GWD-R.xx
SAGA-WG

Version: 0.1

Andre Merzky
CCT/LSU

June 23, 2013

SAGA Resource Management 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 [1]. This document is supposed 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 Resource Management API extension package for the Simple API for Grid Applications (SAGA), a high level, application-oriented API for distributed application development. This Resource Management (RM) API is motivated by a number of use cases collected by the former OGF SAGA Research Group in GFD.70 [2], and by requirements derived from these use cases, as documented in GFD.71 [3]). Also, the SAGA community has been receiving additional new use cases, in particular related to virtualized resources, pilot jobs, and advanced reservation, which call for a revision of the existing SAGA approch to resource and job management.

Contents

1 Introduction

For dynamic resource provisioning scenarios, the `saga::job::service` class from SAGA Core [1] proves to be insufficient, as it does not expose the means to manipulate resource state and lifetime – that is, however, at the very heart of a number of novel SAGA use cases.

First of all, we have seen an increasing acceptance and uptake of the pilot job paradigm. Amongst others, pilot job implementations based on SAGA have been relatively successfull, as they portably provide the pilot job paradigm on a variety of infrastructures, for concurrent use. Pilot jobs however are stateful, and can thus not easily be modeld by the old `job_service` class.

Further, the new iteration of the DRMAA API specification (version 2.0 [4]) is adding the capability for advanced reservation. DRMAA is one important specification upon which SAGA is originally built, and advanced reservation is, although seldom provided on system level, a very desireable programming abstraction for several SAGA use cases. Advanced reservations have, however, similar state properties as pilot jobs, although with different SLA's: they are guaranteed to become available for a specific time frame in the future.

Finally, but also prominently, there is a large set of cloud use cases, in particular on the IaaS level, which seem to *almost* map to the `saga::job` package, apart again from the notion of state which is attached to the dynamically provisioned virtual resources.

So it seems prudent and timely to attempt a new and unified approach to SAGA's original job submission and management package, which should cover that extendet set of use cases. Several boundary conditions for such an approch apply, however:

- The API should, as far as possible, be kept backward compatible with the existing `saga::job` package;
- The API should be careful not to artificially *force* a unification the mentioned classes of use cases, but should rather make semantic differences explicit, if needed.
- Resource management is often (correctly) considered to be a system level concern. This API should be strictly limited to the user aspects of resource management, i.e. to that semantic subset which is frequently exposed at the user level.

This specification document defines such a Resource API. Its concepts, however, go beyond the discussed set of compute-centric use cases: it additionally attemtps to make the extension suitable for data-intensive use cases, where data and network resources are handled in par with compute resources.

1.1 Notational Conventions

In structure, notation and conventions, this documents follows those of the SAGA Core API specification [1], unless noted otherwise.

The names 'SAGA Resource API' and 'SAGA Resource Management API' are used synonymously, and refer to different aspects of the same API defined in this document. In general, the API will not be able to perform low level management operations on remote resources, but is rather targeting the management of user controlled slices on those resources.

1.2 Security Considerations

As the SAGA API is to be implemented on different types of distributed middleware systems, 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 [1] 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.

1.3 Definitions and Terminology

FIXME: complete this list

- **resource slice:** a limited, finite part of a resource which can be under user control.
- **slot:** a positive integer number, describing the optimal number of processes on a compute resource or compute resource slice (optimizing efficiency and performance). That number usually equals the number of logical or physical CPU cores.

2 SAGA Resource API

2.1 Overview

As discussed in the introduction, the SAGA Resource API introduces a notion of stateful resources. Those resources can be reservations one some system's queue, time slices of a system otherwise obtained, dynamically provisioned physical or virtual hardware, or actually also classic, not time constrained job submission endpoints (for backward compatibility, see below).

2.1.1 Backward Compatibility

The SAGA Resource API as defined by this document supersedes the SAGA Job API, but is designed to be backward compatible: the `saga::resource::compute` class extends the `saga::job::service` class, and thus allows for exactly the same operations (and more). Semantically fully equivalent instances of the `saga::resource::compute` class can also be created, either directly from a `job::service` instance, or from that job service's contact URL – that URL is then used as that `saga::resource::compute`'s ID. All jobs which are created on `saga::resource::compute` instances are also the well known `saga::job::job` instances.

2.1.2 Resource States

Resources are stateful entities for this API, and state transitions follow a well defined state model (see fig. 1).

2.1.3 Classes

The SAGA Job-Resource API consists of three main classes: a `manager` class, which represents an entity which provides (i.e. finds, acquires, releases) `resource` instances – which represent the second class of this API package. Multiple resource can be combined into a resource `pool`, which itself extends the `resource` class by some pool management methods, but otherwise behaves equivalently (it is-a resource). Finally, a `resource::description` class is used to inform the manager of the properties of requested resources.

The `resource` class is subclassed to render `compute`, `network` and `storage` resources, and, as mentioned, resource `pools`. Similarly, the `description` class is subclassed to `compute_description`, `storage_description`, and `network_description`.

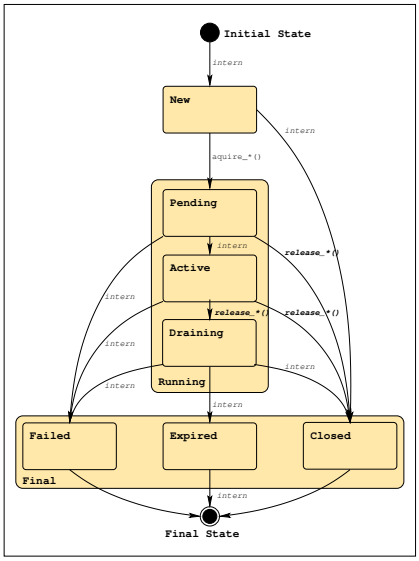

Figure 1: **resource states and state transitions**

A `compute` resource inherits the old `saga::job::service` class for backward compatibility, and is indeed functionally very similar to it. Similarly, the `storage` resource inherits the old `saga::filesystem::directory` class, which makes a storage resource accessible to file system manipulations (copy, more, ...).

The resource `pool` effectively acts as a collection of compute, data and network resources, with job submission capabilities and data management capabilities.

2.2 Specification

```
package resource
{
  enum type
  {
    Compute  =  1,  // accepting jobs
    Network  =  2,  // connectes compute(s) and storage(s)
    Storage  =  3   // mounted on / accessible by compute
  };

  enum state
  {
    Unknown  =   0,  // wut?
    New      =   1,  // will become active eventually
    Pending  =   2,  // will become active eventually
    Active   =   4,  // accepting jobs, jobs can run
    Draining =   8,  // jobs still run, not accepting new jobs
    Running  =  15,  // Pending | Active | Draining
    Closed   =  16,  // closed by user
    Expired  =  32,  // closed by system
    Failed   =  64,  // closed unexpectedly by system or internally
    Final    = 112   // Closed | Expired | Failed
  };


  class description : implements saga::attributes
  {
    // FIXME: use SAGA's SIDL attribute notation
    enum             type         = ''  // reuired
    array<string>    template     = ''
    bool             dynamic      = 'false'
    string           placement    = ''

    time             start        = 'now'
    time             end          = ''
    time             duration     = ''
  };

  class compute_description : public class description
  {
    enum             machine_os   = 'Any'
    enum             machine_arch = 'Any'
    int              size         = '1' // number of slots
```

```
  array<string>    hostnames    = ''
  long             memory       = ''
  string           access       = ''  // fqhn or ip
}

class network_description : public class description
{
  int              size         = ''  // number of IPs
  string           access       = ''  // network device
}

class storage_description : public class description
{
  int              size         = ''  // size in bytes /kB/MB/...
  string           access       = ''  // mnt point or
                                      // provisioning url
}
```

```
////////////////////////////////////////////////////////////////////
//
// The resource manager can translate resource requests into
// stateful resource handles.  It also manages the
// persistency of those resource handles, and of resource
// pools.
//
class  manager : implements saga::object,
               , implements saga::task::async
{
  //----------------------------------------------------------------
  CONSTRUCTOR          (in  session             session,
                        in  string              url = "",
                        out manager             obj);
  DESTRUCTOR           (in  manager             obj);

  //----------------------------------------------------------------
  // list known pilot/vm/ar instances etc. (which can be aquired)
  list_resources       (in  type                type = 'Any',
                        out array<string>       ids);

  // see drmaav2::machine_info?  Add GLUE inspection as
  // read-only attribs?  link to SD or ISN?
  get_resource_description
                       (in  string              id,
                        out description         rd);

  // list available templates
  list_templates       (in  type                type = 'Any',
                        out array<string>       tmpls);

  // human readable description of template
  get_template         (in  string              tmpl,
                        out description         rd);

  //----------------------------------------------------------------
  // aquire compute resource matching from requirements
  aquire_compute       (in  compute_description cd,
                        out compute             cr);

  // return resource handle for some known compute resource.
  // (id can also be old fashioned job::services urls)
  aquire_compute       (in  string              id,
```

```
                        out compute             cr);

    // close compute resource
    release_compute     (in  string             id,
                         in  bool               drain='false');


    //-----------------------------------------------------------------
    // storage resources
    aquire_storage      (in  storage_descrption  nd,
                         out storage            sr);
    aquire_storage      (in  string             id,
                         out storage            sr);
    release_storage     (in  string             id);


    //-----------------------------------------------------------------
    // network resources
    aquire_network      (in  network_descrption  nd,
                         out network            nr);
    aquire_network      (in  string             id,
                         out network            nr);
    release_network     (in  string             id);
    //-----------------------------------------------------------------
  }
```

```
////////////////////////////////////////////////////////////////////
//
class resource : implements saga::monitorable
               , implements saga::task::async
{
  CONSTRUCTOR          (in  session             session,
                        in  string              id,
                        out resource            obj);
  DESTRUCTOR           (in  resource            obj);

  //-----------------------------------------------------------------
  // inspection and state management
  get_id               (out string              id);
  get_type             (out type                t);
  get_state            (out state               s);
  get_state_detail     (out string              sd);
  get_manager          (out manager             m);
  get_access           (out <list>string        url);
  get_description      (out description         rd);

  reconfig             (in  description         rd);

  release              (in  bool                drain   = false);
  wait                 (in  double              timeout = -1.0,
                        in  int                 state   = Final);

  // metrics: state, state_detail
  // attribs: type, id, manager_id, description
  //-----------------------------------------------------------------
}
```

```
////////////////////////////////////////////////////////////////////
//
//
//
class compute : implements saga::resource::resource
              , implements saga::task::async
{
}


////////////////////////////////////////////////////////////////////
//
//
//
class storage : implements saga::resource::resource
              , implements saga::task::async
{
}


////////////////////////////////////////////////////////////////////
//
//
//
class network : implements saga::resource::resource
              , implements saga::task::async
{
}
```

2.3 Specification Details

2.3.1 `saga::resource::description`

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.

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] 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.

[2] 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.

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

[4] P. Tröger, R. Brost, D. Gruber, M. Mamoński, and D. Templeton. Distributed Resource Management Application API Version 2 (DRMAA). Grid Forum Document GFD.194, 2012. Global Grid Forum.