An Extension to the SAGA Service Discovery API

Antony Wilson
(Steve Fisher and Paul Livesey)

4th EGEE Users Forum – Catania 2009
• Purpose of SAGA
• Introduction to SAGA Service Discovery API
• An extension to the Service Discovery API
• Summary
• “Provide a simple API that can be used with much less effort compared to the vanilla interfaces of existing grid middleware. A guiding principle for achieving this simplicity is the 80 – 20 rule: serve 80% of the use cases with 20% of the effort needed for serving 100% of all possible requirements.”

• “Provide a standardized, common interface across various grid middleware systems and their versions.”

• SAGA is an OGF standard with various implementations
The specification for the Service Discovery has been finalised – GFD.144

- Loosely based around the gLite Service Discovery
- Uses the GLUE (version 1.3) model of a service
  - A Site may host many Services
  - A Service has multiple Service Data entries
  - Each Service Data entry is represented by a key and a value

APIs in C, C++, Java and Python complete but not released

Adapter for gLite under development
- Based around GLUE 1.3
  - Once GLUE 2 is finalised the adapter will be updated
• **API allows selection based on three filters:**
  - `serviceFilter` – allows filtering on:
    - type, name, uid, site, url, implementor and relatedService
  - `dataFilter` – no predefined values
    - Uses keys from Service Data entries
  - `authzFilter` – authorization, no predefined values, useful values include:
    - vo, dn, group and role (values dependent on adapter)
  - NB if an `authzFilter` is not provided then one is automatically constructed from the users security context
    - The gLite adapter will provide the VOMS proxy credentials as the default value for the security context

• Each of the filter strings uses SQL92 syntax
• The filters act as if they are part of a WHERE clause
• Selection returns a list of ServiceDescriptions
  – Each description contains:
    ▪ type, name, uid, site, url, implementor, list of relatedServices and service data (key value pairs)

• Example:
  discoverer = SDFactory.createDiscoverer()
  serviceDescriptions =
    discoverer.listServices("type = ‘computing service’", "")
  loop serviceDescriptions
    description.getAttribute("name")
    returns the value of the name attribute

• URL of information system can be passed in with the constructor, or obtained from a conf file
An Extension to the Service Discovery API
• **Problem**: The service discovery API only gives basic information as it cannot represent the GLUE model in three tables

• **Purpose**: To navigate the information model starting from a selected service

• **API will be independent of the underlying information system**

• **Different information systems supported by means of adapters**
  - We will provide a gLite adapter

• **Navigation will be from entity to entity as expressed in the GLUE entity relationship models**
- Computing Service is a Service
- Each box represents a set of entities
- Each entity has associated data
- EntityDataSet object relates to a box in the GLUE ER model
- These objects provide generalised solution for navigating an ER model
eds = new EntityDataSet(serviceDescription)

- Where the serviceDescription is a ServiceDescription of a “computing service”
EntityDataSet

- `getData()`: EntityData[]
- `getRelatedEntities(relatedName:string)`: EntityDataSet
- `getRelatedEntities(relatedName: String, filter: String)`: EntityDataSet
- `listRelatedEntityNames()`: String[]

EntityData

- `existsAttribute(key: String)`: boolean
- `findAttributes(patterns: String)`: String[]
- `getAttribute(key: String)`: String
- `getVectorAttribute(key: String)`: String[]
- `isVectorAttribute(key: String)`: boolean
- `listAttributes()`: String[]
dataSet = eds.getData()

returns a set of data entities, in this case it will be a set of one

loop dataSet

data.listAttributes()

returns ID, TotalJobs, RunningJobs, WaitingJobs …

data.getAttribute("ID")

returns the value of the ID attribute
Finding Related Entities

EntityDataSet
getDataSet():EntityData[]
getRelatedEntities(relatedName:string):EntityDataSet
getRelatedEntities(relatedName: String, filter: String): EntityDataSet
listRelatedEntityNames(): String[]

data 0..*
relatedEntity

EntityData
existsAttribute(key: String): boolean
findAttributes(patterns: String): String[]
getAttribute(key: String): String
getVectorAttribute(key: String): String[]
isVectorAttribute(key: String): boolean
listAttributes(): String[]
eds.listRelatedEntityNames()
returns ComputingEndpoint, ComputingShare and ComputingManager
Navigating to Related Entities

**EntityDataSet**

- `getData()`: EntityData[]
- `getRelatedEntities(relatedName:string)`: EntityDataSet
- `getRelatedEntities(relatedName:String, filter:String)`: EntityDataSet
- `listRelatedEntityNames()`: String[]

**EntityData**

- `existsAttribute(key:String)`: boolean
- `findAttributes(patterns: String)`: String[]
- `getAttribute(key:String)`: String
- `getVectorAttribute(key:String)`: String[]
- `isVectorAttribute(key:String)`: boolean
- `listAttributes()`: String[]
Navigating to Related Entities

- This is a one to many relationship
- A new EntityDataSet would be returned containing all of the computing managers

cmEds = eds.getRelatedEntities("ComputingManager")
dataSet = cmEds.getData()

returns a set of data entities, in this case there will be an entry for each computing manager
Filtered Selection

EntityDataSet

- `getData()`: EntityData[]
- `getRelatedEntities(relatedName:string)`: EntityDataSet
- `getRelatedEntities(relatedName:String, filter: String)`: EntityDataSet
- `listRelatedEntityNames()`: String[]

EntityData

- `existsAttribute(key: String)`: boolean
- `findAttributes(patterns: String)`: String[]
- `getAttribute(key: String)`: String
- `getVectorAttribute(key: String)`: String[]
- `isVectorAttribute(key: String)`: boolean
- `listAttributes()`: String[]
- Start with many computing managers
- Each computing manager can have many benchmarks
bmEds = cmEds.getRelatedEntities("BenchMark", "type='cint2006'")

Selects all the benchmark entities that have a type of 'cint2006'
• A related entity is not necessarily an adjacent entity
• We will provide predefined paths for useful associations
• Service Discovery API specification – DONE
• Service Discovery APIs – Q1 2009
• Service Discovery gLite Adapter for GLUE 2 – Q2/3 2009
• Extended Service Discovery API specification – Q2 2009
• Extended Service Discovery APIs Q3 2009
• Extended Service Discovery gLite Adapter – Q3/4 2009
• Service Discovery is used to make the initial service selection
• The Extended Service Discovery can then be used to:
  – Navigate around the entity relationship model
  – Retrieve data from a selected entity within the service
• We have given an example of how to use the API
• Java doc http://hepunx.rl.ac.uk/egee/sa3-uk/sd/saga-api-java/
• Navigation can be complex as there are “many to many” relationships
  – We hope our API will make navigation as straight forward as possible
• We would like feedback from potential users