GWD-R.xx
SAGA-WG

Version: 0.1

Andre Merzky
CCT/LSU

September 18, 2011

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 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) 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 seservations have, however, similar state properties as pilot jobs, although with different SLA's: they become available for a specific time frame in the future, and can be closed/freed if not needed any longer.

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 new package should, as far as possible, be kept backward compatible with the exitsing `saga::job` package;
- the new package should not *force* a unification the mentioned use cases, but rather try to make semantic differences, where they exist, explicit.

This specification document defines that revised Resource API. Its concepts go, however, somewhat beyond the set of dicussed use cases: it additionally attemtps to make the extension suitable for data-intensive use cases, where data resources are handled in par with compute resources, and where applications can benefit from explicit and implicit expressions of data-compute affinities.

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 controled slices on those remote 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.

2 SAGA Resource API

2.1 Overview

As discussed in the introduction, the SAGA Resorce Management introduces a notion of stateful resources. Those resources can be reservations one some systems'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 RM API as defined by this document supercedes 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 intities 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, creates, destroys) `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.

Additionally, a `resource::description` class is used to inform the manager of the properties od requested resource slices. Further, the `resource` class is subclassed, to render `compute`, `network` and `storage` resources. A `compute` resource inherits the old `saga::job::service` for backward compatibility, and is indeed functionally very similar to it. The `resource::pool` class inherits the

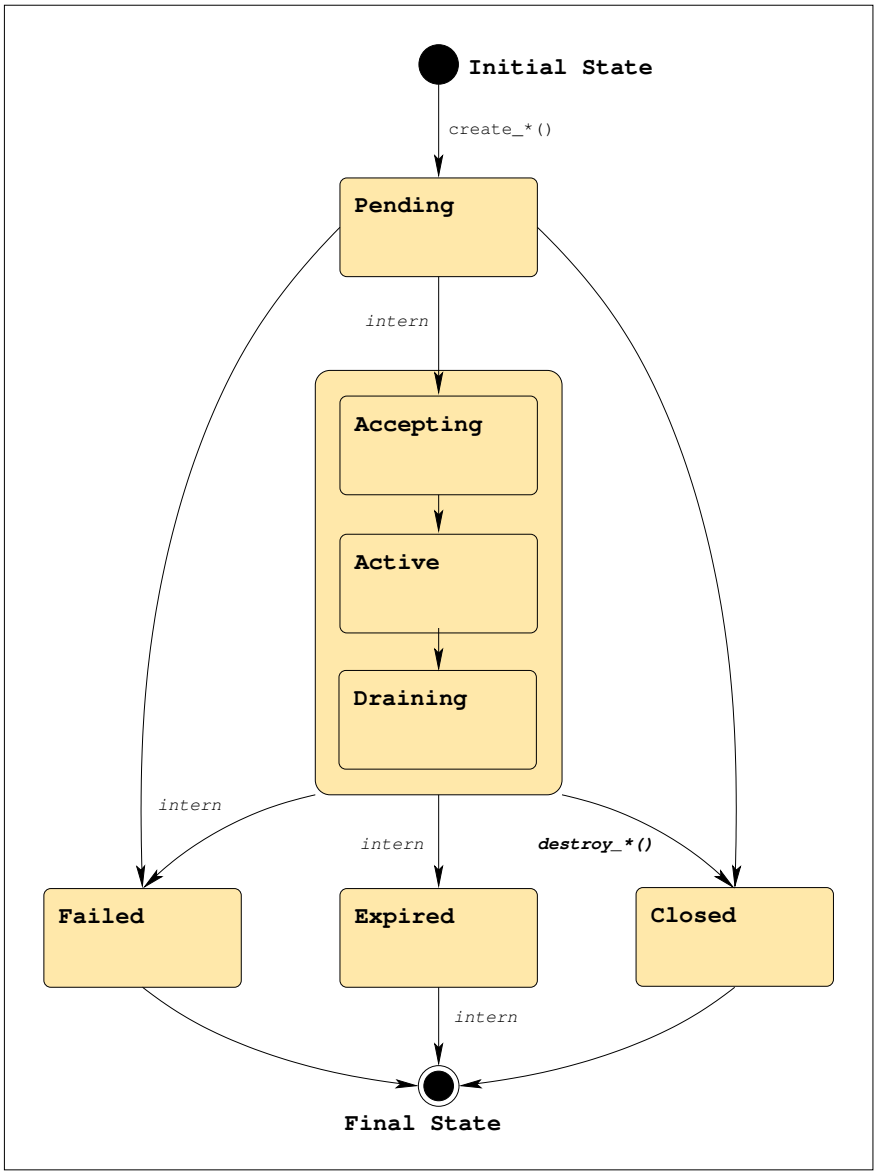

Figure 1: **resource states and state transitions**

`compute` class, but can also manage other resource subtypes. Effectively, a `pool` acts as a collection of compute, data and network resources with job submission capabilities.

2.2 Specification

```
package resource
{
  class  description : implements saga::attributes
  {
    // FIXME: use SAGA's SIDL attribute notation
    string         type
    string         template = 'default'

    enum           machine_os
    enum           machine_arch
    int            cores
    array<string>  hostnames
    long           memory

    time           start
    time           end
    time           duration

    string         pj_executable
    array<string>  pj_args
    array<string>  pj_env
    string         pj_queue

    bool           dynamic = 'true';
  };

  enum state
  {
    Unknown  = 0,  // wut?
    Pending  = 1,  // will become active eventually
    Accept   = 2,  // accepting jobs, jobs are queued
    Active   = 4,  // accepting jobs, jobs can run
    Draining = 5,  // closing, not accepting new jobs
    Closed   = 6,  // closed by user
    Expired  = 7,  // closed by system
    Failed   = 8   // disappeared etc.
  };
```

```
////////////////////////////////////////////////////////////////////
//
// 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 bigjob/vm/ar instances etc. (which can be opened)
  list_ids             (in  string             wildcard = "*",
                        out array<string>      ids);

  // list available templates (e.g. 'diane'
  list_templates       (out array<string>      tmpls);

  // human readable description of template
  // (e.g. http://diane-project.org/)
  get_details          (int string             tmpl);

  //-----------------------------------------------------------------
  // create compute resource matching from requirements
  create_compute       (in  description        rd,
                        out compute            jr);

  // create compute resource from saga::job::service,
  // --> 'backward interoperable'
  create_compute       (in  saga::job::service js,
                        out compute            jr          );

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

```
  // close compute resource
  destroy_compute      (in  string             id,
                        in  bool               drain);


  //--------------------------------------------------------------------
  // network resources (tmpl = ether, myrinet, ...)
  create_network       (in  int                size, // number of hosts
                        in  string             template = "default",
                        out network            nr);
  get_network          (in  string             id,
                        out network            nr);
  destroy_network      (in  string             id);


  //--------------------------------------------------------------------
  // storage resources (tmpl = disk, tape, s3, ...)
  create_storage       (in  int                size,
                        in  string             template = "default",
                        out storage            sr);
  get_storage          (in  string             id,
                        out storage            sr);
  destroy_storage      (in  string             id);
  //--------------------------------------------------------------------
}
```

```
////////////////////////////////////////////////////////////////////
//
// get_description adds resource_type=pool/compute/network/... etc.
//
class resource : implements saga::monitorable
{
  //----------------------------------------------------------------
  // inspection and state management
  get_state            (out state              s);
  get_state_detail     (out string             sd);
  get_id               (out string             id);
  get_manager          (out manager            m);
  get_description      (out description        rd);

  reconfig             (in  description        rd);

  destroy              (in  bool               drain);
  wait                 (in  double             timeout = -1.0);

  // metrics: state, state_detail
  //----------------------------------------------------------------
}
```

```
////////////////////////////////////////////////////////////////////
//
//
//
class compute : implements saga::resource::resource
              , implements saga::job::service
{
  //-----------------------------------------------------------------
  // native JSDL support
  submit               (in  string            jsdl);


  //-----------------------------------------------------------------
  // manage network on jr (jr.type == iaas)
  attach_network       (in  string            id,
                        in  string            iface    = "eth0");
  get_network_id       (in  string            iface    = "eth0",
                        out string            id);
  get_network_iface    (in  string            id,
                        out string            iface);
  detach_network       (in  string            id);


  //-----------------------------------------------------------------
  // manage network on jr (jr.type == iaas)
  attach_storage       (in  string            id,
                        in  string            mnt      = "/scratch");
  get_storage_id       (in  string            mnt      = "/scratch",
                        out string            id);
  get_storage_mount    (in  string            id,
                        out string            mnt);
  detach_storage       (in  string            id);
  //-----------------------------------------------------------------
}
```

```
////////////////////////////////////////////////////////////////////
//
//
//
class network : implements saga::resource::resource
{
  //----------------------------------------------------------------
}


////////////////////////////////////////////////////////////////////
//
//
//
class storage : implements saga::resource::resource
{
  //----------------------------------------------------------------
  // file staging, persistant over multiple job instances
  stage_in             (in  url               src,
                        in  url               tgt,
                        in  bool              rm_src   = false);
  stage_out            (in  url               src,
                        in  url               tgt,
                        in  bool              rm_src   = false);
  //----------------------------------------------------------------
}
```

```
  ////////////////////////////////////////////////////////////////////
  //
  //
  //
  class pool : implements saga::resource::compute
  {
    //-----------------------------------------------------------------
    CONSTRUCTOR          (out pool               rp);
    DESTRUCTOR           (in  pool               rp);

    //-----------------------------------------------------------------
    add                  (in  resource           r);
    add                  (in  string             id);

    remove               (in  resource           r);
    remove               (in  string             id);

    list                 (out array<string>      ids);
    get                  (out array<resource>    rs);

    //-----------------------------------------------------------------
    // set scheduler policy, such as
    // default, round_robin, random, load, my_super_scheduler, ...
    set_job_scheduler    (in  string             s="default");
    //-----------------------------------------------------------------
  }
}
```

2.3 Specification Details

2.3.1 saga::resource::description

```
////////////////////////////////////////////////////////////////////////
//
// The resource description attributes define what slice of
// a resource should be made available at what time.
//
// Allowed values for 'type':
//       'reservation'      : see DRMAAv2.0
//       'pilot_job'        : see TROYv1.0
//       'iaas'             : see OCCIv1.0
//
// Allowed values for 'template'
//        'diane', 'condor' : troy backend
//        'small', 'large'  : EC2 machine template
//        'cluster globus'  : nimbus / FG resource template
//
//  dynamic: can grow and shrink, depending on load etc
//
// Note: DRMAAv2.0 has a notion of 'slots' which map to the
// notion of 'cores' used here.
//
class  description : implements saga::attributes
{
  // FIXME: use SAGA's SIDL attribute notation
  string         type
  string         template = 'default'

  enum           machine_os
  enum           machine_arch
  int            cores
  array<string>  hostnames
  long           memory

  time           start
  time           end
  time           duration

  string         pj_executable
  array<string>  pj_args
  array<string>  pj_env
```

```
    string          pj_queue

    bool            dynamic = 'true';
  };
```

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.xx, 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.