" value="12" type="java.lang.String"
      override="false" />

</service>
```

In this example, backups are done every 12 hours and stored under the $HOME/.gcore/... /existICBackups folder. 10 backups are maintained and when a new one is available, the oldest one is discarded. If an absolute path is indicated as backupDir, the backups are not stored under the gCore persistent folder.

2.4 Design and Interface

The functionalities delivered by the service are logically organized across 5 port-types:

- XMLCollectionAccess, Sink and SinkEntry are dedicated to the publishing phase
- XMLQueryAccess allows to execute XQuery expression over the instance's content
- XMLStorageAcess is used to remotely manage the XML Storage underlying the service ins

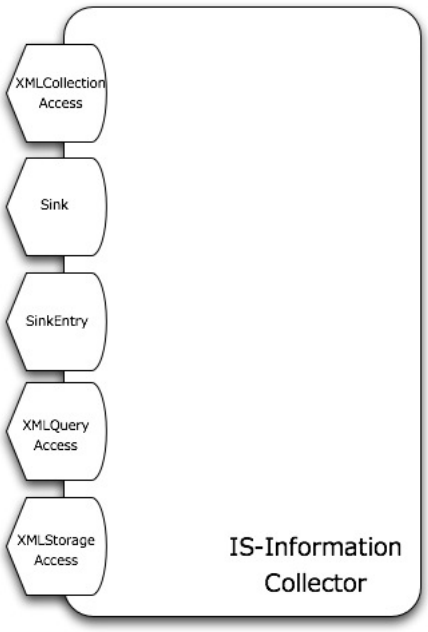

Figure 2: IS-Collector port-types

2.4.1 XMLCollectionAccess port-type

The XMLCollectionAccess port-type has been added from version 3.0 on. It defines the publication and gathering phase of the service following the the XML Realization (WS-DAIX) Specification, Version 1.0 of the Web Services Data Access and Integration [5]. It exposes the following operations:

- *addDocuments*: accept a list of new XML documents and eventually a list (of the same cardinality) of metadata records. An existing target collection must be also specified in case of generic XML documents.
- *removeDocuments*: remove a list of documents from a collection.
- *getDocuments*: retrieve a list of documents stored in a given collection starting from their identifiers.
- *createSubcollection*: create a sub-collection in a parent collection
- *removeSubcollection*: remove a sub-collection from a parent collection
- *addSchema*: bind a collection to an XML Schema
- *removeSchema*: remove the binding from a collection and an XML Schema

2.4.2 XQueryAccess port-type

The XMLQueryAccess supports the discovery phase. It exposes a single operation:

- *XQueryExecute*: process queries compliant with the XQuery 1.0 [10] specification that are executed against the actual content of the XML database.

2.4.3 Sink port-type

The Sink port-type is an aggregator sink port-type extending the ServiceGroupEntryAggregatorSink port-type defined in the Aggregator Framework [4]. Instance states acting as aggregator sources can invoke this port-type using the client classes provided by the Aggregator Framework:

- *org.globus.wsrf.impl.servicegroup.client.ServiceGroupRegistrationClient*: the register() method accepting the endpoint reference of the instance state (WS-Resource) and the endpoint reference of the Sink port-type allows to register the aggregator source to the sink.

- *commonj.timers.Timer*: an instance of Timer is returned by the register() method representing the registration task, if the cancel() method on the timer is invoked, the registration is deleted.

2.4.4 XMLStorageAccess port-type

This port-type allows remote management on the XML database instance. It exposes the following operations:

- *Backup*: trigger a full backup of the actual XML database's content. The exported data are stored in a zip archive under the backupDir.

- *Restore*: restore a previous backup of the XML database.

- *Shutdown*: safely shutdown the XML Storage.

- *Connect*: reconnect the service to the XML Storage.

3 IS-Registry

3.1 Role

The IS-Registry is the gateway to entering in a gCube infrastructure for gCube resources by means of registering/unregistering their profiles.

The IS-Registry performs three fundamental tasks:

- decide if accept or not a new resource (semantic validation)

- validate a resource before its registration (syntactic validation)

- execute post-deletion actions to keep consistent the IS content

3.2 Design and Interface

The design of the service is distributed across two port-types: the ResourceRegistration and the Factory. Both of them work for clients in a stateless manner, however the Factory creates a stateful WS-Resource [7] for notification purposes.

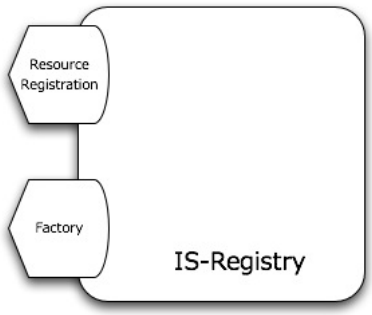

Figure 3: IS-Registry port-types

3.2.1 ResourceRegistration port-type

The ResourceRegistration port-type manages the registration/update/removal of GCUBE Resources. Its only expected client is the IS-Publisher. It also in turn uses the IS-Publisher to contact the XMLCollectionAccess' port-type of the InformationCollector instance in scope.

The port-type exposes three operations:

- *create*: take as input a CreateMessage containing the string serialization of the resource profile to register;
- *update*: take as input an UpdateMessage containing the new profile that will replace an existing one;
- *remove*: take as input a RemoveMessage containing the unique identifier of the resource to be removed and its type;

The first two operations throw an *InvalidResourceFault* if the profile was not correct/valid and a *ResourceNotAcceptedFault* if the profile was not accepted because of the instance's configured filters.

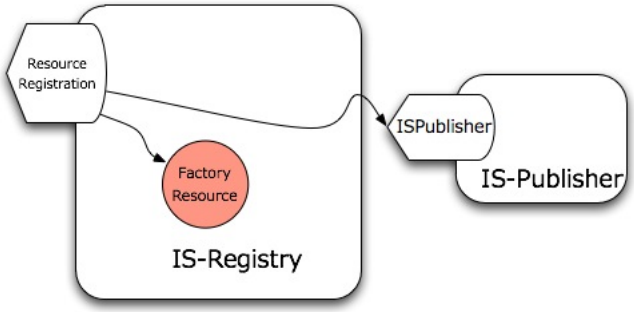

Figure 4: IS-Registry ResourceRegistration port-type

3.2.2 Factory port-type

From the functional point of view, the Factory port-type is practically a wrapper around the ResourceRegistration port-type to provide backwards compatibility to previous IS-Publisher and

testers implementation. Therefore, it exposes the following operations:

- *createResource*
- *updateResource*
- *removeResource*

that are mapped on the respective ResourceRegistration's operations. The usage of this port-type is strongly deprecated and it will likely disappear in the next releases of the service.

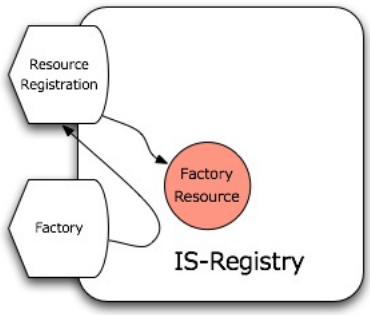

Figure 5: IS-Registry Factory port-type

3.2.3 Factory Resource

At start up time, the Factory port-type is in charge of creating the singleton FactoryResource. This resource (whose name is derived from previous versions of the service) exposes a set of WS-ResourceProperties [8] registered as Topics [6] in the IS-Notifier, making possible for interested clients to subscribe on events representing the changes of status of Infrastructure constituents (e.g. the disappearance of a Running Instance).

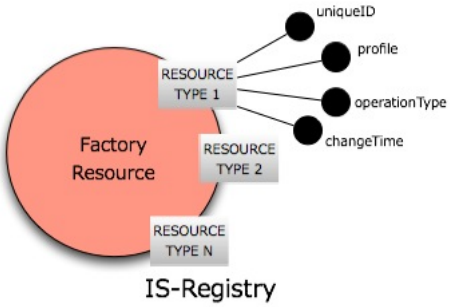

Figure 6: IS-Registry Factory Resource

This is the list of Resource Properties exposed:

```
<xsd:element name="RegistryFactoryResourceProperties">
        <xsd:complexType>
```

```
            <xsd:sequence>
                   <xsd:element ref="tns:RunningInstance" minOccurs="1" maxOccurs="1"/>
                   <xsd:element ref="tns:ExternalRunningInstance" minOccurs="1"
                   maxOccurs="1"/>
                   <xsd:element ref="tns:Service" minOccurs="1" maxOccurs="1"/>
                   <xsd:element ref="tns:Collection" minOccurs="1" maxOccurs="1"/>
                   <xsd:element ref="tns:GHN" minOccurs="1" maxOccurs="1"/>
                   <xsd:element ref="tns:MetadataCollection" minOccurs="1" maxOccurs="1"/>
                   <xsd:element ref="tns:GenericResource" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
</xsd:element>
```

and each element of the sequence is of type ResourceProperty defined as follows:

```
<xsd:complexType name="RegistryProperty">
<xsd:sequence>
<xsd:element name="uniqueID" type="xsd:string" nillable="true"/>
<xsd:element name="profile" type="xsd:string" nillable="true"/>
<xsd:element name="operationType" type="xsd:string" nillable="true"/>
   <xsd:element name="changeTime" type="xsd:dateTime" nillable="true"/>
</xsd:sequence>
</xsd:complexType>
```

Note that:

- *uniqueID* is the identifier of the resource
- *profile* is the string serialization of the resource's profile
- *operationType* is the type of operation performed on the resource (allowed values are: create, update, destroy)
- *changeTime* is the time the operation occurs

4 IS-Publisher

4.1 Role

The IS-Publisher models the behavior of providers of information to the Information System. It is the interface for publishing any piece of information in the IS. Clearly, the IS-Registry and the IS-InformationCollector can be directly invoked, but the IS-Publisher provides an high level interface for contacting instances of these services. Moreover, it completely hides the complexity of the actual deployment scenario of the IS.

4.2 Design

The IS-Publisher is a Java library providing a reference implementation for a group of interfaces defined in the gCore Framework [2].

More specifically:

- by implementing the *org.gcube.common.core.informationsystem.publisher.ISPublisher* interface, the library allows gCube services to publish GCUBEResources and instances' states as of WS-ResourceProperty documents;
- by implementing the *org.gcube.common.core.informationsystem.publisher.ISGenericPublisher and org.gcube.common.core.informationsystem.publisher.ISResource* interfaces, the library provides a way to publish generic XML documents in the IS;
- by implementing the *org.gcube.common.core.informationsystem.publisher.ISLocalPublisher* interface, it provides a subscription/notification mechanism based on local events.

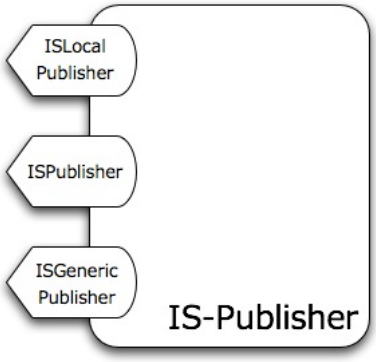

Figure 7: IS-Publisher Design

At runtime, all the above interfaces are dynamically bound by gCore to the implementation provided by the library.

Each registration request creates an internal resource sent to the appropriate IS service. Instance states and generic XML documents are wrapped as ISResources and sent to the Information Collector. GCUBEResources are instead sent to the IS-Registry service for validation and approval.

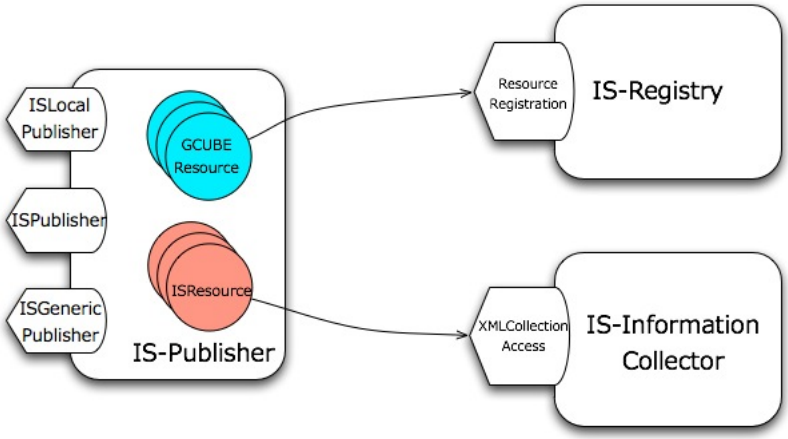

Figure 8: IS-Publisher Interactions

4.3 Registering Resources

4.3.1 Instance State

An instance state is the set stateful WS-Resource created by that instance following the WSRF [7] patterns.

In order to be published, a WS-Resource has to expose a view of its state as a mean of Resource-Property and declare a registration file for that properties in its JNDI (¡service folder¿/etc/deploy-jndi-config.xml). This declaration is obtained through the publicationProfile element inside the service section. This is an example of such declaration:

```
<service name="...">
<resource name="publicationProfile" type="org.gcube.common.core.state.GCUBEPublicationProfile">
            <resourceParams>
                <parameter>
                    <name>factory</name>
                    <value>org.globus.wsrf.jndi.BeanFactory</value>
                </parameter>
                <parameter>
                    <name>mode</name>
                    <value>push</value>
                </parameter>
                <parameter>
                    <name>fileName</name>
                    <value>Registration.xml</value>
                </parameter>
            </resourceParams>
        </resource>
</service>
```

The registration file specifies which properties have to be published, when and how following the syntax defined for the WS-MDS Aggregator Source registrations. The following example shows a registration file for 3 resource properties (RPString, RPDate, RPAny):

```
<ServiceGroupRegistrationParameters
    xmlns:sgc="http://mds.globus.org/servicegroup/client"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:agg="http://mds.globus.org/aggregator/types"
    xmlns="http://mds.globus.org/servicegroup/client">
    <Content xsi:type="agg:AggregatorContent"
        xmlns:agg="http://mds.globus.org/aggregator/types">
        <agg:AggregatorConfig>
            <agg:GetMultipleResourcePropertiesPollType
                xmlns:stateful="http://gcube-system.org/namespaces/test/stateful">
                <agg:PollIntervalMillis>60000</agg:PollIntervalMillis>
                <agg:ResourcePropertyNames>stateful:RPString</agg:ResourcePropertyNames>
                <agg:ResourcePropertyNames>stateful:RPDate</agg:ResourcePropertyNames>
                <agg:ResourcePropertyNames>stateful:RPAny</agg:ResourcePropertyNames>
            </agg:GetMultipleResourcePropertiesPollType>
        </agg:AggregatorConfig>
       <agg:AggregatorData/>
```

```
    </Content>
</ServiceGroupRegistrationParameters>
```

Note that the ResourcePropertyNames have to be fully qualified with the namespace declared in the service's WSDL (http//gcube-system.org/namespaces/test/stateful in the example above).

Name of the file aside, the other parameter to consider in the JNDI file is the mode. Clients may select the push or pull to publish their instance state. The chosen mode heavily impacts on the behavior of the ISPublisher.

Publishing with push mode With the push mode, the resource properties are (re)published whenever the values of one of them changes. If the RPs change every now and then, intermittently or there are peaks of changes in the RPs but longer periods without any change, this is the preferred way to go.

Publishing with pull mode When RPs change quite frequently and constantly and it's not critical to have them refreshed immediately, the pull mode has to be selected. In this modality, the ISPublisher periodically harvests and collects the RP values from the WS-Resource and publishes them in the Information Collector service. The polling period is indicated in the PollIntervalMillis parameter inside the registration file.

4.3.2 GCUBEResource

GCUBEResource profiles are managed by interacting with the ResourceRegistration portType of IS-Registry service. The ISPublisher here acts as a simple mediator by selecting the IS-Registry instance in the publishing scope and invoking its operations.

4.3.3 XML Document

Starting from the release 3.0 (Feb 2011), the ISPublisher offers the possibility to publish well-formed XML documents in the IS. This feature exploits the new XMLCollectionAccess portType of the Information Collector compliant with the WS-DAIX [5] specification. This raises significant opportunities for clients to create their own collections of indexed documents that can be queried through the IS-Client library.

4.4 Library Implementation Notes

Behind the exposed interfaces, there are three main paths of execution inside the library depending on the type of resources to manage.

If the published resource is a GCUBEResource, it is synchronously sent to the IS-Registry instance in the given scope. The selection of the appropriate instance is done by a set of handlers defined in org.gcube.common.informationsystem.publisher.impl.registrations.resources.

A completely different approach if followed when an Instance State is published. Each registered instance state requires a dedicated running task that reacts to the state changes. An instance of the org.gcube.common.informationsystem.publisher.impl.instancestates.InstanceStatePublisher class keeps track of the activated tasks. If the pull mode has been selected, an instance of RegisterInstanceStatePullHandler class periodically harvests the resource properties document and creates an ISResource to send to the InformationCollector for indexing. If the push mode has been chosen, an instance of RegisterInstanceStatePushHandler acts as an observer of the resource properties values and each time one of them changes receives a notification from the RPSet (gCore) and then sends the resource properties document to the InformationCollector.

The third path is related to the publication of generic XML Documents. The received documents (wrapped as ISResource) are simply sent to the InformationCollector for indexing.

4.4.1 Configuration

The behavior of the ISPublisher can be partially configured via a properties file. This file is located in GLOBUS_LOCATION/config/ISPublisher.properties. This is an example of such a file with a bit of explanation on the configurable properties:

```
# Timeout in the communications with the IS-Registry
REGISTRY_CHANNEL_TIMEOUT=60000

# Timeout in the communications with the IS-InformationCollector
COLLECTOR_CHANNEL_TIMEOUT=120000

# Max number of parallel registrations for this gHN
MAX_PARALLEL_REGISTRATIONS=100

# Max tries for publishing the resource (-1 means unlimited attempts)
RESOURCE_PUBLICATION_MAX_ATTEMPTS=3

# Interval between two bulk publications
BULK_PUBLICATIONS_INTERVAL=20000
```

4.4.2 Bulked Publications

To support an optimization of the load at infrastructure level, from release 3.0 on, instance states and XML documents to be sent to the InformationCollector are queued and periodically sent in a bulk way (i.e. in a single invocation). This is achieved with a new internal publisher named GCUBEGenericBulkPublisher. The interval between two bulk publications is specified in the BULK_PUBLICATIONS_INTERVAL of the configuration file.

5 IS-Client

5.1 Role

The ISClient is set of interfaces, templates and abstract classes defined in the context of the gCore Framework [2] to model the discovery of resources belonging an infrastructure on the IS. ISClient is also the name of the main interface for retrieving and sending queries.

A reference implementation of the ISClient is provided with the gHN distribution to interact with the IS-InformationCollector service.

5.2 Design

The entry point to the ISClient is the *ISClient* interface that defines the methods for obtain query objects and send them to the IS-InformationCollector instance in scope:

- *getQuery(Class¡QUERY¿ type)* which takes as input parameter the type of the query;
- *getQuery(String name)* which takes as input parameter the name of the query;

- *execute(ISQuery¡RESULT¿ query, GCUBEScope scope)* which takes as input parameter the query and the scope and returns a list of RESULT matching the query.

Queries cannot be instantiated with constructors as standard Java objects. They rather need to be obtained from the ISClient. A query object is an instance of a class implementing the *org.gcube.common.core.informationsystem.client.ISQuery interface*. Implementing classes declare also a type parameter RESULT indicating the type of the expected results. When the query is passed to the execute() method, instances of this type are returned.

Depending on the query, parameters and/or filters can be set to the query object to add further conditions.

5.3 Queries

There are three types of queries:

- *pre-defined* (or template) queries (GCUBEResourceQuery or WSResourceQuery)
- *named* queries
- *caller-defined* queries (GCUBEGenericQuery)

Each query is modeled by a class implementing the org.gcube.common.core.informationsystem.client.ISQuery interface. Query objects are not directly instantiated but must be obtained by the ISClient library.
To get any of the query objects described in the following, the getQuery() method has to be invoked

5.3.1 Pre-defined Queries: Querying GCUBEResources

The following queries classes are available to query over gCube Resources:

- GCUBECollectionQuery.class
- GCUBECSInstanceQuery.class
- GCUBECSQuery.class
- GCUBEExternalRIQuery.class
- GCUBEGenericResourceQuery.class
- GCUBEGHNQuery.class
- GCUBEMCollectionQuery.class
- GCUBERIQuery.class (with the implicit filters that the returned RIs are in the ready state)
- GCUBEServiceQuery.class
- GCUBETPQuery.class
- GCUBEVREQuery.class

All of them return a List of specialized org.gcube.common.core.resources.GCUBEResource objects (e.g. GCUBEServiceQuery returns a list of GCUBEService objects).

Once obtained the desired query object, the developer can add filters to better target the query on his needs.

Supported filters are:

- AtomicCondition with the atomic conditions can be specified that a node with a determined path *MUST* have a specified value

```
query.addAtomicConditions(new AtomicCondition("//Endpoint/@EntryName",
"gcube/annotationmanagement/abe/factory"));
```

- GenericCondition with the generic conditions can be specified an entire condition expression (using $result as starting node of every used path)

```
query.addGenericCondition("$result/[path] eq '[something]' or
\$result/[another path] eq '[something else]'");
```

5.3.2 Pre-defined Queries: Querying WS-ResourceProperties Documents

To query over GCUBEWSResources the following query class must be used:

```
ISClient client = GHNContext.getImplementation(ISClient.class);
 WSResourceQuery query = client.getQuery(WSResourceQuery.class);
```

As for GCUBEResourceQuery, also WSResourceQuery objects can be further refined with AtomicCondition and GenericCondition to better target the query.

Once executed, the query returns a List of RPDocument objects matching the query conditions. The RPDocument exposes the following getter methods for retrieving a set of common information about the resource.

Typically, a caller has also to access the values of the Resource Properties inside the document and this can be done by invoking the evaluate() method of the RPDocument object. This method accepts and executes an XPath expression on the WS-ResourceProperties document encapsulated inside the RPDocument.

5.3.3 Named queries

Named queries are widely used queries that are distributed with the ISClient to easy the caller's life. They are identified by a name and from the implementation point of view they are GCUBE-GenericQuery instances with an already injected query expression.

To get a named query:

```
 GCUBEGenericQuery query = client.getQuery("[one of the listed queries]");
```

Depending on the query, it is also possible to customize the query by setting specific parameters. To set a parameters:

```
query.addParameters(new QueryParameter("NAME","VALUE"));
```

Parameters' names are defined and querable inside the class query.

5.3.4 Caller-defined queries

The library also offers the possibility to execute custom queries by loading a GCUBEGenericQuery object and then set the whole query expression as follows

```
ISClient client =  GHNContext.getImplementation(ISClient.class);
GCUBEGenericQuery query = client.getQuery(GCUBEGenericQuery.class);
query.setExpression("myqueryexpression");
```

Custom queries can be over gCube Resource as well as WS-ResourceProperties documents. However, in order to successfully create them, the client has to be aware of the how the XML database organization on the IS-InformationCollector in order to target the proper collection or sub-collection of resources. Moreover, these queries can be used to retrieve generic XML Documents.

6 In Conclusion

The gCube Information System is a configurable and integrated set of components for collecting, storing, processing, and discovering information within an e-Infrastructure. Its main characteristics are the openness thanks to the adoption of widely-known standards and highly scalability by choosing the optimal deployment, i.e. by replicating and distributing its services according to the infrastructure needs.

Client libraries simplify to an extreme degree the interface with the services by creation a mediator layer that completely abstract over the service's interfaces and their deployment scenario.

References

[1] eXist-db Open Source Native XML Database. http://exist.sourceforge.net/.

[2] gcore framework. https://gcore.wiki.gcube-system.org/gCube/index.php.

[3] gCube Framework Official Website. http://www.gcube-system.org/.

[4] Globus toolkit 4.0: Aggregator framework. http://www.globus.org/toolkit/docs/4.0/info/aggregator/.

[5] Web Services Data Access and Integration – The XML Realization (WS-DAIX) Specification, Version 1.0. http://www.ogf.org/documents/GFD.75.pdf.

[6] Web Services Notification 1.2. http://www.ibm.com/developerworks/library/specification/ws-notification/.

[7] Web Services Resource Framework (WSRF). http://www.globus.org/wsrf/specs/ws-wsrf.pdf.

[8] WS-ResourceProperties 1.2. http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf.

[9] WS-ServiceGroup 1.2. http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-pr-01.pdf.

[10] XQuery 1.0. http://www.w3.org/TR/xpath-functions/.

[11] Leonardo Candela, Donatella Castelli, and Pasquale Pagano. gCube: a service-oriented application framework on the grid. *ERCIM News*, 72:48 – 48, 2008. In: ERCIM News, vol. 72 pp. 48 - 48. ERCIM, 2008.

[12] Leonardo Candela, Donatella Castelli, and Pasquale Pagano. Making virtual research environments in the cloud a reality: the gCube approach. *ERCIM News*, 83:32 – 33, 2010. In: ERCIM News, vol. 83 pp. 32 - 33. Special issue: Cloud Computing Platforms, Software, and Applications. Ercim, 2010.

[13] Leonardo Candela, George Kakaletris, Pasquale Pagano, Giorgios Papanikos, and Fabio Simeoni. The gCube interoperability framework. In Seamus Ross Donatella Castelli, Yannis Ioannidis, editor, *2nd DL.org Workshop - Making Digital Libraries Interoperable*, pages 35 – 42. Dl.org, 2010. In: 2nd DL.org Workshop - Making Digital Libraries Interoperable (Glasgow, Scotland, 9-10 September 2010). Proceedings, pp. 35 - 42. Donatella Castelli, Yannis Ioannidis, Seamus Ross (eds.). Dl.org, 2010.

[14] Fabio Simeoni, Leonardo Candela, David Lievens, Pasquale Pagano, and Manuele Simi. Functional adaptivity for digital library services in e-infrastructures: the gCube approach. In Agosti M. et al., editor, *Research and Advanced Technology for Digital Libraries. 13th European Conference*, pages 51 – 62. Springer Verlag, 2009. In: ECDL 2009 - Research and Advanced Technology for Digital Libraries. 13th European Conference (Corfu, Greece, 27 September - October 2 2009). Proceedings, pp. 51 - 62. Agosti M., Borbinha J., Kapidakis S., Papatheodorou C., Tsakonas G (eds.). (Lecture Notes in Computer Science, vol. 5714). Springer Verlag, 2009.