GWD-R.xx
SAGA-RG

Steve Fisher and A Paventhan
Rutherford Appleton Laboratory, UK

18 May 2007
Updated: 2 Jan 2008

SAGA API Extension: Service Discovery API

Status of This Document

This document provides information to the grid community, proposing a standard for an extension to the Simple API for Grid Applications (SAGA). As such it depends upon the SAGA Core API Specification [2]. This document is intendeded to be used as input to the definition of language specific bindings for this API extension, and as reference for implementors of these language bindings. Distribution of this document is unlimited.

Copyright Notice

Abstract

This document specifies a Service Discovery API extension to the Simple API for Grid Applications (SAGA), a high level, application-oriented API for grid application development. This Service Discovery API is motivated by a number of Use Cases collected by the OGF SAGA Research Group in GFD.70 [4], and by requirements derived from these Use Cases, as specified in GFD.71 [5]). It allows users to find services with minimal prior knowledge.

Contents

1 Introduction

Most of the SAGA use cases [4] exhibit a need for service discovery (SD) - though it is sometimes described as resource discovery. For example the DiVA entry says:

> DiVA infrastructure must; a) Discover available components on distributed resources. The list of available components must be searchable by different attributes. This overlaps the needs of RealityGrid.

and:

> On startup, the application must gather a list of available "components". Typically this is done by consulting a local configuration file to find the locations of the binaries (or bytecode files) associated with each component as well as their names and interface definitions. For DiVA, we would like to support the discovery of remote modules as well by contacting information services on other machines or a broker that locates components on all machines in a given Virtual Organization. From the application programmers point of view, they want to be presented with a searchable database of components (regardless of location) that can be queried and sorted based on criteria

> such as “name”, “location”, interface definition, etc... Organization as an Relational Database or LDAP directory or even a flat-file is unimportant. The API should be able to hide these details as a query for components that satisfy the search criteria is presented.

This API extension is tailored to provide exactly this functionality, at the same time keeping coherence with the SAGA Core API look & feel, and keeping other Grid related boundary conditions (in particular middleware abstraction and authentication/authorization) in mind.

1.1 Notational Conventions

In structure, notation and conventions, this documents follows those of the SAGA Core API specification [2], unless noted otherwise.

1.2 Security Considerations

As the SAGA API is to be implemented on different types of Grid (and non-Grid) middleware, it does not specify a single security model, but rather provides hooks to interface to various security models – see the documentation of the `saga::context` class in the SAGA Core API specification [2] for details.

A SAGA implementation is considered secure if and only if it fully supports (i.e. implements) the security models of the middleware layers it builds upon, and neither provides any (intentional or unintentional) means to by-pass these security models, nor weakens these security models’ policies in any way.

2 SAGA Service Discovery API

2.1 Introduction

The SAGA Service Discovery API provides a mechanism to locate services.

The main SAGA APIs assume that certain URLs are known and will be passed in to those calls. For example, the constructor for the `saga::job_service` class takes the URL of a resource manager. The specification allows the implementation to find the resource manager if no URL is provided. It is, however, likely that more information from the user are required to obtain a suitable resource manager. We would expect that a saga::job_service implementation might also make use of this service discovery API. Another example where the user needs to locate a service is to make a `saga::rpc` call.

It is expected that this SD API will make use of various information systems or other service discovery mechanisms. The quality of the information returned will depend upon the quality of the data in the back-end system or systems.

2.1.1 Service Model

The API is based upon the GLUE (version 1.3) model of a service [1] as summarised in figure 1.

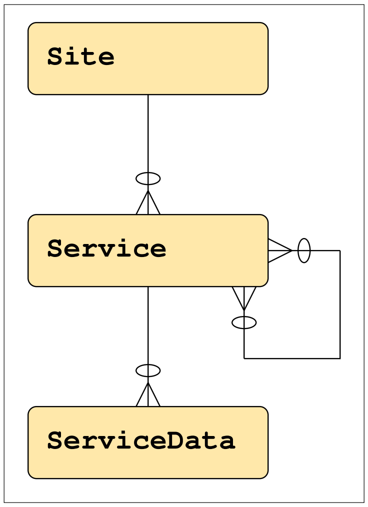

Figure 1: ER diagram of Service Model

The attributes are not shown as they as more subject to change as GLUE [3] evolves. The figure indicates that a *Site* may host many *Services* and a *Service* has multiple *ServiceData* entries associated with it. Each *ServiceData* entry is represented by a key and a value, thus allowing any set of keyword/value pairs to be associated with an instance of a *Service*. In addition, a *Service* has a many-to-many relationship with itself. This allows the model to describe groupings of services.[1]

2.1.2 Classes

The SAGA Service Discovery API consists of a `discoverer` class with a single method: `list_services()`. This returns a list of objects of the `service_description` class, filtered according to several specified filters. The `service_description` class has a method `get_url()` – which is all that most people will use to obtain the address registered for the service. In the case of a Web Service this will be the service endpoint. It also implements the `attribute` interface, and thus exposes additional properties of the service, such as service type, uid and others. These might be used by those who wish to generate a web page of services, or need detailed information for other purposes.

There is an operation `get_related_services` that returns the set of related `service_description`s, which represent related services. Finally, there is a method `get_data` to access the set of further key value pairs. This method returns an `service_data` object, which also implements the `attribute` interface and gives readonly access to all the key names and values in the *ServiceData*. By making the `service_description` implement the attribute interface and referencing a separate object holding the key value pairs, potential key name clashes between the two sets of attributes are avoided.

2.2 Specification

```
package saga.sd {

  class discoverer : implements saga::object
  {
    CONSTRUCTOR   (in  session    session,
                   out discoverer dis);
    DESTRUCTOR    (in  discoverer dis);
```

[1] It is possible that this Service Discovery API may be incompatible with a future version of GLUE; however the concepts required by the API are currently included in the working draft of GLUE 2.0. Future revisions of this document will address this issue.

```
      list_services (in  string     service_filter,
                     in  string     vo_filter,
                     in  string     data_filter,
                     out array<service_description> services);
    }

    class service_description : implements saga::object
                                implements saga::attribute
    {
      get_url              (out string       url);
      get_related_services (out array<service_description>
                                             services);
      get_data             (out service_data data);

      // Attributes:
      //   name:  url
      //   desc:  url to contact the service
      //   mode:  ReadOnly
      //   type:  String
      //   notes: The get_url() call obtains the same information.
      //
      //   name:  type
      //   desc:  type of service
      //   mode:  ReadOnly
      //   type:  String
      //   notes: The specification imposes no rules on the
      //          values of this field except that it must
      //          not be an empty string.
      //
      //   name:  uid
      //   desc:  unique identifier of service
      //   mode:  ReadOnly
      //   type:  String
      //   notes: The specification imposes no rules on the
      //          values of this field except that it must
      //          not be an empty string.
      //
      //   name:  site
      //   desc:  name of site
      //   mode:  ReadOnly
      //   type:  String
      //   notes: The specification imposes no rules on the
      //          values of this field except that it must
      //          not be an empty string.
      //
      //   name:  name
```

```
  //   desc:  name of service - not necessarily unique
  //   mode:  ReadOnly
  //   type:  String
  //   notes: The specification imposes no rules on the
  //          values of this field except that it must
  //          not be an empty string.
  //
  //   name:  related_services
  //   desc:  uid of related services
  //   mode:  ReadOnly, optional
  //   type:  Vector String
  //   value: -
  //   notes: This returns the uids of the related services.
  //          This is unlike the call get_related_services()
  //          which returns an array of service_descriptions.
  //
  //   name:  VO
  //   desc:  Names of Virtual Organisations able to use the
  //          service
  //   mode:  ReadOnly, optional
  //   type:  Vector String
  //   value: -
  //   notes: This returns the names of the VOs that may be
  //          able to use the service. Access to the service
  //          may be further controlled by an authorization
  //          mechanism; but this is outside the scope of
  //          this API.
  //
}

class service_data : implements saga::object
                     implements saga::attribute
{
  // no CONSTRUCTOR
  DESTRUCTOR (in service_data sd);

  // Attributes(extensible):
  //
  // no attributes pre-defined
}
```

2.3 Specification Details

class discoverer

The discoverer object is the entry point for service discovery. Apart from the constructor and destructor it has one operaration: list_services which returns the list of descriptions of services matching the specified filter strings.

There are three filter strings: service_filter, vo_filter and data_filter which act together to restrict the set of services returned.

Each of the filter strings uses SQL92 syntax as if it were part of a WHERE clause acting to select from a single table that includes columns corresponding to each key name in the key/value pairs as specified as attributes for the service_description class. If the programming language permits it, empty strings may be replaced by a representatation of NULL. SQL92 has been chosen because it is widely known and has the desired expressive power.

Three strings are used rather than one as this clarifies the description of the functionality, avoids problems with key values being themselves existing GLUE attributes, and facilitates implementation.

Only the following operators are permitted in the expressions: IN, LIKE, AND, OR, NOT, =, >=, >, <=, <, <> in addition to column names, parentheses, column values as single quoted strings, numeric values and the comma. An implementation should try to give an informative error message if the filter string does not conform it is, however, sufficient to report in which filter string the syntax error was found.

The LIKE operator matches patterns:

'%xyz' matches all entries with trailing xyz

'xyz%' matches all entries with leading xyz

'%xyz%' matches all entries with xyz being substring

Column names are not case sensitive but values are.

For matching on multivalued attributes it is sufficient that one attribute in the information system matches.

Service Filter

Column names in the `service_filter` are dependent upon the GLUE service definition. Only those attributes considered useful to service discovery are supported. For GLUE 1.2 these are:

type type of service. This API does not restrict values of the service type - it might be a DNS name, a URN or any other string.

name name of service (not necessarily unique)

uid unique identifier of service

site name of site the service is running at

url the endpoint to contact the service - will normally be used with the LIKE operator

related_services for related services. The user should specify the service's uid.

Some examples are:

- `type = 'org.glite.security.voms'`
- `site IN ('INFN-CNAF', 'RAL-LCG2')`
- `type = 'ResourceBroker' AND Site LIKE '%INFN%'`

VO Filter

There is only one column name in the `vo_filter` string:

vo Virtual Organization - will often be used with the IN operator. This API does not restrict the values of a VO - it might be a DNS name, a URN or any other string.

Some examples are:

- `VO IN ('cms', 'atlas')`
- `VO = 'dteam'`

Data Filter

Column names in the the `data_filter` string are taken from the service data key/value pairs. No keys are predefined by this specification.

If values are specified as numeric values and not in single quotes the service data will be converted from string to numeric for comparison.

Some examples are:

- `source = ’RAL-LCG2’ OR destination = ’RAL-LCG2’`
- `RunningJobs >=1  AND RunningJobs <= 5`

```
- CONSTRUCTOR
  Purpose:  create a new discoverer object
  Format:   CONSTRUCTOR     (in session session,
                             out discoverer dis);
  Inputs:   session:         session handle. If omitted the
                             default session will be used.
  Outputs:  dis:             new discoverer object
  Throws:   NotImplemented
            NoSuccess
  Notes:

- DESTRUCTOR
  Purpose:  Destructor for discoverer object.
  Format:   DESTRUCTOR      (in discoverer dis)
  Inputs:   dis:             object to be destroyed
  Outputs:  -
  Throws:   -
  Notes:    -

- list_services
  Purpose:  return the set of services that pass the set of
              specified filters
  Format:   list_services   (in  string  service_filter,
                             in  string  vo_filter,
                             in  string  data_filter,
                             out array<service_description>
                                         services);
  Inputs:   service_filter:  filter on the basic service and
                             site attributes and on related
                             services
```

```
                vo_filter:        filter on VOs associated with
                                  the service
                data_filter:      filter on key/value pairs
                                  associated with the service
      Outputs:  -
      Throws:   NotImplemented
                BadParameter
                AuthorizationFailed
                AuthenticationFailed
                NoSuccess
      Notes:    - if any filter has an invalid syntax, a
                  'BadParameter' exception is thrown.
                - if any filter uses invalid keys, a
                  'BadParameter' exception is thrown.
```

class service_description

The service_description class implements the SAGA attribute interface and offers getter methods for the user to obtain details of that service. The attributes are based on those found in GLUE. In addition it has the methods listed below.

```
   - get_url
      Purpose:  return the URL to contact the service
      Format:   get_url          (out string url);
      Inputs:   -
      Outputs:  url:             URL to contact the service
      Throws:   NotImplemented
                DoesNotExist
                NoSuccess
      Notes:    The URL may also be obtained using the
                attribute interface.

   - get_related_services
      Purpose:  return the set of related services
      Format:   get_related_services (out array<service_description>
                                                  services);
      Inputs:   -
      Outputs:  services:        set of related
                                 service_description objects
      Throws:   NotImplemented
```

```
             NoSuccess
  Notes:     This function returns an array of
             service_descriptions.  Alternatively, the
             attribute interface may be used to get the
             uids of the related services.

- get_data
  Purpose:   return a service_data object with the
             ServiceData key/value pairs
  Format:    get_data         (out service_data data);
  Inputs:    -
  Outputs:   data:            a service_data object
  Throws:    NotImplemented
             NoSuccess
```

class service_data

The service_data class implements the SAGA attribute interface and offers getter methods for the user to read key/value pairs defined by the service publisher. The service publisher is completely free to define his own key names. Access to the keys and values is through the attribute interface. The class provides no other methods. This class has no CONSTRUCTOR, as it can only be created by calling `get_service_data()` on a `service_description` instance.

```
 - DESTRUCTOR
  Purpose:   Destructor for service_data object.
  Format:    DESTRUCTOR      (in service_data sd)
  Inputs:    sd               object to be destroyed
  Outputs:   -
  Throws:    -
  Notes:     -
```

2.4 Examples

This C++ example shows how SAGA service discovery model can be used to retrieve services from the underlying information system. All the “Broker” services with a name of “CERN-PROD-rb” and owned by either “Atlas” or “DTeam” and for which the “RunningJobs” parameter is greater than 10 are requested. The service objects returned from the `list_services` call are then queried for attributes and key/values using its getter methods. It would be more common to issue a sufficiently precise query so that any service returned would be suitable and then call `get_url` on the first service returned.

Code Example

```
#include <iostream>
#include <vector>
#include <string>
#include <saga.hpp>
using namespace std;
using namespace saga;
main() {
  saga::discoverer d (SAGA_DEFAULT_SESSION);
  vector<string> attrib_names;
  vector<string> attrib_values;
  string svc_filter = "Type = ’Broker’ AND name = ’CERN-PROD-rb’";
  string vo_filter = "VO IN (’atlas’, ’dteam’)";
  string data_filter = "RunningJobs > 10";
  vector<saga::service_description> slist =
        d.list_services(svc_filter, vo_filter, data_filter);
  std::cout << "Total number of services found = " << slist.size()
        << endl;
  for (int i = 0; i < slist.size(); i++) {
    attrib_names = slist[i].list_attributes();
    cout << "SERVICE #" << i <<  endl;
    cout << "----------------------------------------------" << endl;
    for (int j = 0; j < attrib_names.size(); j++) {
      if (slist[i].attribute_is_vector(attrib_names[j])) {
        attrib_values = slist[i].get_vector_attribute(attrib_names[j]);
        cout << attrib_names[j] << ":" << endl;
        for (int k = 0; k < attrib_values.size(); k++)
          cout << attrib_values[k] << endl;
      } else {
        cout << attrib_names[j] << " = " <<
            slist[i].get_attribute(attrib_names[j]) << endl;
      }
    }
    cout << "----------------------------------------------" << endl;
  }
}
```

This C example is equivalent to the C++ one above.

Code Example

```
SAGA_SD_Discoverer *sd = SAGA_SD_create_discoverer(session_handle);

```

```
if (sd == NULL) {
  fprintf(stderr, "Could not create SAGA SD object: %s",
                              SAGA_Session_get_error(session_handle));
  return -1;
}

char service_filter[] = "Type = ’Broker’ AND Name = ’CERN-PROD-rb’";
char vo_filter[] = "VO IN (’atlas’, ’dteam’)";
char data_filter[] = "RunningJobs > 10";

SAGA_SD_ServiceDescription *slist = SAGA_SD_list_services(
      sd, service_filter, vo_filter, data_filter);

printf("Total number of services found : %d\n", slist->size);

for (int i = 0; i < slist->size; i++) {
  printf("SERVICE #%d\n", i);
  printf("---------------------------------------------");
  SAGA_SD_Attribute *keys = SAGA_SD_list_attributes(slist[i]);
  for (int j = 0; j < keys->size; j++) {
    if (SAGA_SD_attribute_is_vector(keys->names[j])) {
      SAGA_SD_Values *values = SAGA_SD_get_vector_attribute(slist[i],
             keys->names[j]); printf("%s: ", key->names[j]);
      for (int k = 0; k < values->size; k++) {
        printf("    %s\n", values->value[k]);
      }
      SAGA_SD_free_values(values);
    } else {
        printf("%s = %s\n", key->names[j],
             SAGA_SD_get_attribute(slist[i], key->names[j]));
    }
  }
  printf("---------------------------------------------");
  SAGA_SD_free_attributes(keys);
}
SAGA_SD_free_services(slist);
```

3 Intellectual Property Issues

3.1 Contributors

This document is the result of the joint efforts of several contributors. The authors listed here and on the title page are those committed to taking permanent stewardship for this document. They can be contacted in the future for inquiries about this document.

Steve Fisher
s.m.fisher@rl.ac.uk
Rutherford Appleton Lab
Chilton
Didcot
Oxon
OX11 0QX
UK

A Paventhan
paventhan@rl.ac.uk
Rutherford Appleton Lab
Chilton
Didcot
Oxon
OX11 0QX
UK

We wish to thank Pascal Kleijer (NEC Corporation) and Andre Merzky (Vrije Universiteit, Amsterdam) for making written comments on earlier drafts and encouraging us to be true to the SAGA style.

3.2 Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

3.3 Disclaimer

This document and the information contained herein is provided on an "As Is" basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

3.4 Full Copyright Notice

References

[1] S. Anreozzi et al. GLUE Schema Specification version 1.3. OGF: https://forge.gridforum.org/sf/go/doc14185?nav=1, 2007.

[2] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, and C. Smith. A Simple API for Grid Applications (SAGA). Grid Forum Document GFD.90, 2007. Global Grid Forum.

[3] B. Konya, L. Field, and S. Andreozzi. GLUE working group of the OGF. http://forge.gridforum.org/sf/projects/glue-wg.

[4] A. Merzky and S. Jha. A Collection of Use Cases for a Simple API for Grid Applications. Grid Forum Document GFD.70, 2006. Global Grid Forum.

[5] A. Merzky and S. Jha. A Requirements Analysis for a Simple API for Grid Applications. Grid Forum Document GFD.71, 2006. Global Grid Forum.