

SAGA Tutorial - Part II

Introduction to the SAGA API

Agenda



- Recap:Grid Application Programming
- established Grid APIs (Globus, gLite, CoG, GAT, ..., OGF)
- SAGA as Unifying Approach
 - Requirements
 - Structure
 - Walkthrough
 - Ongoing Developments
- Discussion

Recap: Grid Applications



Types of Grid Applications

- 1. legacy applications
- 2. legacy distributed applications
- 3. Grid aware applications

Legacy Applications



- no access to application code
- virtualization of heterogenuity
- use cases: remote resource utilization, high throughput
- favourite technique: sandboxing (virtualization)
- no need for Grid APIs (application exists, non-mutable)
- not a topic for this tutorial

Legacy Distributed Applications Legacy Distributed Applications

- aware of distribution (MPI, CORBA, ...)
- not aware of Grid properties (VO)
- usually not very dynamic or adaptive (bootstrapping!)
- use cases: scientific applications, bussiness applications
- favourite technique: emulation (GridMPI, etc.)
- no need for new Grid APIs (APIs non-mutable)
- not a topic for this tutorial

Grid Aware Applications



- aware of distribution, heterogeneity, VOs, elasticity etc.
- often dynamic and adaptive structure
- use cases: collaborative, adaptivite, auto-optimized, scalabile, ...
- favourite technique: depends on structure, middleware, ...
- need for Grid APIs (see Part 1)
- topic for this tutorial :-)

Grid APIs and Frameworks



- Middleware often targets legacy applications (Unicore, Globus, Condor, ...)
- some are distribution aware (MPICH-G, Ninf-G, ...)
- few APIs exist for Grid aware applications
 - GridFTP
 - GRAM
 - gLite
 - CoG
 - GAT
 - Cloud APIs

Grid APIs: Globus (pre-WS)



- low level API for the Globus Grid Middleware
- scope reflects Globus services:
 - GridFTP
 - GRAM
 - MDS
 - Replicas
- some low level API abstractions (xio, gss-assist)
- CoG provides higher level API abstraction for Globus (Java)

GridFTP Example



```
globus_module_activate (GLOBUS_FTP_CLIENT_MODULE);

globus_ftp_client_handleattr_init (&handle_attr);

globus_ftp_client_handle_init (&handle, &handle_attr);

globus_ftp_client_handle_cache_url_state (&handle, server.c_str());
```

GridFTP Example (ii)



```
GridFTP: Get File Size ____
globus_ftp_client_operationattr_init (&attr);
globus ftp client operationattr set mode (&attr, ...);
globus_result_t success = globus_ftp_client_size
                             (&handle,
                             url.c str(),
                             &attr,
                             &size,
                             GLOBUS_NULL, // done_callback,
                             GLOBUS NULL);
if (success != GLOBUS SUCCESS)
```

GridFTP



- API covers full scope of GridFTP protocol
- low level control over connection and operations
- allows syncronous and asyncronous calls (callbacks)

GRAM Example



GRAM



- API provides full scope of GRAM protocol
- low level control over operations
- syncronous and asyncronous calls
- job details encapsulated in job description (RSL)

GRAM Example (RSL)



```
_GRAM: RSL example ____
directory = "/home/user/demo" )
 jobtype = mpi )
 executable = "/home/user/demo/mpi-application" )
 maxWallTime = "10")
count = "8" )
( architecture = "i386" )
&
 directory = "/home/user/demo" )
 jobtype = mpi )
executable = "/home/user/demo/mpi-application" )
 maxWallTime = "10")
count = "16" )
( architecture = "i386" )
( resourceManagerContact = "fs2.das2.nikhef.nl" )
```

GRAM - RSL



- GRAM comes with its own job / resource description language
- most middlewares invent their own languages
- requirements are interpreted in several places (resource broker, queue manager, ...)

gLite Example



```
qLite: Job Submit -
client.Delegate (delegID,
                 "https://cream-ce-01:8443/.../CREAMDelegation",
                 "/tmp/x509up u202");
client.Register ("https://cream-ce-01:8443/.../CREAM",
                 "https://cream-ce-01:8443/.../CREAMDelegation",
                 delegID,
                 JobDescriptionBuffer,
                 "/tmp/x509up u202",
                 uploadURL_and_jobID,
                 0, false);
                ("https://cream-ce-01:8443/.../CREAM",
client.Start
                 uploadURL and jobID[1]);
```

gLite



- moves security details to API level
- in some sense, is a customized globus like environment
- shows its Globus foundations
- faithful to the web service paradigm (app-level WSDL)
 - → API reflects gLite service interface
- JDL vs. RSL

CoG Example



```
- CoG: Job Submit —
String gramContact = "pitcairn.mcs.anl.gov:6722:...";
String rsl = %(executable=...)(...)(...);
GramJob job = null;
try {
   job = new GramJob (rsl);
  Gram.request (gramContact, job);
catch (GramException e) {
```

CoG



- covers same scope as Globus API
- hides complexity and API evolution
- separates functional and non functional API parts

 new versions provide additional functionality (workflow, GUI, ...) and cover non-globus middleware

GAT Example



GAT



- tries to abstract Grid Middleware functionality
- tries to hide middleware details
- implementable on multiple middleware systems
- usability limited by scope of its use cases (historical reasons)

Summary



- diversity of Grid Middleware implies diversity of APIs
- some APIs try to generalize Grid programming concepts
- difficult to keep up with MW development, and to stay simple

OGF



- Open Grid Forum (GF, EGF, GGF) tries to standardize Grid MW
- e.g. single job description language (JSDL)
- focuses on interfaces, but also protocol and architecture

APIs within OGF



- OGF focuses on services
- some effort on higher level APIs
 - Distributed Resource Management Application API (DRMAA)
 - Remote Procedure Calls (GridRPC)
 - Checkpoint and Recovery (GridCPR)
 - Job Submission and Description Language (JSDL)
- numerous service interfaces, often WS-based (WSRF)

OGF: DRMAA



- implementable on all major resource management services
- simple means to define jobs, and to submit them
- basic job management features (status, kill)
- job templates for bulk job management

DRMAA Example



```
- DRMAA Job Submit —
drmaa_job_template_t * job_template;
if (! (job template = create job template (exe, 5, 0))
  fprintf (stderr, "create_job_template failed\n");
  return 1;
while ( ( drmaa_errno = drmaa_run_job (job_id,
                                       sizeof (jobid)-1,
                                       job_template,
                                       diagnosis,
                                       sizeof (diagnosis)-1)
        ) == DRMAA ERRNO DRM COMMUNICATION FAILURE )
  fprintf(stderr, "drmaa run job failed: %s\n", diagnosis);
  sleep (1);
```

OGF: GridRPC



- 'standardizes' the three existing RPC implementations for Grids
- example of 'gridified API'
- simple: get function handle, call function
- explicit support for async rpc calls

OGF: GridRPC



```
_ GridRPC: Matrix Multiplication —
double A[N*N], B[N*N], C[N*N];
initMatA (N, A);
initMatB (N, B);
grpc initialize (arqv[1]);
grpc_function_handle_t handle;
grpc function handle default (&handle, "mat mult");
if ( grpc_call (&handle, N, A, B, C) != GRPC NO ERROR)
 exit (1);
grpc function handle destruct (&handle);
grpc_finalize ();
```

OGF: GridCPR



- Grids seem to favour application level checkpointing
- GridCPR allows to manage checkpoints
- defines an architecture, service interfaces, and, aehem, no API

OGF: JSDL



- extensible XML based language for describing job requirements
- does not cover resource description (on purpose)
- does not cover workflows, or job dependencies etc (on purpose)
- JSDL is extensible (ParameterSweep, SPMD)

OGF: JSDL



```
JSDL: Simple Job ____
 <jsdl:JobDefinition>
   <JobDescription>
      <Application>
         <jsdl-posix:POSIXApplication>
            <Executable>/bin/date</Executable>
         </jsdl-posix:POSIXApplication>
      </Application>
      <Resources ...>
         <OperatingSystem>
            <OperatingSystemType>
               <OperatingSystemName>LINUX</OperatingSystemName>
            </OperatingSystemType>
         </OperatingSystem>
      </Resources>
   </JobDescription>
<jsdl:JobDefinition>
```

OGF: JSDL



- XML: embeddable into WSRF (WS-Agreement etc.)
- XML, but relatively flat
- maps well to existing JDLs, but is 'more complete'
- extensible (resource description, job dependencies, workflow)
- top down approach!

OGF: Summary



- some APIs exist in OGF, and are successfull
- OGF APIs do not cover the complete OGF scope
- the various API standards are disjunct
- WSDL as service interface specification cannot replace an application level API (wrong level of abstraction)
- SAGA tries to address these issues

OGF: top-down vs. bottom-up



- bottom-up often agrees on (semantic) LCD + backend specific extensions
- top-down usually focuses on semantics of application requirements
- bottom-up tends to be more powerful
- top-down tends to be simplier and concise
- we very much prefer top-down!

SAGA



SAGA

Simple API for Grid Applications

SAGA overview



- SAGA API structure and scope
- walkthrough
- planned extensions
- some implementation details

SAGA as a OGF API



- SAGA: Simple API for Grid Applications
- OGF approach to a uniform API layer (facade)
- provides application level abstractions for Grid operations
- major influences: DRMAA, GridRPC, OREP, JSDL, ...

SAGA Design Principles



- governing principle: **80:20 rule** simplicity versus control!
- top-down approach: use case driven!
- defines application level abstractions
- extensible: stable look & feel + API packages
- major influences: POSIX, GAT, CoG, LSF, Globus, . . .
- API Specification is Language Independent (IDL)
 Renderings exist in C++, Python, Java
 Examples here are in C++



```
SAGA: File Management -
saga::directory dir ("any://remote.host.net//data/");
if ( dir.exists ("a") && ! dir.is_dir ("a") )
 dir.copy ("a", "b", Overwrite);
list <string> names = dir.find ("*-{123}.txt");
saga::directory tmp = dir.open_dir ("tmp/", Create);
saga::file file = dir.open ("tmp/data.txt");
```



- API is clearly POSIX (libc + shell) inspired
- where is my security??
- what is 'any: / /' ???



```
SAGA: Job Submission -
saga::job_description jd; // details left out
saga::job service js ("any://remote.host.net");
          j = js.create_job (jd);
saga::job
j.run ();
cout << "Job State: " << j.get_state () << endl;</pre>
j.wait ();
cout << "Retval " << j.get_attribute ("ExitCode") << endl;</pre>
```



```
SAGA: Job Submission —
saga::job service js ("any://remote.host.net");
saqa::job
                       j = js.run_job ("touch /tmp/touch.me");
cout << "Job State: " << j.get_state () << endl;</pre>
j.wait ();
cout << "Retval " << j.get_attribute ("ExitCode") << endl;</pre>
```



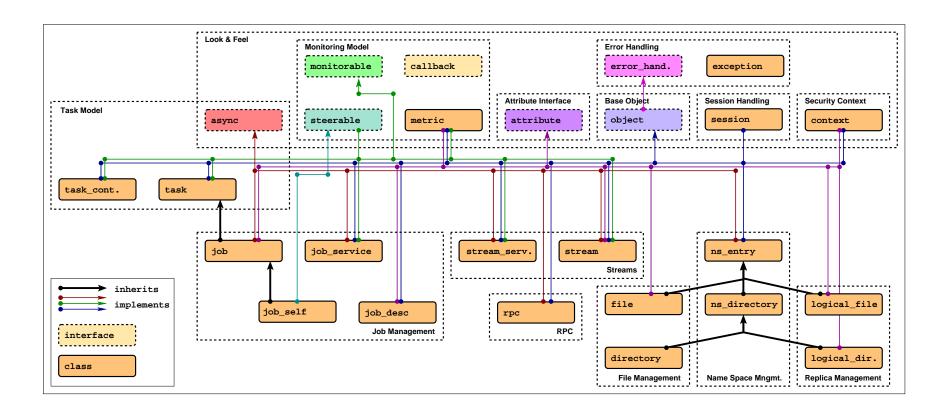
- stateful objects!
- yet another job description langauge? |:-(|
- many hidden/default parameters (to keep call signatures small)
- 'any: //' again!
- TIMTOWTDI (there is more than one way to do it)

SAGA Intro: 10.000 feet



- object oriented: inheritance, interfaces very moderate use of templates though!
- functional and non-functional elements strictly separated
 - non-functional API: look & feel- orthogonal to functional API often not mappable to remote operations
 - functional API: API 'Packages' extensible typically mappable to remote operations
- few inter-package dependencies allows for partial implementations

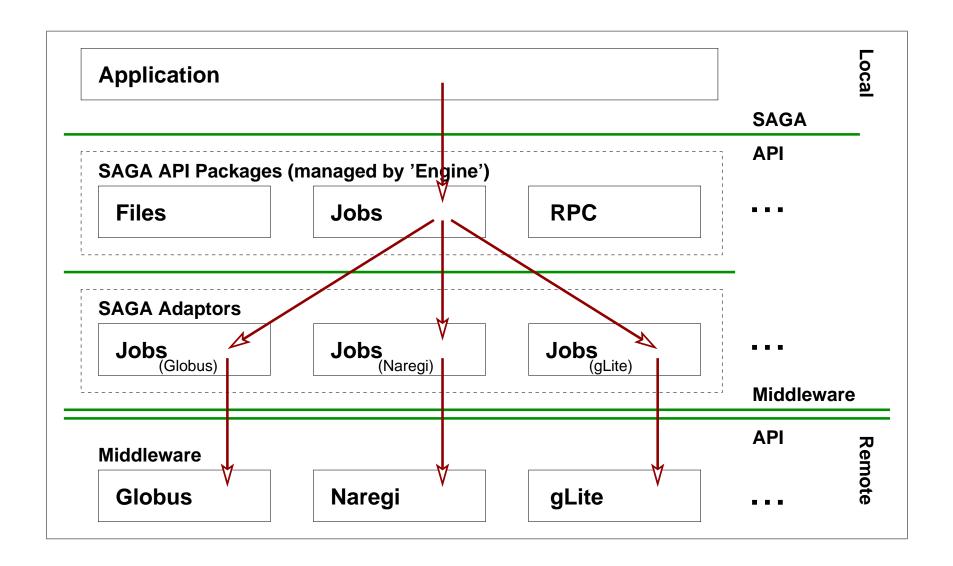




SAGA API Structure: look & feel (top) + API packages (bottom)

Implementation







Look & Feel					
1					:
! !					:
<u> </u>					:
<u> </u>					
I					· · · · · · · · · · · · · · · · · · ·
					· · · · · · · · · · · · · · · · · · ·
					;
					:
					:
I I					:
! !					:
,				,	_
	1:		<u> </u>	!	
			;		
			i		
			;		
! !	1 1			;	
! !				:	·
! !	1	,		:	
! !	1			:	; ;
<u> </u>	:		:	:	
1	1		1		
	:		; ;		
	1		: :		: :
	-		1 :		1 1
				:	!
				:	: :
				:	
			;	:	



	Look & Feel
	,
	Base Object
	object
2-1-1	
inherits	
implements	
interface	
class	

SAGA Look & Feel:

saga::object allows for object uuids, clone() etc.

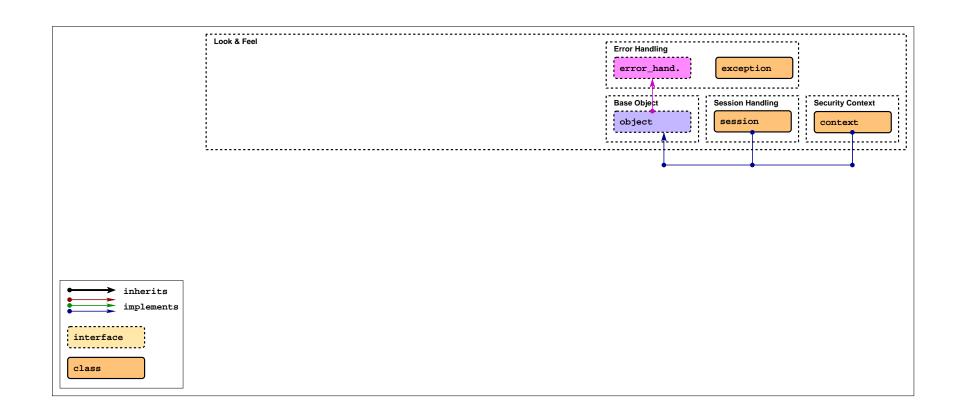


	Look & Feel	٠		,	
	•	Error	Handling	•	
	<u> </u>	· ·			
	<u> </u>	er	ror_hand.	exception	
	<u> </u>	; '			
	•	1			
	<u> </u>	,			
		Base	Object		
	<u> </u>		•		
	<u>.</u>	; ; ob;	ject		
	į.				
	•	1	i		
	1	 			
→ inherits					
					
implement	:s				
,					
nterface					
lass					

SAGA Look & Feel:

errors are based on exceptions or error codes.

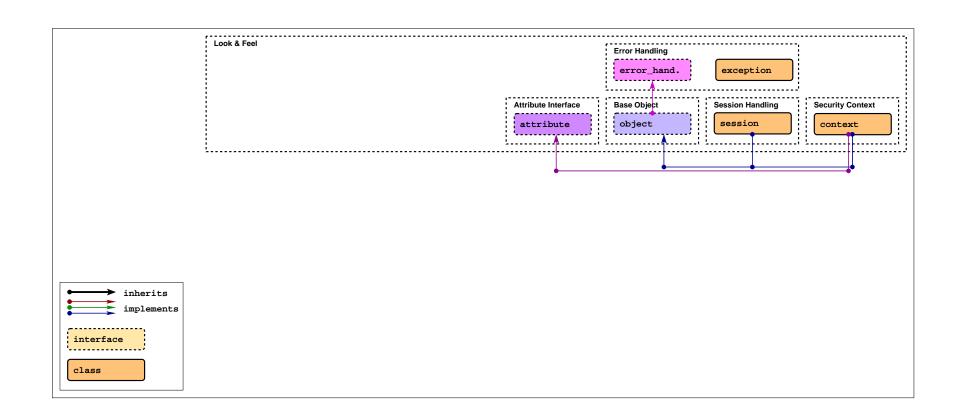




SAGA Look & Feel:

session and credential management is hidden.

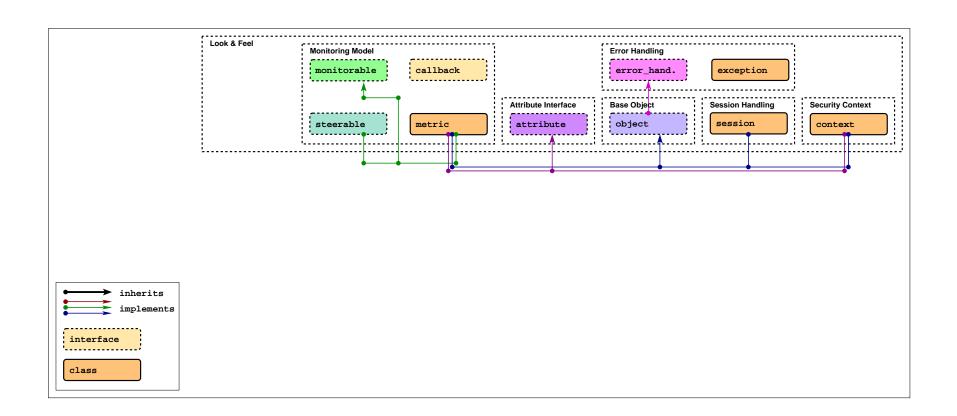




SAGA Look & Feel:

Attribute interface for meta data.

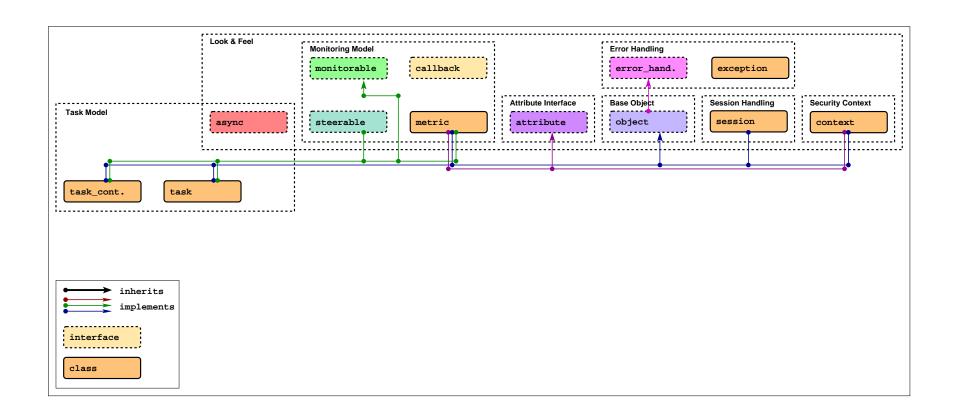




SAGA Look & Feel:

Monitoring includes asynchronous notifications.

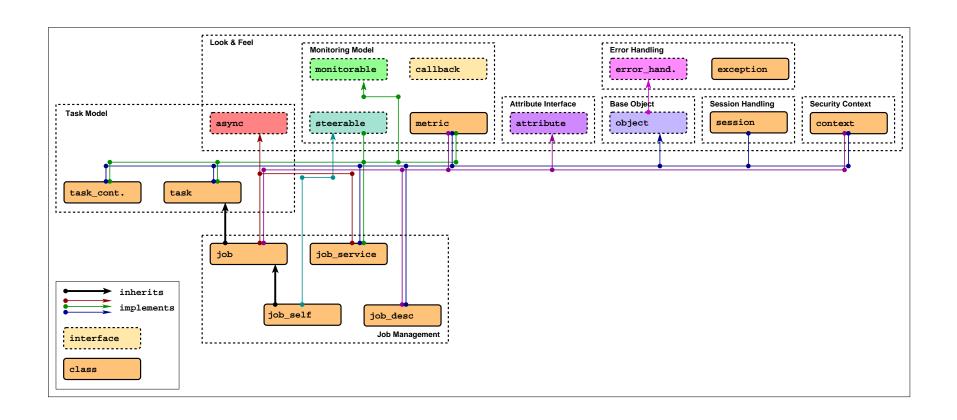




SAGA Look & Feel:

the task model adds asynchronous operations.

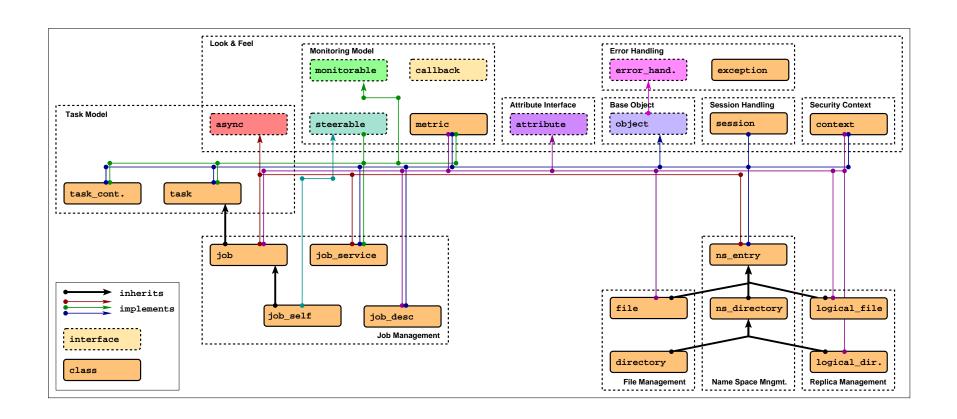




SAGA API Package 'job':

create and manage remote processes.

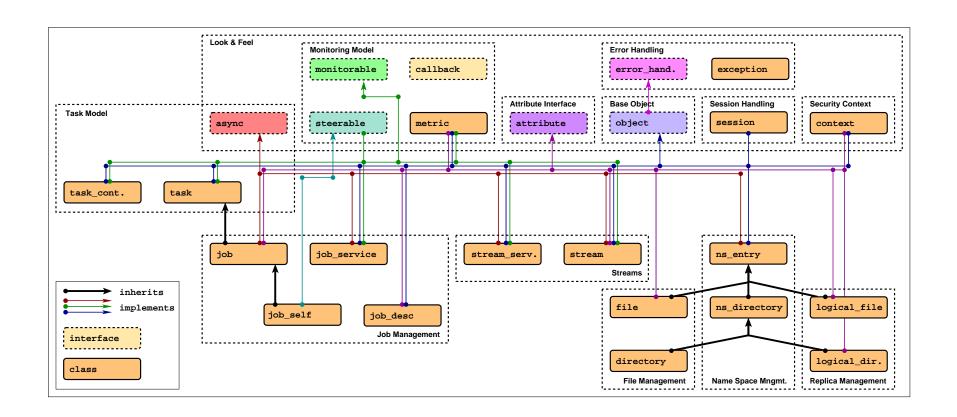




SAGA API Package 'name_spaces':

manage files, replicas, etc.

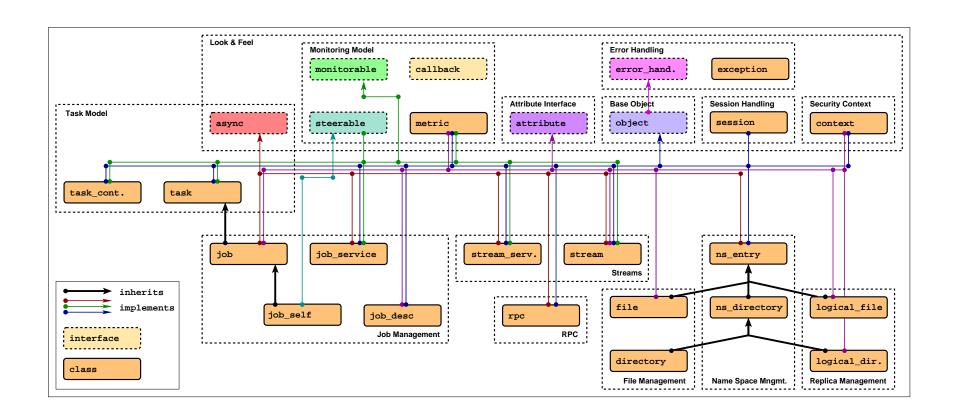




SAGA API Package 'stream':

SAGA rendering of BSD streams.





SAGA API Package 'rpc':

remote procedure calls.

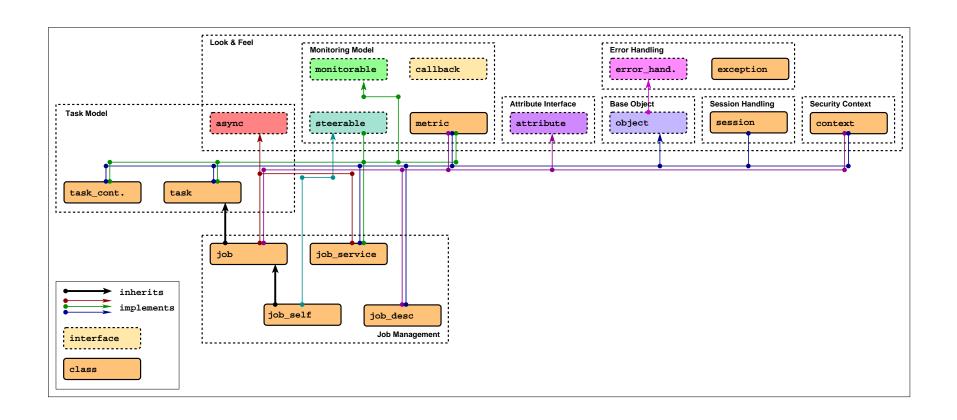
SAGA: Jobs



Functional API Packages

SAGA: Jobs





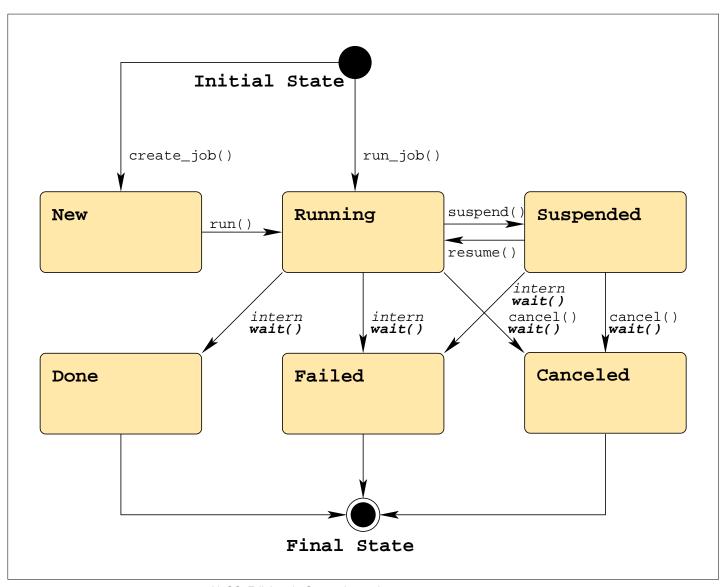
SAGA: Jobs - Overview



- job_service uses job_description to create job instances
- job_description attributes are based on JSDL
- state model is based on / synced with BES
- job_self represents the SAGA application
- job submission and management, but no resource discovery, job dependencies, or workflows

SAGA: Job States





SAGA Examples: Jobs



```
job submission —
saga::job service js ("gram://headnode.gram.net");
saqa::job
                       j = js.run job ("/bin/sleep 10",
                                        "clusternode-2.gram.net");
cout << "Job State: " << j.get_state () << endl;</pre>
j.wait ();
cout << "Retval " << j.get_attribute ("ExitCode") << endl;</pre>
```

SAGA Examples: Jobs



```
_ job submission _
saga::job_description jd;
saga::job_service js ("gram://remote.host.net");
saqa::job
          j = js.create_job (jd);
j.run ();
cout << "Job State: " << j.get_state () << endl;</pre>
j.wait ();
cout << "Retval " << j.get_attribute ("ExitCode") << endl;</pre>
```

SAGA Examples: Jobs



```
jobs (cont.) -
j.run ();
j.wait ();
j.cancel ();
j.suspend ();
j.resume ();
j.signal (SIGUSR1);
j.checkpoint ();
j.migrate (jd);
```

SAGA Examples: Job Descr.



```
job description - JSDL based

saga::job_description jd;

jd.set_attribute ("Executable", "/bin/tail");
jd.set_attribute ("WorkingDirectory", "data/");
jd.set_attribute ("Cleanup", "False");

jd.set_vector_attribute ("Arguments", ["-f", "my_log"]);
jd.set_vector_attribute ("Environment", ["TMPDIR=/tmp/"]);
jd.set_vector_attribute ("FileTransfer", ["my_log >> all_logs"]);
```

SAGA Job Description



SAGA JD attributes:

Executable
CandidateHosts
NumberOfProcesses
WorkingDirectory
Input
JobStartTime
TotalPhysicalMemory
Queue

FileTransfer

Arguments
SPMDVariation
ProcessesPerHost
Interactive
Output
WallTimeLimit
CPUArchitecture
JobProject

Environment
TotalCPUCount
ThreadsPerProcess
Cleanup
Error
TotalCPUTime
OperatingSystemType
JobContact

SAGA Job Description



- leaning heavily on JSDL, but flat
- borrowing from DRMAA
- mixes hardware, software and scheduling attributes!
- cannot be extended
- no support for 'native' job descriptions (RSL, JDL, ...)
- only 'Executable' is required
- backend MAY ignore unsupported keys!

cd /tmp/data && rm -rf *

SAGA Example: job service



```
___ job service ___
saga::job::service js ("gram://remote.host.net/");
vector<string> ids = js.list (); // list known jobs
while ( ids.size () )
  string id = ids.pop_back ();
  saga::job j = js.get_job (id); // reconnect to job
  cout << id << " : " << j.get_state () << endl;</pre>
```

SAGA Job Service



- represents a specific job submission endpoint
- job states are maintained on that endpoint (usually)
- full reconnect may not be possible (I/O streaming)
- lifetime of state up to backend
- reconnected jobs may have different job description (lossy translation)

SAGA Examples: Job 'Self'



```
saga::job self = js.get_self ();

self.signal    (SIGUSR1);
self.checkpoint ();
self.migrate    (jd);

self.wait    ();    // blocks forever :-P
self.cancel ();
```

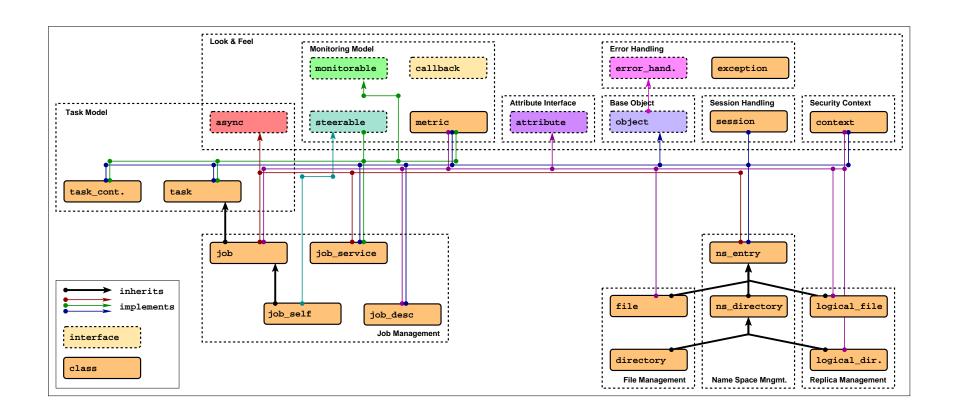
SAGA Job Self



- represents the calling application instance
- self.signal (SIGUSR1);
 is different from
 ::kill (::getpid (), SIGUSR1);
 as it goes over the job manager (accounting, logging, cleanup...)
- 'job::self' is the only SAGA object to which metrics can be added (details later...)

SAGA: Name Spaces etc.





SAGA: Name Spaces



- interfaces for managing entities in name spaces
- files, replicas, information, resources, steering parameter, checkpoints, . . .
- manages hierarchy (mkdir, cd, ls, ...)
- manages NS entries as opaque (copy, move, delete, ...)

SAGA: Files



- implements name space interface, and adds access to content of NS entries (files)
- Posix oriented: read, write seek
- Grid optimizations: scattered I/O, pattern based I/O, extended I/O

SAGA: Replicas



- implements name space interface, and adds access to properties of NS entries (logical files / replicas)
- O/REP oriented: list, add, remove replicas; manage meta data
- Grid optimizations are hidden (replica placement strategies, consistency and version management, ...)

SAGA Examples: NameSpaces Open Gr



```
name space management .
saga::ns dir dir ("gridftp://remote.host.net//data/");
if ( dir.is entry ("a") && ! dir.is dir ("a") )
 dir.copy ("a", "../b");
 dir.link ("../b", "a", Overwrite);
list <string> names = dir.find ("*-\{123\}.text.");
saga::ns dir tmp = dir.open dir ("tmp/", DeReference);
saga::ns entry entry = dir.open ("tmp/data.txt");
entry.copy ("data.bak", Overwrite);
```



```
file access _
saga::file f ("gridftp://remote.host.net/data/data.bin");
char buf[100];
if (f.get_size() >= 223)
  int pos = f.seek (123, Current);
  int len = f.read (100, buf);
```



```
\mathsf{L} file access - scattered I/O \mathsf{L}
saga::file f ("gridftp://remote.host.net/data/data.bin");
saga::ivec ivecs[100];
ivecs[0].buffer = NULL;
ivecs[0].offset = 1;
ivecs[0].leng_in = 10;
if (f.get_size() >= 223)
  f.read v (ivecs);
```



```
file access - pattern based I/O -
saga::file f ("gridftp://remote.host.net/data/data.bin");
char buf[100];
string pattern ("(0,17,36,6,(0,0,2,6))");
if (f.get_size() >= 223)
  int len = f.read p (pattern, buf);
```



```
file access - extended I/O -
saga::file f ("gridftp://remote.host.net/data/data.bin");
char buf[100];
string mode ("JPEG-crop");
string spec ("coord=0,0,10,10");
if (f.get_size() >= 223)
  int len = f.read e (mode, spec, buf);
```

SAGA Examples: Replicas



```
replica management
saga::logical_directory dir ("raptor://remote.host.net/data/");
if (dir.is entry ("a") | dir.is link ("a") )
 dir.copy ("a", "../b");
 dir.link ("../b", "a");
saga::logical_file file = dir.open ("tmp/data.txt");
list <string> locations = file.list_locations ();
file.replicate ("gridftp://other.host.net/tmp/a.dat");
```

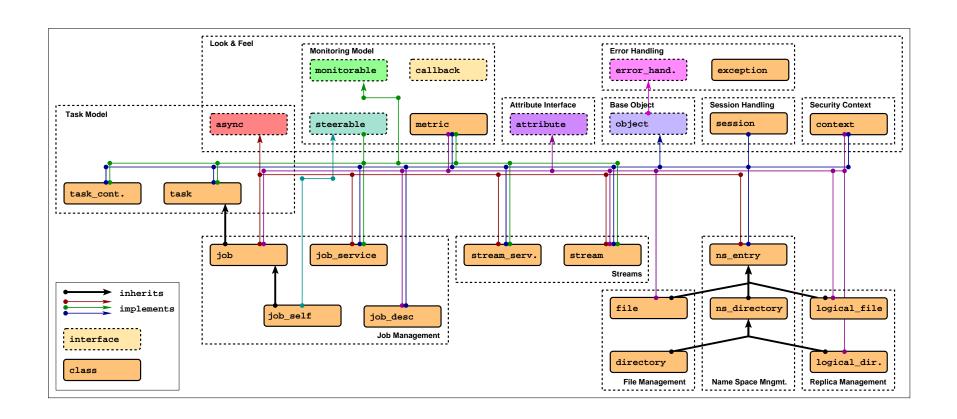
SAGA Examples: Replicas



```
.replica meta data —
saga::logical_directory dir ("raptor://remote.host.net/data/");
list <string> files = dir.find ("*", "type=jpg");
while ( file.size () )
  saga::logical_file lf (file.pop_front ());
  lf.replicate ("file://localhost/adta/all_jpg", Overwrite);
```

SAGA: Streams





SAGA: Streams



- simple and BSD socket oriented
- not supposed to replace MPI etc, but allows for simple application level communication
- potentially slow, but can be implemented efficiently

SAGA Examples: Streams

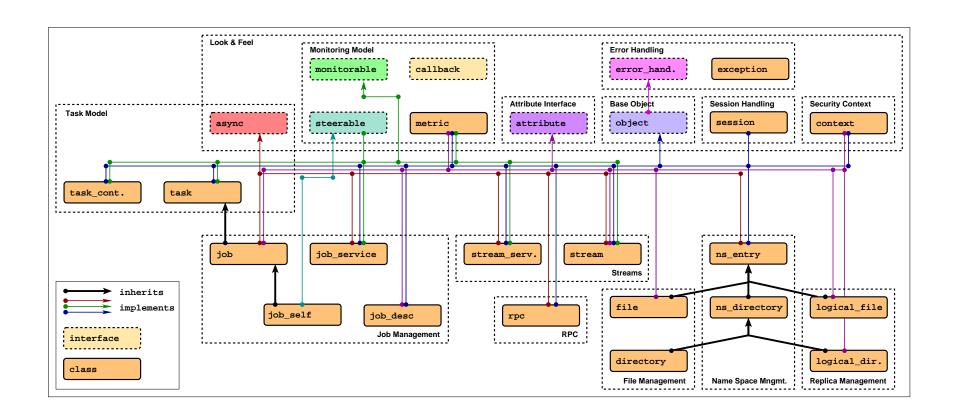


```
stream server
saga::stream_service ss ("tcp://localhost:1234");
saga::stream_client sc = ss.serve ();
sc.write ("Hello client", 13);
```

```
char buf [13];
saga::stream_client sc ("tcp://remote.host.net:1234");
sc.connect ();
sc.read (buf, 13);
cout << buf << endl;
```

SAGA: RPC





SAGA: RPC



- maps GridRPC standard into the SAGA look & feel
- parameters are stack of structures (similar to scattered I/O)
- future revision will work on optimized data handling

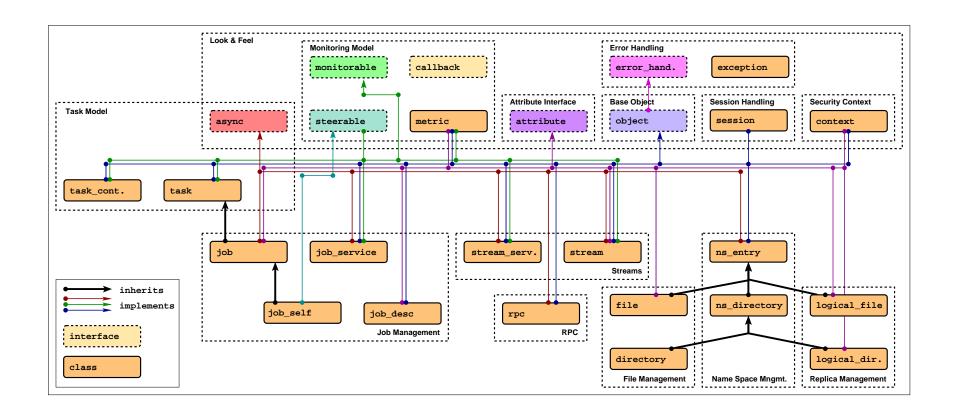
SAGA Examples: RPC



```
remote procedure call
saga::rpc rpc ("ninfg://remote.host.net:1234/random");
list <saga::rpc::parameter> params;
params.push_back (new saga::rpc::parameter (Out, 10));
rpc.call (params);
cout << "found random number: " << atoi (param.buffer) << endl;
delete (params.pop_front ());</pre>
```

SAGA: Session and Context





SAGA: Session Management



- by default hidden (default session is used)
- session is identified by lifetime of security credentials and by objects in this session (jobs etc.)
- session is used on object creation (optional)
- saga::context is used to attach security tokens to a session
- the default session has default contexts



_ default sessions saga::ns dir dir ("gridftp://remote.host.net//data/"); if (dir.is entry ("a") && ! dir.is dir ("a")) dir.copy ("a", "../b"); dir.link ("../b", "a", Overwrite); list <string> names = dir.find (" $*-\{123\}$.text."); saga::ns dir tmp = dir.open dir ("tmp/", DeReference); saga::ns entry entry = dir.open ("tmp/data.txt"); entry.copy ("data.bak", Overwrite);



```
context management.
saga::context c1 (saga::context::Globus);
saga::context c2 (saga::context::Globus);
c2.set attribute ("UsrProxy", "/tmp/x509up u123.special");
saga::session s;
s.add context (c1);
s.add_context (c2);
saga::ns_dir dir (s, "any://remote.host.net/data/");
```



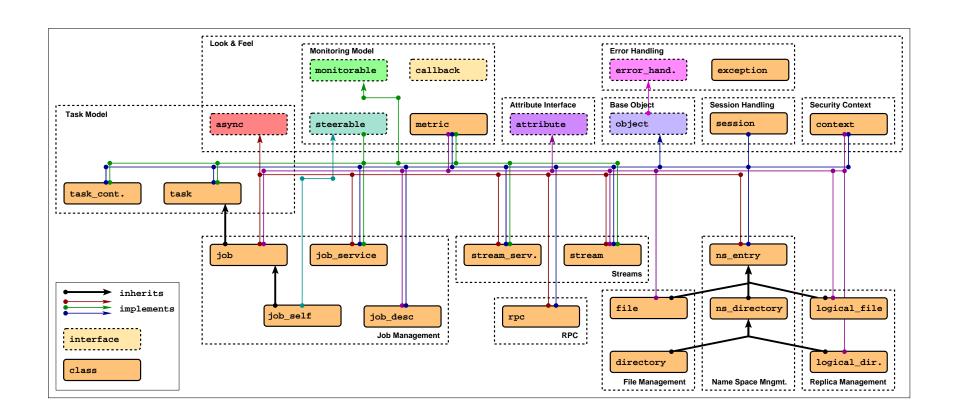
```
session inheritance
saga::dir dir (s, "gridftp://remote.host.net/data/");
saga::file file = dir.open ("data.bin");
s.remove_context (c1);
s.remove_context (c2);
file.copy ("data.bin.bak"); // works - state is sticky!
```



```
—— authorization —
// server side code
saga::stream_service ss ("tcp://localhost:1234");
saga::stream client sc = ss.serve ();
saga::context c = sc.get context ();
if ( c.get type == Globus &&
     c.attribute_equals ("RemoteID", "O=MyCA, O=MyOrg, CN=Joe") )
  sc.write ("welcome!", 9);
else
  sc.write ("bugger off!", 12);
  sc.close ();
```

SAGA: Monitoring





SAGA: Monitoring



- monitoring of Grid entities (jobs, files, ...)
- monitoring of interactions (task state, notification, ...)
- monitorables have metrics
- metrics can be pulled, or subscribed to (callbacks)
- some metrics can be written (basic steering)

SAGA Examples: Monitoring Open Copen Copen



```
\mathsf{	extstyle 	e
saga::job job = js.create_job (jd);
 job.run ();
saga::metric m = job.get metric ("MemoryUsage");
while (1)
                        cout << "Memory Usage: " << m.get_value () << endl;</pre>
                        sleep (1);
```

SAGA Examples: Monitoring



```
callbacks.
class my_cb : public saga::callback
 public:
    bool cb (saga::monitorable obj,
             saga::metric
                                m,
             saga::context c)
      cout << "Memory Usage: " << m.get_value () << endl;</pre>
      return (true);
my cb cb;
saga::job job = js.create_job (jd);
job.run ();
saga::metric m = job.get_metric ("MemoryUsage");
 m.add callback ("MemoryUsage", cb);
```

SAGA Examples: Monitoring



```
callbacks.
class my_cb : public saga::callback
 public:
    bool cb (saga::monitorable obj,
             saga::metric
                                m,
             saga::context c)
      cout << "Memory Usage: " << m.get_value () << endl;</pre>
      return (true);
my cb cb;
saga::job job = js.create_job (jd);
job.run ();
job.add_callback ("MemoryUsage", cb);
```

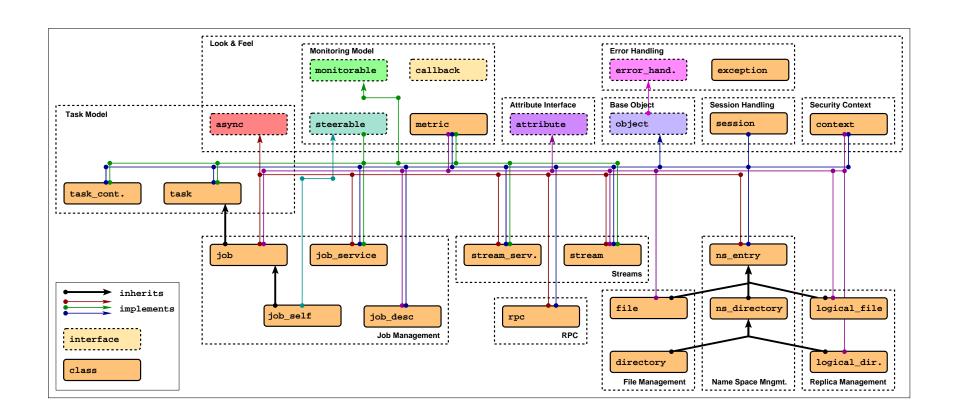
SAGA Examples: Monitoring



```
— callbacks (cont.) —
class my_cb : public saga::callback
 public:
    bool cb (saga::monitorable obj,
             saga::metric
                               m,
             saga::context c)
      cout << m.get_name () << " : " << m.get_value () << endl;</pre>
      return (true);
list <string> metrics = job.list metrics ();
while ( metrics.size () )
  job.add_callback (metrics.pop_front (), cb);
```

SAGA: Tasks





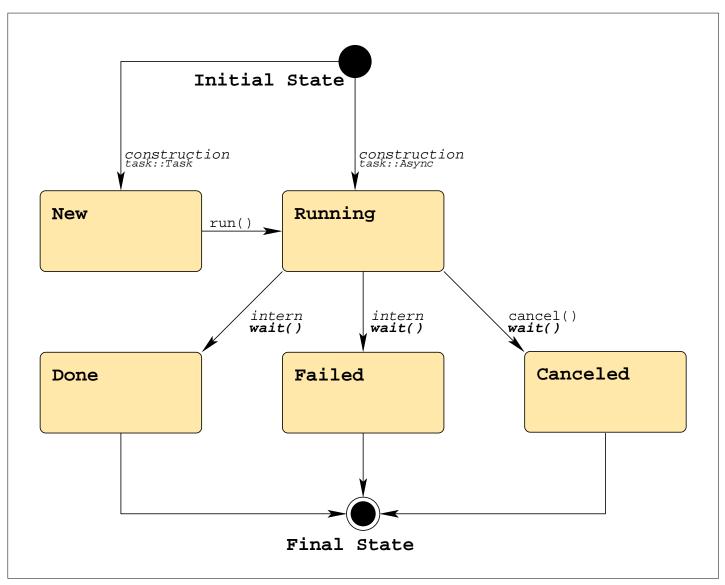
SAGA: Tasks



- asyncronous operations are a MUST in distributed systems, and Grids
- saga::task represents an syncronous operation
 (e.g. file.copy ())
- saga::task_container manages multiple tasks
- tasks are stateful (similar to jobs)

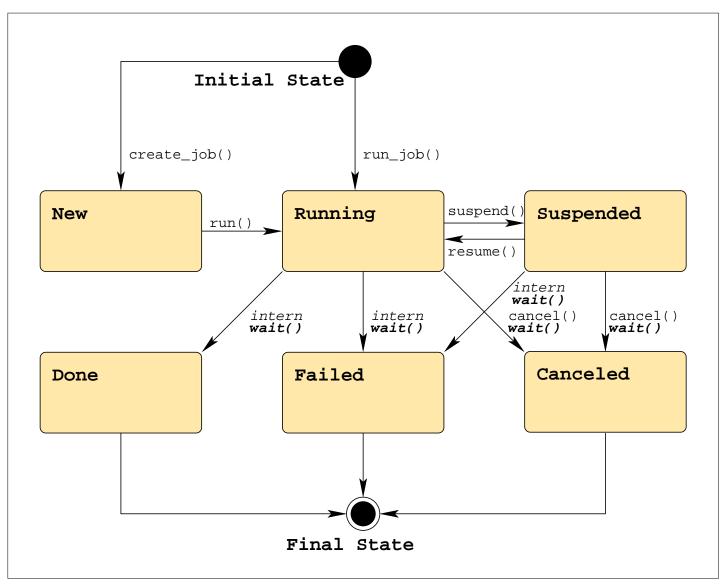
SAGA: Task States





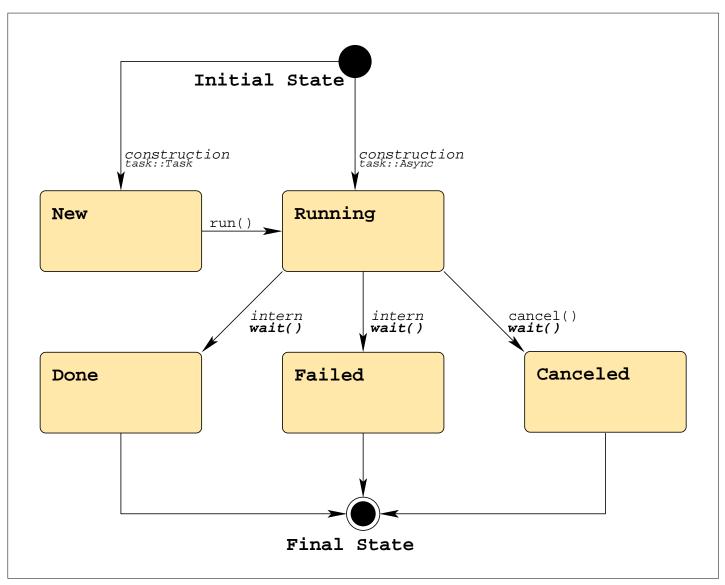
SAGA: Job States





SAGA: Task States





SAGA: Tasks



- different versions for each method call: sync, async, task
- signature basically the same
- differ in state of task representing that method

SAGA Examples: Tasks



```
- tasks (i) ___
saga::file file ("gsiftp://remote.host.net/data/data.bin");
// normal, synchronous
file.copy ("data.bak");
// async versions
saga::task t1 = file.copy <saga::task::Sync> ("data.bak.1");
saga::task t2 = file.copy <saga::task::Async> ("data.bak.2");
saga::task t3 = file.copy <saga::task::Task> ("data.bak.3");
// t1: Done
// t2: Running
// t3: New
```

SAGA Examples: Tasks



```
tasks (ii)

t3.run ();

cout << t3.get_state () << endl; // Running

t2.wait ();

t3.wait ();

// t1, t2, t3: Done (or Failed...)
```

SAGA Examples: Tasks



```
tasks container -
saga::task_container tc;
tc.add (t1);
tc.add (t2);
tc.add (t3);
tc.run ();
saga::task done_task = tc.wait (Any);
tc.wait (All);
```

SAGA Examples: Tasks



```
tasks jobs and notification =
saga::task task = file.copy <saga::task::Asyn> ("b");
saga::job job = js.run job ("remote.host.net", "/bin/date");
task.add callback ("State", my cb);
job.add callback ("State", my cb);
saga::task container tc;
tc.add (task);
tc.add (job);
tc.wait ();
```

SAGA planned extensions



- message based communication
- information service (Advert Service)
- checkpoint & recovery (GridCPR)

SAGA v2: Messages



```
Messaging server

saga::sender snd ("tcp://localhost:1234");

saga::msg msg;

msg.set_size (100); // arbitrary size!

msg.set_data ("abcd...");

sc.send (msg);
```

```
char buf [13];
saga::receiver rec ("tcp://remote.host.net:1234");

// int size = rec.test ();
saga::msg = rec.receive (); // internal buffer allocation
```

SAGA v2: Messages



- messages are received intact or not at all
- implies protocol, but is silent about interop
- async zero copy implementation is possible

SAGA v2: Adverts



- persistent storage of application level information
- semantics of information undefined (app!)
- possibly allows storage of serialized SAGA objects (object persistency)

SAGA v2: Adverts



```
Adverts -
saga::advert_directory adir ("any//remote.host.net/data/");
list <string> adverts = adir.find ("*", "type=jpg");
while ( adverts.size () )
  saga::advert ad (averts.pop_front ());
 ad.get_attribute ("description");
```

SAGA v2: Adverts



```
saga::file f (url);
saga::advert ad ("any//remote.host.net/streams/", Create);
ad.attach ("my_file", f);

saga::advert ad ("any//remote.host.net/streams/");
saga::file f = ad.get_attachement ("my_file");
```

SAGA v2: CPR



- no examples yet, API in flux (serice spec in flux)
- allows to manage (find, move, stage, archive) checkpoints
- allows to trigger checkpointing of jobs
- probably name space based, with notification on CP creation

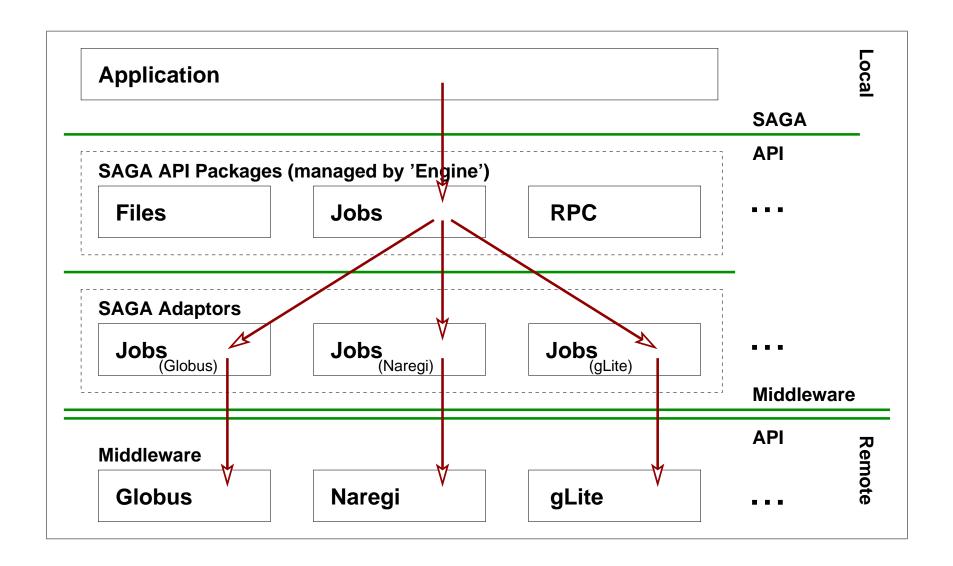


Questions about API?

Comments?

Implementation





Implementation Status



- C++
 - complete implementation by CCT/VU
 - in sync with spec, work in progress
 - other language bindings planned on top (C, Python, Perl, .Net)
- Java (out of sync with the spec)
 - partial implementation (jobs, files) by DEISA/EPCC
 - complete implementation by OMII-UK
 - possible complete implementation at VU
- MiddleWare Bindings
 - DEISA/EPCC binds to DEISA
 - OMII-UK binds to OMII stack, but is flexible
 - C++ adaptor based GT4 and XtreemOS planned



Contact

http://forge.ggf.org/sf/projects/saga-core-wg

---- wiki, CVS details



Questions?