Steve Fisher and Antony Wilson

Rutherford Appleton Laboratory, UK

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 [3]. This document is intended 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.

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 of the grid or grids they plan to use.

Contents

1 Introduction

Most of the SAGA use cases [4] exhibit a need for service discovery (SD) - though it is sometimes described in other terms, such as searching for resources, or even searching for components. It would appear that the real need is to discover services. The fact that these services may make use of certain resources or components is not the prime interest of the the end user – ultimately access will be via the interface offered by the service.

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 [3], 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 [3] 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::job_service` class takes the URL of a resource manager. The specification of the job_service class allows the implementation to find the resource manager if no URL is provided. A user could therefore delegate to the job_service the task of choosing a suitable resource manager or, if more control on the choice of resource manager is required, the user could use the service discovery API directly and pass in the URL of the resource manager. It is likely that an implementation of the job_service would itself use the service discovery API to locate a resource manager when none has been specified. Another example where the user needs to locate a service is to make a `saga::rpc::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 summarized in figure 1. This service model is also compatible with GLUE 2.0 [2].
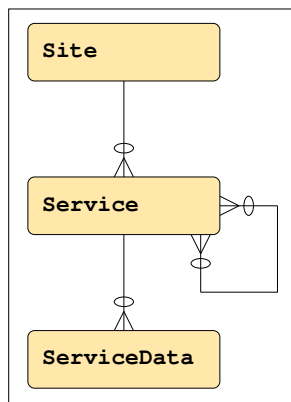

Figure 1: ER diagram of Service Model

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.

It is possible that the Service Discovery API may be incompatible with a future version of GLUE. This issue will be addressed, if necessary, in a future revision of this document.

2.1.2 Classes

The SAGA Service Discovery API consists of a `discoverer` class with a single method: `list_services()`. This method returns a list of objects of the `service_description` class, filtered according to several specified filters.

The `service_description` class has three methods. One of these, `get_url`, 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 its type and uid. These might be used by those who wish to generate a web page of services, or need detailed information for other purposes. There is a method `get_related_services` that returns the set of related `service_descriptions`, which represent related services. Finally, there is a method `get_data` to access the set of further key value pairs. This method returns a `service_data` object, which also implements the `attribute` interface giving *ReadOnly* access to all the key names and values in the `service_data` object.

By making the `service_description` implement the attribute interface, and by referencing a separate `service_data` 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);
```

```
    CONSTRUCTOR   (in  string     url,
                   out discoverer dis);
    CONSTRUCTOR   (in  session    session,
                   in  string     url,
                   out discoverer dis);
    CONSTRUCTOR   (out discoverer dis);
    DESTRUCTOR    (in  discoverer dis);

    list_services (in  string     service_filter,
                   in  string     data_filter,
                   in  string     authz_filter,
                   out array<service_description> services);

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

  class service_description : implements saga::object
                              implements saga::attribute
  {
    // no CONSTRUCTOR
    DESTRUCTOR (in service_description sdesc);

    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. In addition, all
    //          SAGA services should use types constructed
    //          with the form: ``saga::<package>::<class>''
    //          where <package> and <class> are the names
```

```
//           of the package and class providing the
//           service.
//
//    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:  implementor
//    desc:  name of the organisation providing the
//           implementation of the 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:  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.
  }
```

```
    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

The API will typically use some underlying information system and should not
contact the services to check their availability. The user must expect that a
service provided by the Service Discovery API may not be available. Even
if the API were to contact a service to confirm its availability, by the time
the user attempts to use that service, it may have failed. Similarly the API
cannot be guaranteed to provide a complete set of matching services - it is the
responsibility of the implementation to apply any algorithm it chooses to return
a set of services.

The API may try to use an underlying information system but not be able to
access it. The precise behaviour is implementation dependent - for example if it
uses adapters it may try a different one. If no result can be returned because of
information system or other internal problems, it should throw the `NoSuccess`
exception.

class discoverer

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

An implementation should return the results in a random order if there is more
than one result to avoid any tendency to overload particular services while
leaving others idle.

There are three filter strings: `service_filter`, `data_filter` and `authz_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 as described below for that filter type. SQL92 has been chosen because it is widely known and has the desired expressive power. Multi-valued attributes are treated as if there were multiple entries in the table, one for each combination of values of the multi-valued attributes. A service will only be included once in the returned list of services.

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 as it makes it impossible to specify constraints that correlate, for example, service and authz information. 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 string 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.

Note that the treatment of multi-valued attributes defined above may not always be intuitive. For example if an attribute, q has values 'a' and 'b' and `q <> 'a'` is specified in a filter, then the service will be accepted because the entry in the table where q has value 'b' is consistent with the filter.

No use-case has been identified for the operators `>=`, `>`, `<=`, `<` to be applied to strings. An Implementation wishing to support these comparison operators on strings must select a collation sequence. Alternatively an implementation may treat all string comparisons as true, or reject them as invalid SQL.

Service Filter

Column names in the `service_filter` 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. However, all SAGA services should use types constructed with the form:

```
org.ogf.saga.service.<service_type_name>
```

where `<service_type_name>` is the name of the type of SAGA service. The names must be entirely lower case and must start with a letter optionally followed by letters, digits and hyphens. The last character must not be a hyphen. Service type names for the core services defined in GFD.90 are defined in table 1 below. For each `<service_type_name>`, the package and class implementing that service is shown. Other SAGA packages are responsible for defining their own service names within their specifications.

Name	Package	Class
job	saga.job	job_service
file	saga.file	file and directory
logical-file	saga.logical_file	logical_file and logical_directory
stream	saga.stream	stream_service
rpc	saga.rpc	rpc

Table 1: SAGA service type names.

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

implementor name of the organisation providing the implementation of the service

related_service the uid of a service related to the one being looked for

Some examples are:

- `type = 'org.ogf.saga.service.job'`

- `site IN ('INFN-CNAF', 'RAL-LCG2')`

- `type = 'org.glite.ResourceBroker' AND Site LIKE '%.uk' AND implementor = 'EGEE'`

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`

Authz Filter

The set of column names in the `authz_filter` is not defined. Instead the list below shows a possible set of names and how they might be interpreted. Each of these column names could reasonably be related to an authorization decision.

vo virtual organization - will often be used with the IN operator

dn an X.509 "distinguished name"

group a grouping of people within a Virtual Organization

role values might include "Administrator" or "ProductionManager

Some examples are:

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

- `VO = 'dteam'`

The `list_services` operation is overloaded: the last parameter the `authz_filter` may be omitted. If it is omitted the authorization filtering is performed on the contexts in the session. This is quite different from including the `authz_filter` parameter with an empty string which means that there is *no* authz filtering.

```
  - CONSTRUCTOR
    Purpose:  create a new discoverer object
    Format:   CONSTRUCTOR     (in session session,
                               out discoverer dis);
    Inputs:   session:        session handle
    Outputs:  dis:            new discoverer object
    Throws:   NotImplemented
              NoSuccess
```

Notes:

- CONSTRUCTOR
  Purpose: create a new discoverer object
  Format: CONSTRUCTOR   (in string url,
                               out discoverer dis);
  Inputs: url: URL to guide the implementation
  Outputs: dis: new discoverer object
  Throws: NotImplemented
            NoSuccess
  Notes: The url specified as in input parameter is to
            assist the implementation to locate the underlying
            information system such that it can be queried.

- CONSTRUCTOR
  Purpose: create a new discoverer object
  Format: CONSTRUCTOR   (in session session,
                               in string url,
                               out discoverer dis);
  Inputs: session: session handle
            url: URL to guide the implementation
  Outputs: dis: new discoverer object
  Throws: NotImplemented
            NoSuccess
  Notes: The url specified as in input parameter is to
            assist the implementation to locate the underlying
            information system such that it can be queried.

- CONSTRUCTOR
  Purpose: create a new discoverer object
  Format: CONSTRUCTOR   (out discoverer dis);
  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   data_filter,
                               in  string   authz_filter,
                               out array<service_description>
                                             services);
   Inputs:    service_filter:  filter on the basic service and
                               site attributes and on related
                               services
              data_filter:     filter on key/value pairs
                               associated with the service
              authz_filter:    filter on authorization information
                               associated with the service
   Outputs:   -
   Throws:    NotImplemented
              BadParameter
              AuthorizationFailed
              AuthenticationFailed
              NoSuccess
   Notes:     - The last parameter, the authz_filter, may be
                omitted. In this case an implicit authz_filter is
                constructed from the contexts of the session.
              - If any filter has an invalid syntax, a
                'BadParameter' exception is thrown.
              - If any filter uses invalid keys, a
                'BadParameter' exception is thrown. However
                the data_filter never signals invalid
                keys as there is no schema with permissible
                key names.
```

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. This class has no CONSTRUCTOR as objects of this type are created only by other objects in the service discovery API.

```
   - DESTRUCTOR
     Purpose:  Destructor for service_data object.
     Format:   DESTRUCTOR      (in service_description sdesc)
```

```
      Inputs:   sdesc                  object to be destroyed
      Outputs:  -
      Throws:   -
      Notes:    -

  - 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
    Notes:    -
```

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. Service publishers are completely free to define their 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, using a possible C++ binding, how the SAGA service discovery model can be used to retrieve services from the underlying information system. All the SAGA job services (org.ogf.saga.service.job) with a name of "CERN-PROD-rb" and accessible by a context of the current session 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.

Note that this example is a little artificial as it would be more sensible 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 ───
1     #include <iostream>
2     #include <vector>
3     #include <string>
4     #include <saga.hpp>
5
6     using namespace std;
7
8     main() {
9       saga::sd::discoverer d (SAGA_DEFAULT_SESSION);
10      vector<string> attrib_names;
11      vector<string> attrib_values;
12      string svc_filter = "type = 'org.ogf.saga.service.job'
13                      AND name = 'CERN-PROD-rb'";
14      string data_filter = "RunningJobs > 10";
15      vector<saga::sd::service_description> slist =
16            d.list_services(svc_filter, data_filter);
17      cout << "Total number of services found = " << slist.size()
18            << endl;
19      for (int i = 0; i < slist.size(); i++) {
20        cout << "SERVICE #" << i <<  endl;
21        attrib_names = slist[i].list_attributes();
22        for (int j = 0; j < attrib_names.size(); j++) {
23            cout << "  " << attrib_names[j] << " = " <<
24                slist[i].get_attribute(attrib_names[j]) << endl;
25        }
26      }
27    }
```

This C example is similar to the C++ one above but this time includes the authz filter parameter. This is just an extract from a possible C binding.

```
                          ─── Code Example ───
1    SAGA_SD_Discoverer *sd =
2           SAGA_SD_create_discoverer(SAGA_DEFAULT_SESSION);
3
4    if (sd == NULL) {
5      fprintf(stderr, "Could not create SAGA SD object: %s",
6            SAGA_Session_get_error(SAGA_DEFAULT_SESSION));
7      return -1;
8    }
9
10   char service_filter[] = "type = 'org.ogf.saga.service.job'
11                          AND name = 'CERN-PROD-rb'";
12   char authz_filter[] = "VO IN ('atlas', 'dteam')";
13   char data_filter[] = "RunningJobs > 10";
14
15   SAGA_SD_ServiceDescription *slist = SAGA_SD_list_services(
16           sd, service_filter, data_filter, authz_filter);
17
18   printf("Total number of services found : %d\n", slist->size);
19
20   for (int i = 0; i < slist->size; i++) {
21     printf("SERVICE #%d\n", i);
22     SAGA_SD_Attribute *keys = SAGA_SD_list_attributes(slist[i]);
23     for (int j = 0; j < keys->size; j++) {
24        printf("   %s = %s\n", key->names[j],
25               SAGA_SD_get_attribute(slist[i], key->names[j]));
26     }
27     SAGA_SD_free_attributes(keys);
28   }
29   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	Antony Wilson
dr.s.m.fisher@gmail.com	antony.wilson@stfc.ac.uk
Rutherford Appleton Lab	Rutherford Appleton Lab
Chilton	Chilton
Didcot	Didcot
Oxon	Oxon
OX11 0QX	OX11 0QX
UK	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. We are also indebted to A. Paventhan who worked on the original draft and implementation.

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. Andreozzi et al. GLUE Schema Specification version 1.3. `https://forge.gridforum.org/sf/go/doc14185?nav=1`, 2007.

[2] S. Andreozzi et al. GLUE Schema Specification version 2.0. `http://www.ogf.org/Public_Comment_Docs/Documents/2008-06/ogf-glue-2_public_comment.pdf`, 2008. This is the May 2008 version submitted for public comment.

[3] 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, 2008. Open Grid Forum.

[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.