CENTER FOR COMPUTATION
& TECHNOLOGY

SAGA

A Simple API for Grid Applications

Introduction to the SAGA API

omii-uk
www.omii.ac.uk

Agenda

- background recap
- API structure and scope
- API walkthrough
- implementation details
- API extensions

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
  - DRMAA
  - gLite
  - CoG
  - GAT
  - Cloud APIs

Grid APIs and Frameworks

- 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**

Open Grid Forum (OGF)

- The Open Grid Forum (aka GF, EGF, GGF) standardizes distributed computing infrastructures/MW
- e.g. standardized job description language (JSDL)
- focuses on interfaces, but also protocols, architecture, APIs

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 ser vice interfaces, often WS-based (WSRF)

OGF: DRMAA

- implementable on all major resource management services
- simple means to define and submit jobs
- basic job management features (status, kill)
- job templates for bulk job management
- DRMAA.v2 is expected by end of 2010 – OO, extended, SAGA aligned

DRMAA Example

```
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
- gridrpc.v2 adds support for remote data handles

OGF: GridRPC

```
double A[N*N], B[N*N], C[N*N];

initMatA (N, A);
initMatB (N, B);

grpc_initialize (argv[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 favor application level checkpointing
- GridCPR allows to manage checkpoints
- defines an architecture, service interfaces, and scope of client API
- a SAGA extensions exists with a matching client 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, ...)

SAGA

A Simple API for Grid Applications

OGF: JSDL

```
<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 API specs exist in OGF, and are successful
- OGF APIs do not cover the complete OGF scope
- the various API standards are disjoint
- 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 simpler and more concise

- ***we very much prefer top-down!***

SAGA

Simple API for Grid Applications

SAGA
A Simple API for Grid Applications

SAGA Design Principles

- SAGA: Simple API for Grid Applications
  - OGF approach to a uniform API layer (facade)
- governing principle: 80:20 rule
  - simplicity versus control!
- top-down approach
  - use case driven!
  - defines application level abstractions
- extensible
  - stable look & feel
  - API packages
- API Specification is language independent (IDL)
  - Renderings exist in C++, Python, Java
  - Examples here are in C++

SAGA

A Simple API for Grid Applications

SAGA Intro: Example 1

```
// SAGA: File Management example

saga::filesystem::directory dir ("any://remote.host.net//data/");

if ( dir.exists ("a") && ! dir.is_dir ("a") )
{
  dir.copy ("a", "b", Overwrite);
}

list <saga::url> names = dir.find ("*-{123}.txt");

saga::filesystem::directory tmp  = dir.open_dir ("tmp/", Create);
saga::filesystem::file      file = dir.open     ("tmp/data.txt");
```

SAGA Intro: Example 1

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

SAGA Intro: Example 2

```
// SAGA: Job Submission example

saga::job::description jd;
// details left out

saga::job::service js ("any://remote.host.net/");
saga::job::job     j = js.create_job (jd);

j.run ();

cout << "Job State: " << j.get_state () << endl;

j.wait ();

cout << "Retval " << j.get_attribute ("ExitCode") << endl;
```

SAGA

A Simple API for Grid Applications

SAGA Intro: Example 2'

```
// SAGA: Job Submission example



saga::job::service js ("any://remote.host.net");
saga::job::job     j = js.run_job ("touch /tmp/touch.me");



cout << "Job State: " << j.get_state () << endl;

j.wait ();

cout << "Retval " << j.get_attribute ("ExitCode") << endl;
```

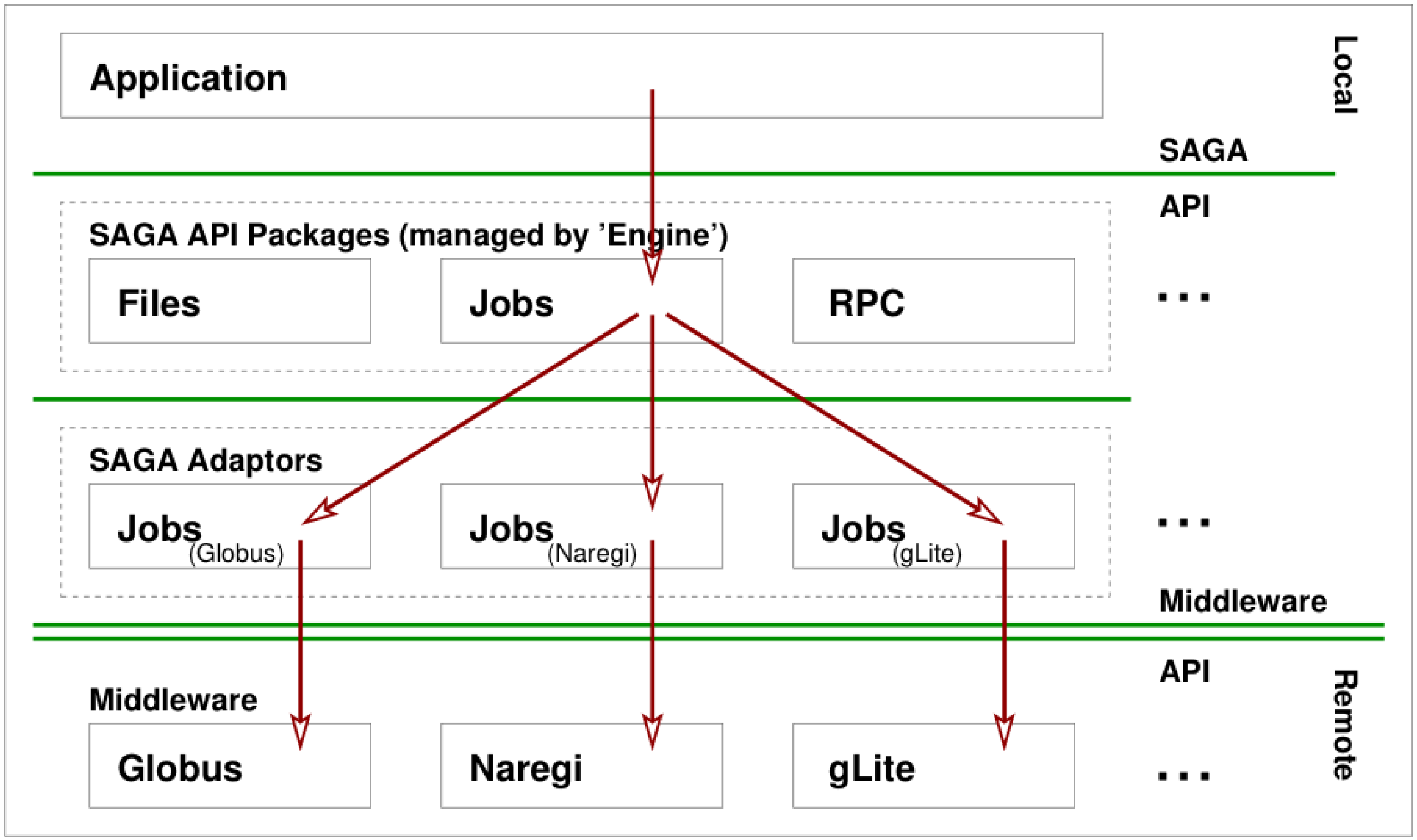
SAGA Intro: Example 2

- stateful objects!
- yet another job description language? :-(
- many hidden/default parameters
  - keeps 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
    - typically not mappable to remote operations
  - functional API:
    - API 'Packages' extensible
    - typically mappable to remote operations
- few inter-package dependencies - allows for partial implementations

Implementation

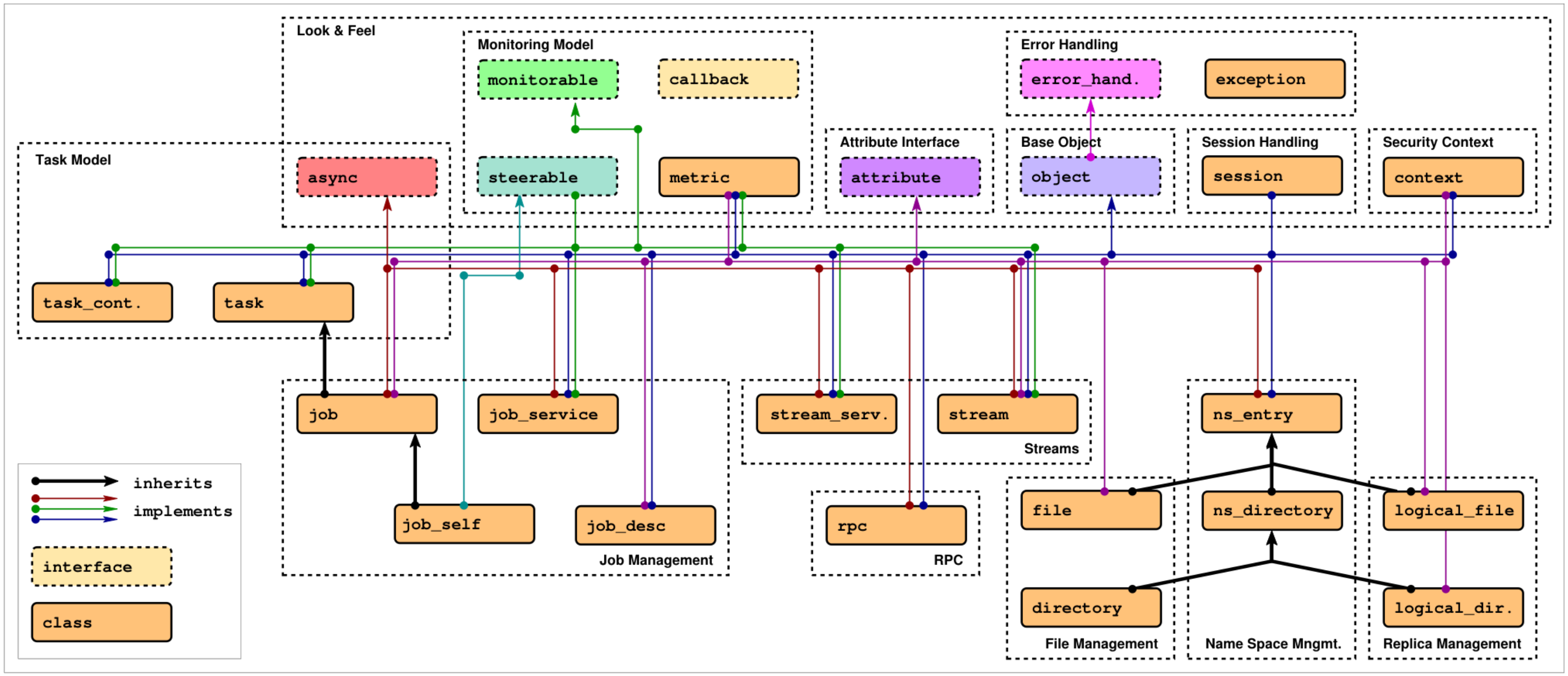


SAGA Class Hierarchy

SAGA

A Simple API for Grid Applications

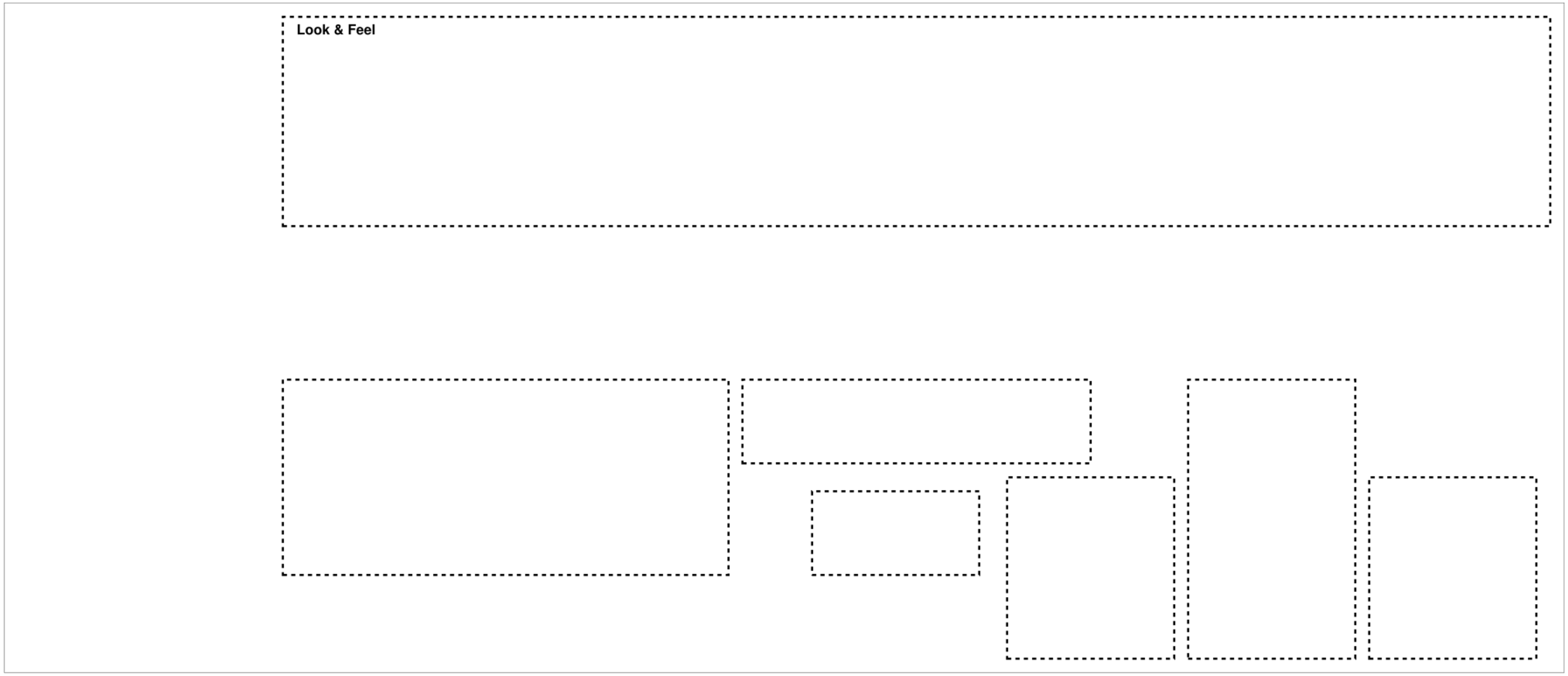
SAGA: Class hierarchy



SAGA

A Simple API for Grid Applications

SAGA Class hierarchy



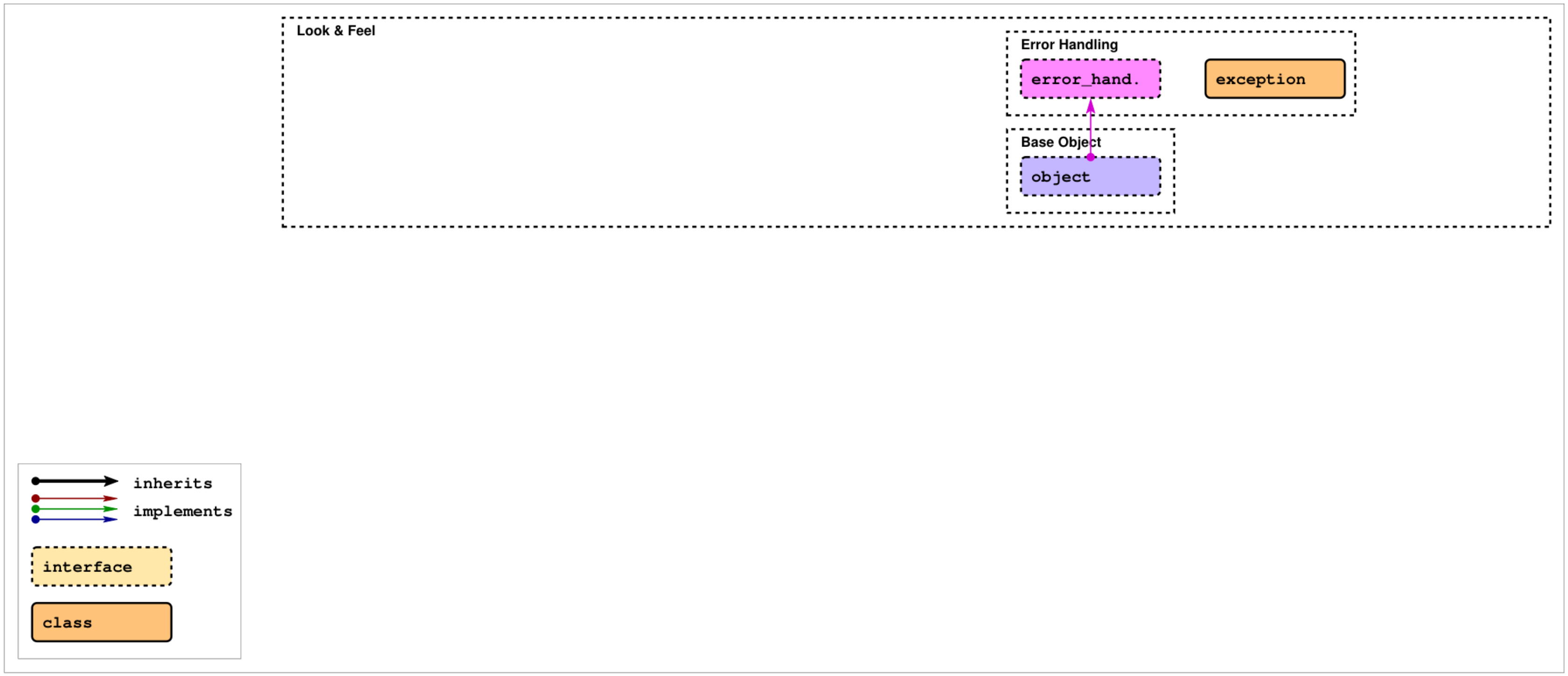
SAGA Class hierarchy

SAGA Look & Feel:
saga::object allows for object uuids, clone() etc.

SAGA

A Simple API for Grid Applications

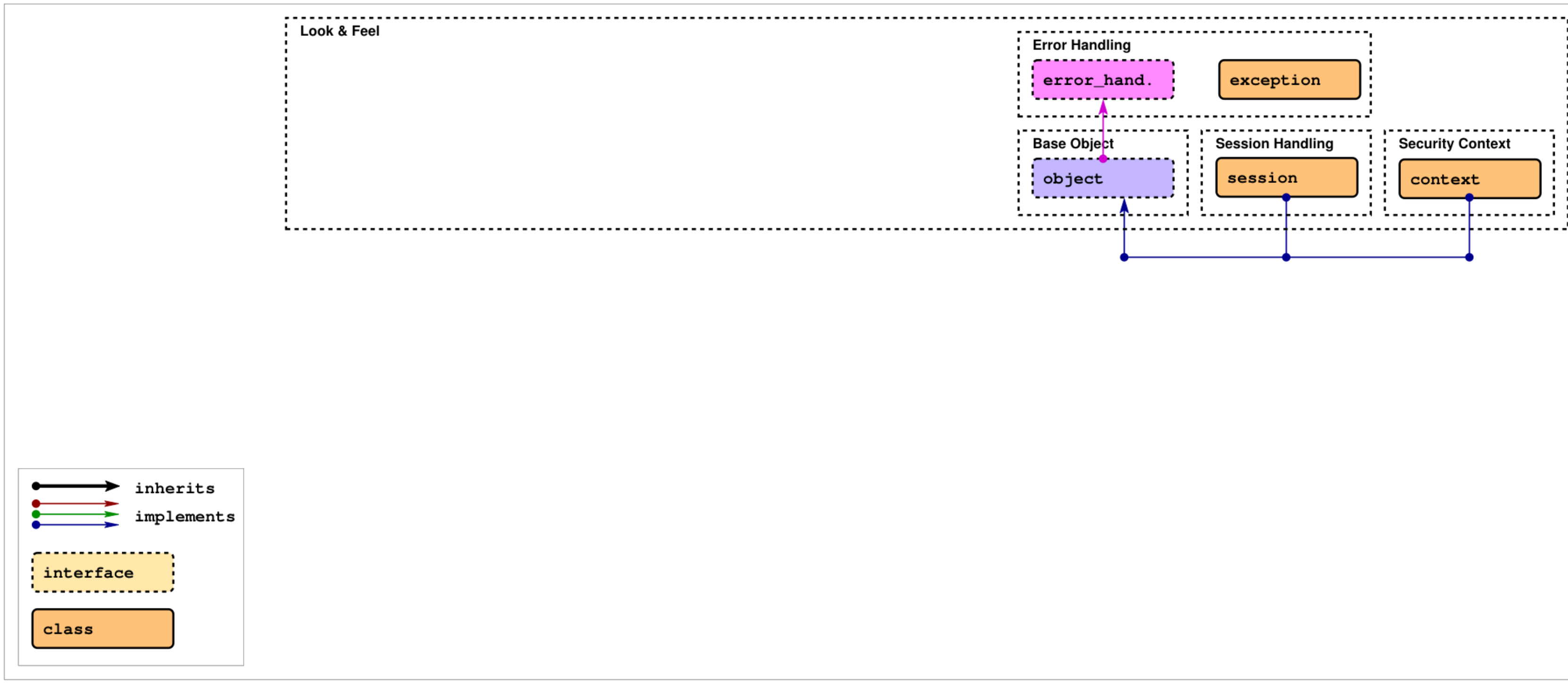
SAGA Class hierarchy



SAGA Look & Feel:
Errors are based on exceptions or error codes.

SAGA
A Simple API for Grid Applications

SAGA Class hierarchy

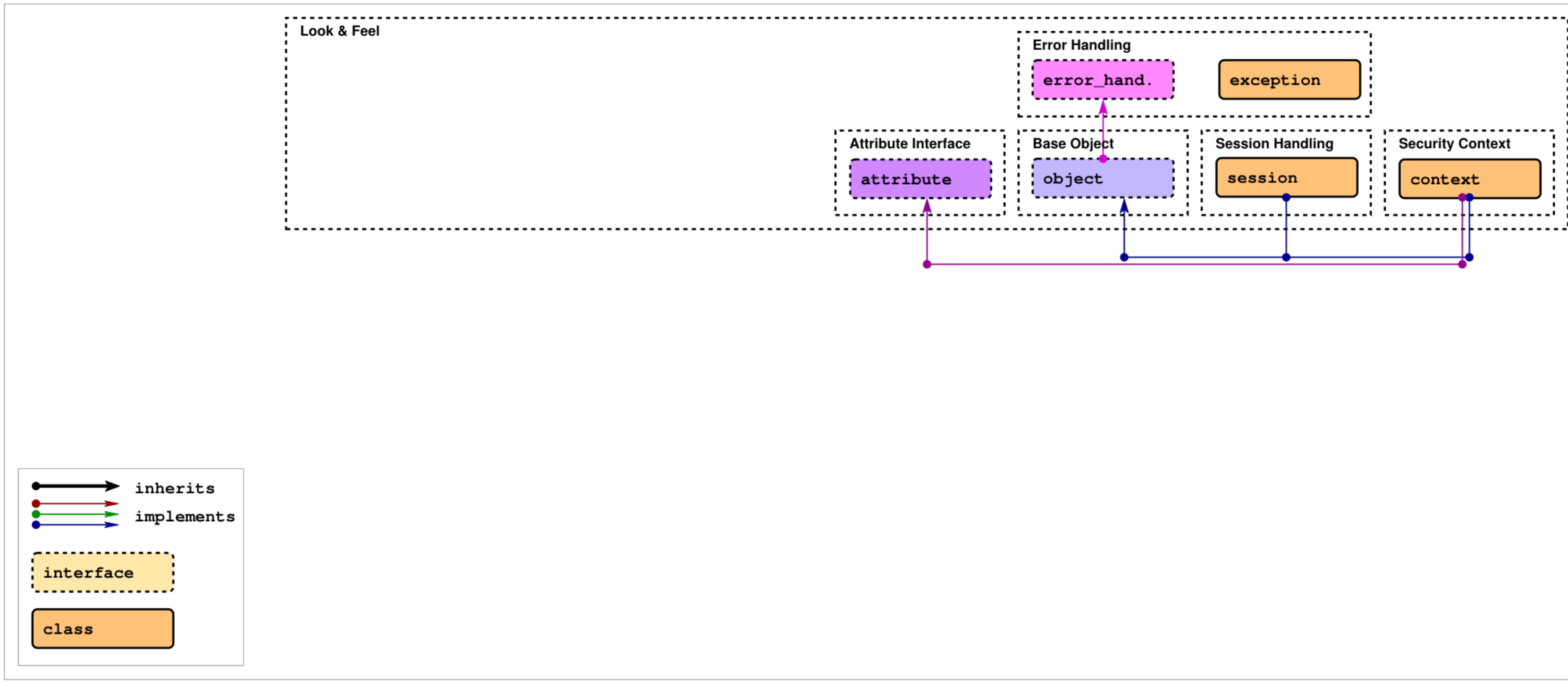


SAGA Look & Feel:
Session and credential management is hidden.

SAGA

A Simple API for Grid Applications

SAGA Class hierarchy

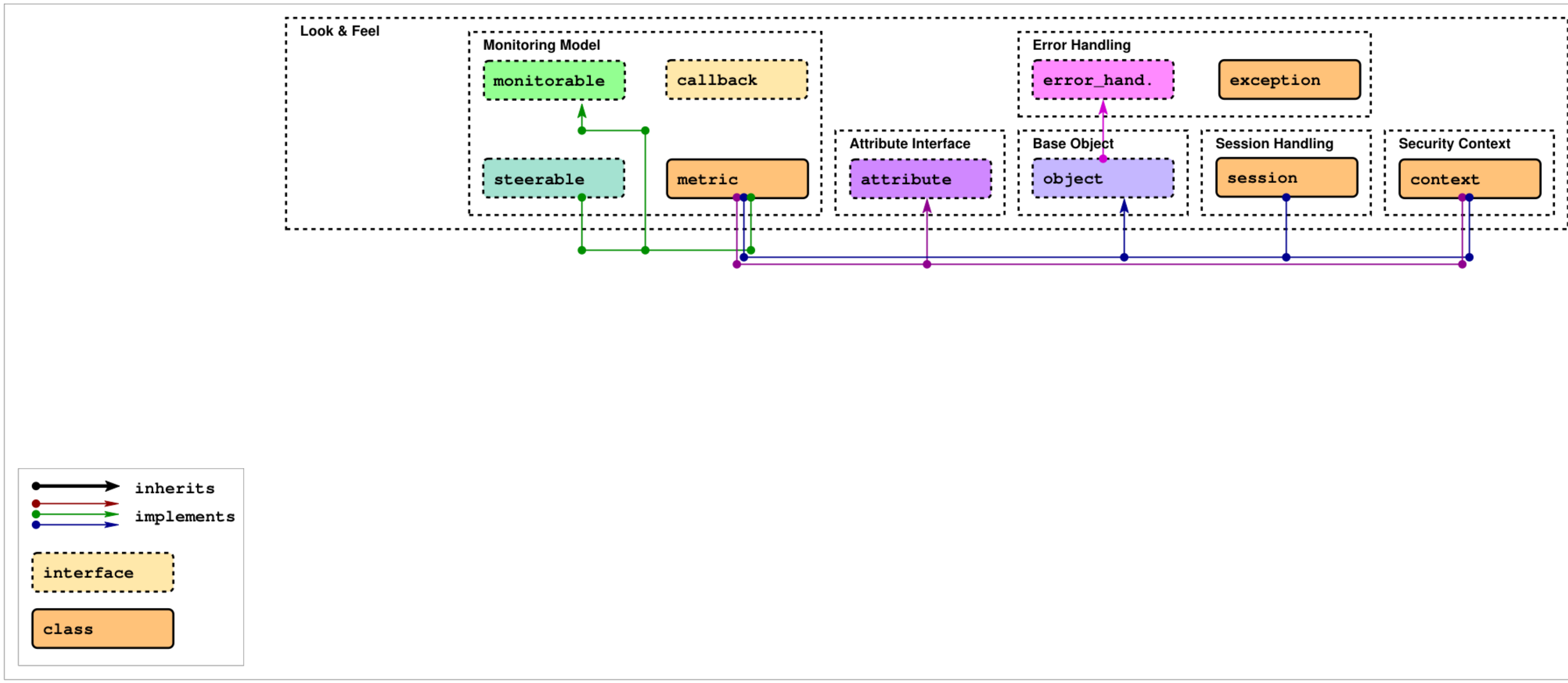


SAGA Look & Feel:

Attribute interface for meta data.

SAGA
A Simple API for Grid Applications

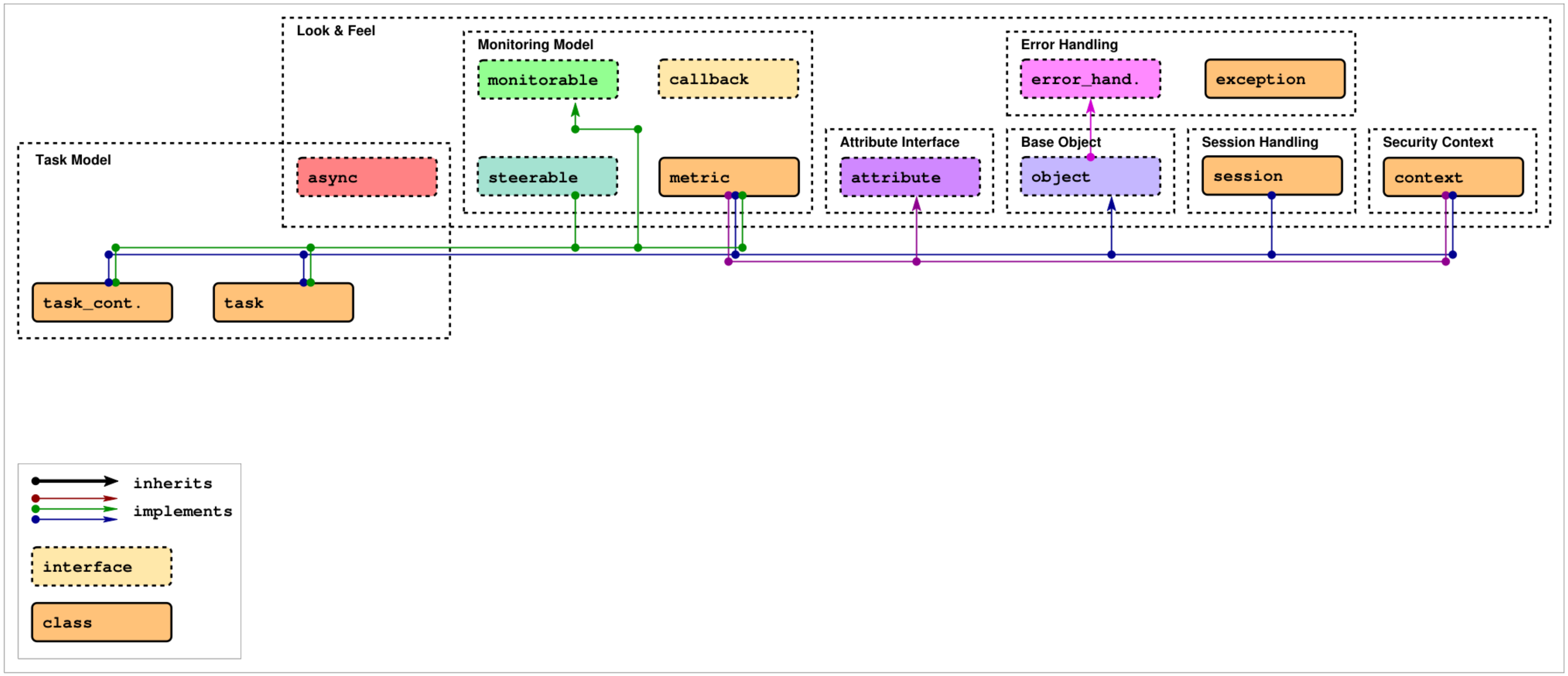
SAGA Class hierarchy



SAGA Look & Feel:
Monitoring includes asynchronous notifications.

SAGA
A Simple API for Grid Applications

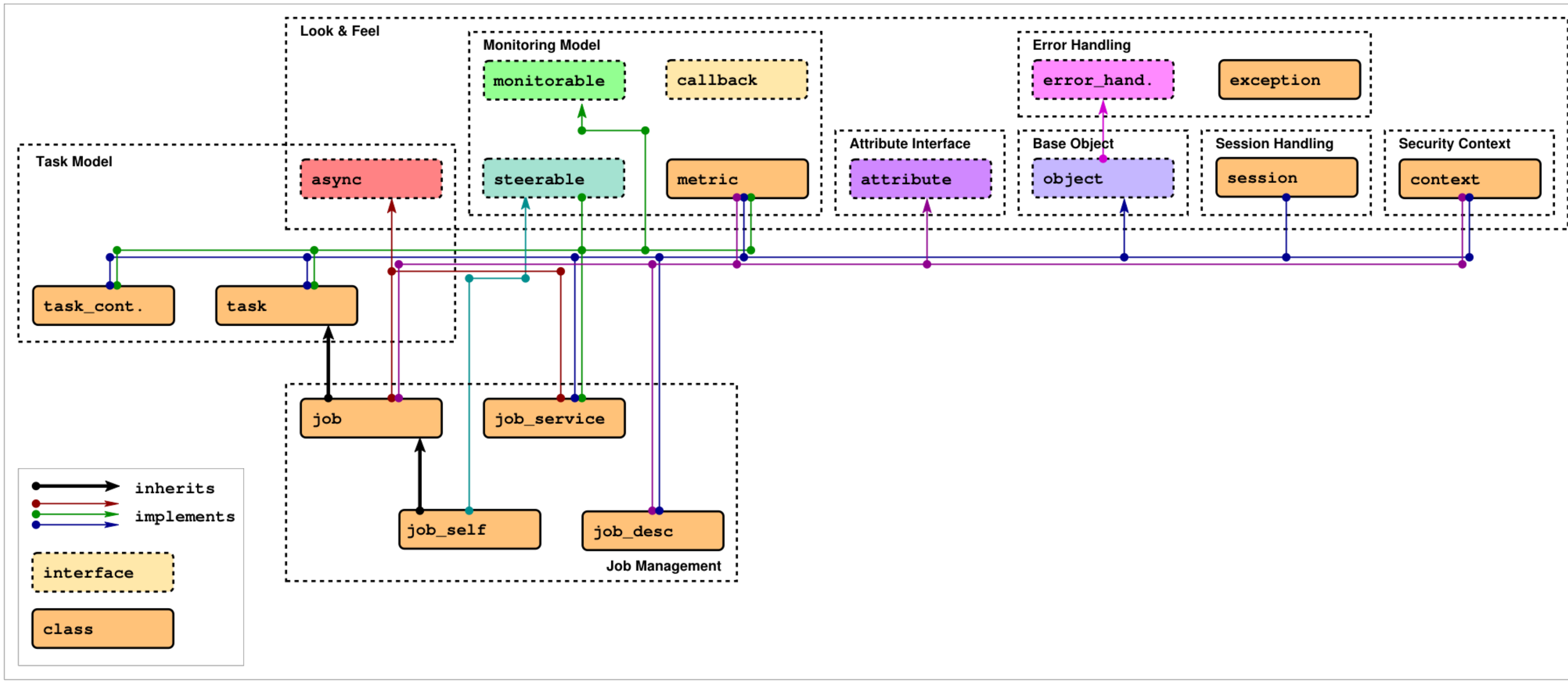
SAGA Class hierarchy



SAGA Look & Feel:
The task model adds asynchronous operations.

SAGA
A Simple API for Grid Applications

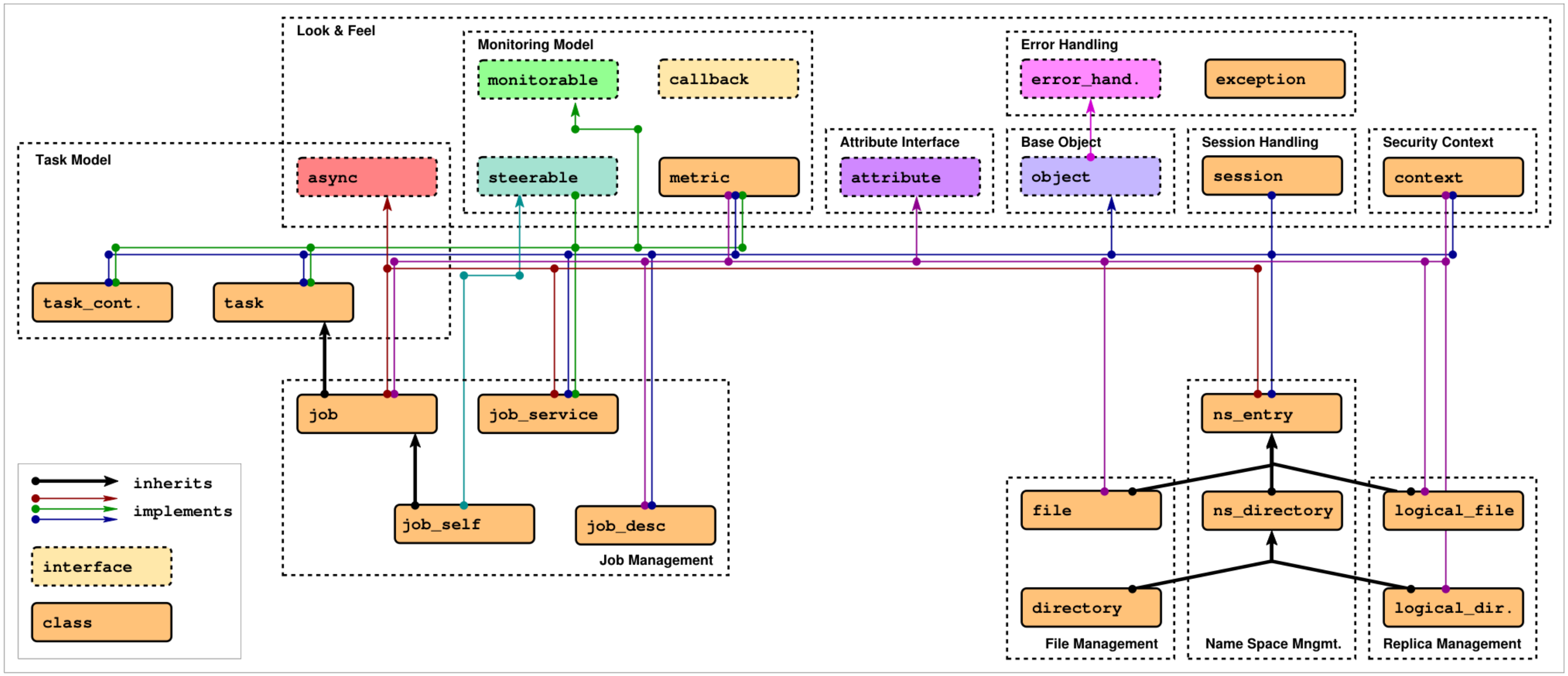
SAGA Class hierarchy



SAGA API Package 'job':
Create and manage remote processes.

SAGA
A Simple API for Grid Applications

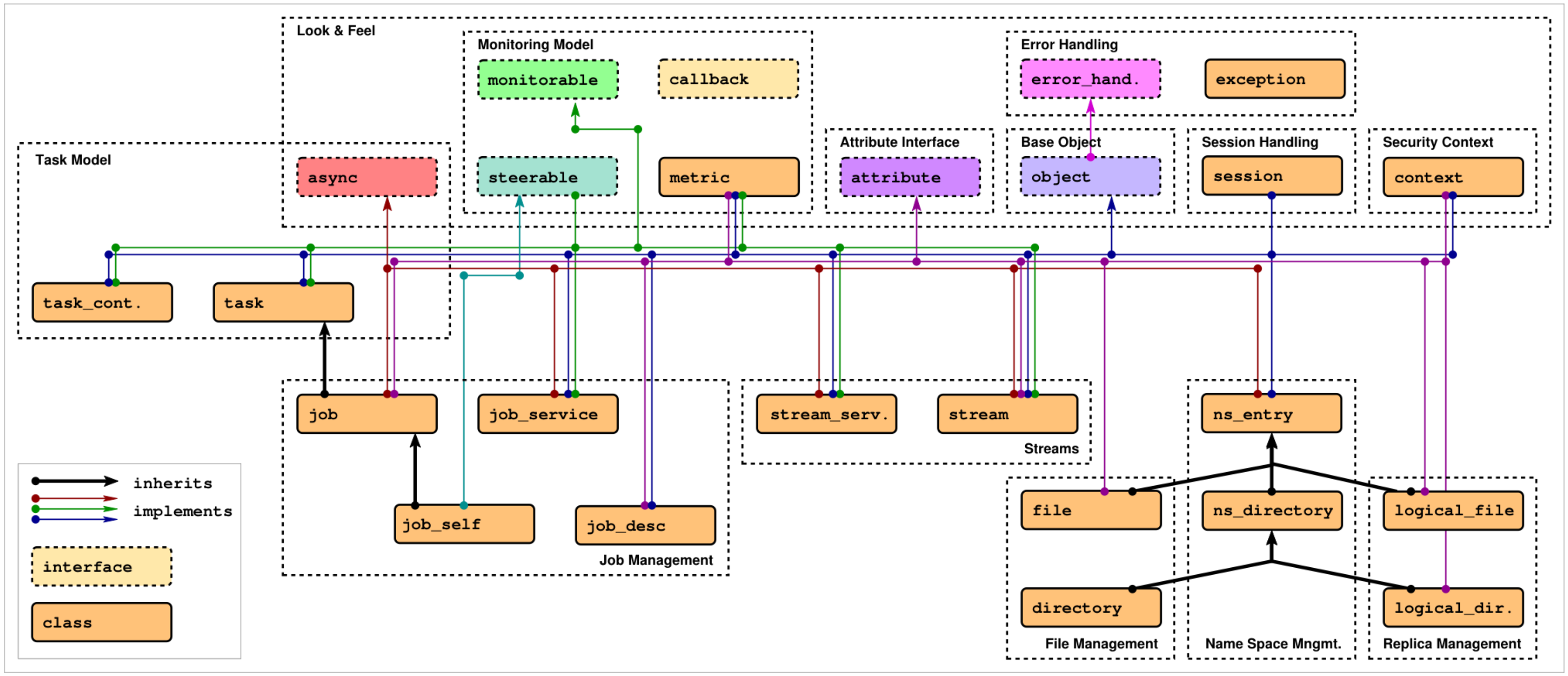
SAGA Class hierarchy



SAGA API Package 'name_spaces':
Manage files, replicas, etc.

SAGA
A Simple API for Grid Applications

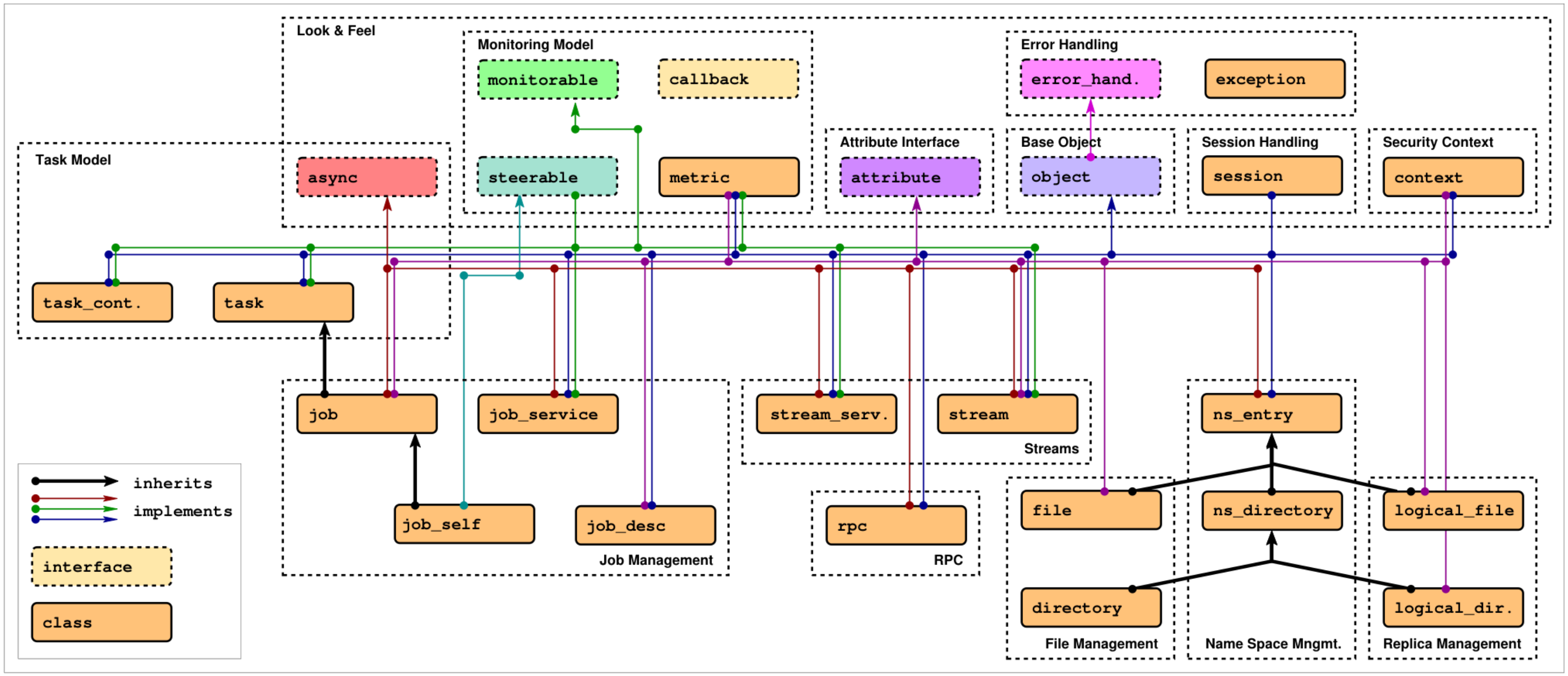
SAGA Class hierarchy



SAGA API Package 'stream':
SAGA rendering of BSD streams.

SAGA
A Simple API for Grid Applications

SAGA Class hierarchy



SAGA API Package 'rpc':
Remote procedure calls.

SAGA: Class hierarchy

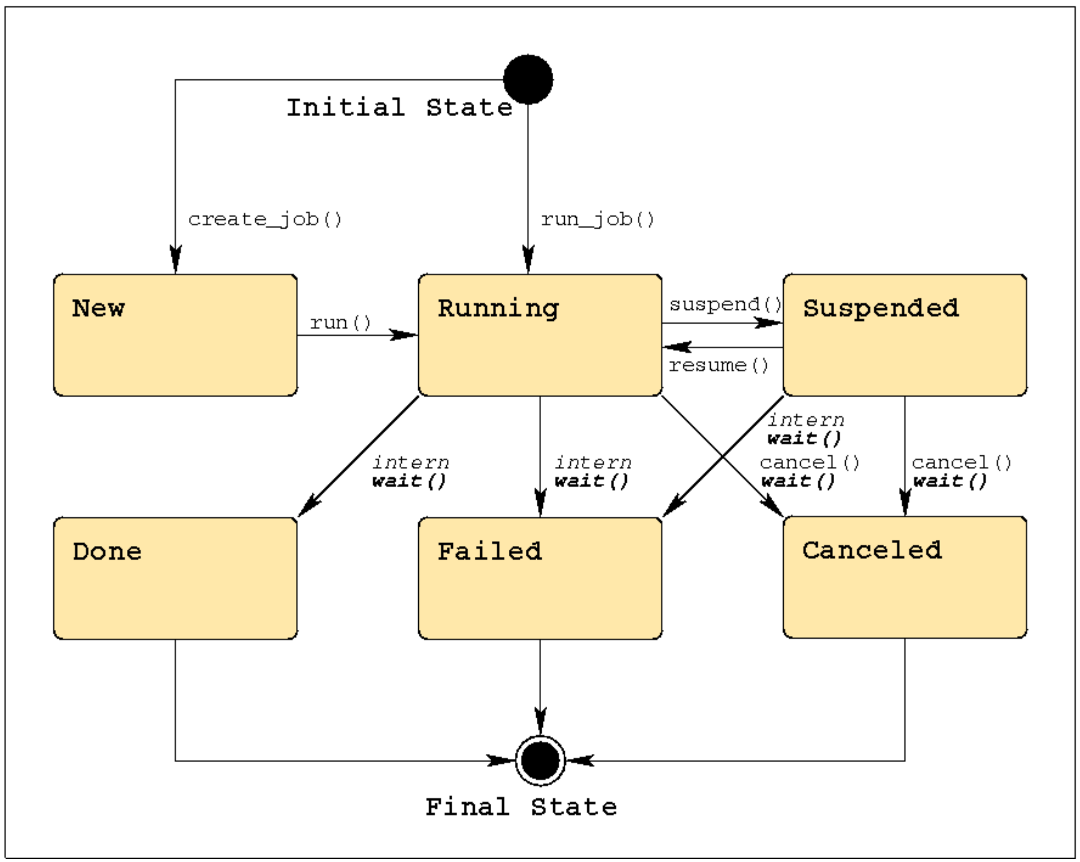
Functional API Packages

SAGA Job Package: Example 1

```
saga::job::description jd;
saga::job::service     js ("gram://remote.host.net");
saga::job              j = js.create_job (jd);

j.run ();

cout << "Job State: " << j.get_state () << endl;

j.wait ();

cout << "Retval " << j.get_attribute ("ExitCode") << endl;
```

SAGA
A Simple API for Grid Applications

SAGA Job Package: job states



SAGA Job Package: job operations

```
j.run     ();
j.wait    ();
j.cancel  ();

j.suspend ();
j.resume  ();

j.signal     (SIGUSR1);
j.checkpoint ();
j.migrate    (jd);
```

SAGA Job Package: job description

```
saga::job::description jd;

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

// pseudo code *blush*
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 Package: job description

Executable	WorkingDirectory	TotalPhysicalMemory
Arguments	*Interactive*	CPUArchitecture
Environment	Cleanup	OperatingSystemType
CandidateHosts	Input	*Queue*
SPMDVariation	Output	JobProject
TotalCPUCount	Error	*JobContact*
NumberOfProcesses	*JobStartTime*	FileTransfer
ProcessesPerHost	WallTimeLimit	
ThreadsPerProcess	TotalCPUTime	

SAGA Job Package: 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 Job Package: 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 ();  // fetch one job id
  saga::job j  = js.get_job (id);  // reconnect to job

  cout << id << " : " << j.get_state () << endl;
}
```

SAGA Job Package: 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
A Simple API for Grid Applications

SAGA Job Package: job::self

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

// self representst *this* SAGA application

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

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

SAGA Job Package: job::self

- represents the *calling application instance*
- self.signal (SIGUSR1);

  goes via the job manager, unlike

  ::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

SAGA Namespace Package

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

SAGA
A Simple API for Grid Applications

SAGA Namespace Package: example

```
saga::name_space::directory d ("ssh://remote.host.net//data/");

if ( d.is_entry ("a") && ! d.is_dir ("a") )
{
  d.copy ("a", "../b");
  d.link ("../b", "a", Overwrite);
}

list <saga::url> names = d.find ("*-{123}.text.");

saga::name_space::directory tmp = d.open_dir ("tmp/data/1",
                                               saga::name_space::CreateParents);

saga::name_space::entry data = tmp.open ("data.txt");

data.copy ("data.bak", Overwrite); // uses cwd
```

SAGA Namespace Package

- name space entries are opaque: the name space package can never look inside
- directories *are* entries (inheritance)
- inspection: `get_cwd()`, `get_url()`, `get_name()`, `exists()`, `is_entry()`, `is_dir()`, `is_link()`, `read_link()`
- manipulation: `create()`, `copy()`, `link()`, `move()`, `remove()`
- permissions: `permissions_allow()`, `permissions_deny()`
- wildcards are supported (POSIX influence...)

SAGA Filesystem Package

- implements name space interface
- adds access to **content** of namespace::entries (files)
- POSIX oriented: `read(), write(), seek()`
- optimizations: for distributed file access:
  - scattered I/O
  - pattern based I/O
  - extended I/O (from GridFTP)

SAGA Filesystem Package: Example

```
saga::filesystem::file f ("any://remote.host.net/data/data.bin");

char mem[1024];
saga::mutable_buffer buf (mem);

if ( f.get_size () >= 1024 )
{
  buf.set_data (mem + 0, 512);
  f.seek (512, saga::filesystem::Start);
  f.read (buf);
}

if ( f.get_size () >= 512 )
{
  buf.set_data (mem + 512, 512);
  f.seek (0, saga::filesystem::Start);
  f.read (buf);
}
```

SAGA
A Simple API for Grid Applications

SAGA Filesystem Package

- provides access to the **content** of filesystem entries (sequence of bytes)
- saga buffers are used to wrap raw memory buffers
- saga buffers can be allocated/managed by the SAGA implementation
- several incarnations of read/write: posix style, scattered, pattern based

SAGA Filesystem Package: Flags

```
enum flags {
  None              =    0,
  Overwrite         =    1,
  Recursive         =    2,
  Dereference       =    4,
  Create            =    8,
  Exclusive         =   16,
  Lock              =   32,
  CreateParents     =   64,
  Truncate          =  128,  // not on name_space
  Append            =  256,  // not on name_space
  Read              =  512,
  Write             = 1024,
  ReadWrite         = 1536,  // Read | Write
  Binary            = 2048   // only on filesystem
}
```

SAGA Advert Package

- persistent storage of application level information
- semantics of information defined by application
- allows storage of serialized SAGA objects (object persistency)
- ***very useful for bootstrapping and coordinating distributed application components***

SAGA Advert Package: Example

```
// example for browing my task adverts
saga::advert::directory todo ("any//remote.host.net/my_tasks/");

// pseudo vector code
list <saga::url> urls = todo.find ("*", ["priority=urgent"]);

while ( urls.size () )
{
  saga::advert ad (urls.pop_front ());
  std::cout << ad.get_attribute ("title")       << std::endl;
  std::cout << ad.get_attribute (« deadline")   << std::endl;
  std::cout << ad.get_attribute ("description") << std::endl << std::endl;
}
```

SAGA
A Simple API for Grid Applications

SAGA Advert Package: Example

```
// master side code: advertise (publish) a saga::file instance
saga::file f (url);

saga::advert ad ("any//remote.host.net/files/my_file_ad", Create);

ad.store_object (f);
```

```
// client side code: retrieve file instance
saga::advert ad ("any//remote.host.net/files/my_file_ad");

saga::file f = ad.retrieve_object ();
```

SAGA: Class hierarchy

Look & Feel Packages

SAGA Session: Example – default session

```
saga::ns_dir dir ("any://remote.host.net//data/");

if ( dir.is_entry ("a") && ! dir.is_dir ("a") )
{
  dir.copy ("a", "../b");
  dir.link ("../b", "a", Overwrite);
}

list <saga::url> names = dir.find ("*-{123}.text.");

saga::name_space::directory sub   = dir.open_dir ("tmp/");
saga::name_space::entry     entry = dir.open     ("data.txt");

entry.copy ("data.bak", Overwrite);
```

SAGA Session: Example – explicit session

```
saga::context c1 (saga::context::X509);
saga::context c2 (saga::context::X509);

c2.set_attribute ("UserProxy", "/tmp/x509up_u123.special");

saga::session s;

s.add_context (c1);
s.add_context (c2);

saga::name_space::dir dir (s, "any://remote.host.net/data/");
```

SAGA Session: Properties

- 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 can attach security tokens to a session
  - the default session has default contexts

SAGA Session: Lifetime

```
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");  // this works – session is sticky!
```

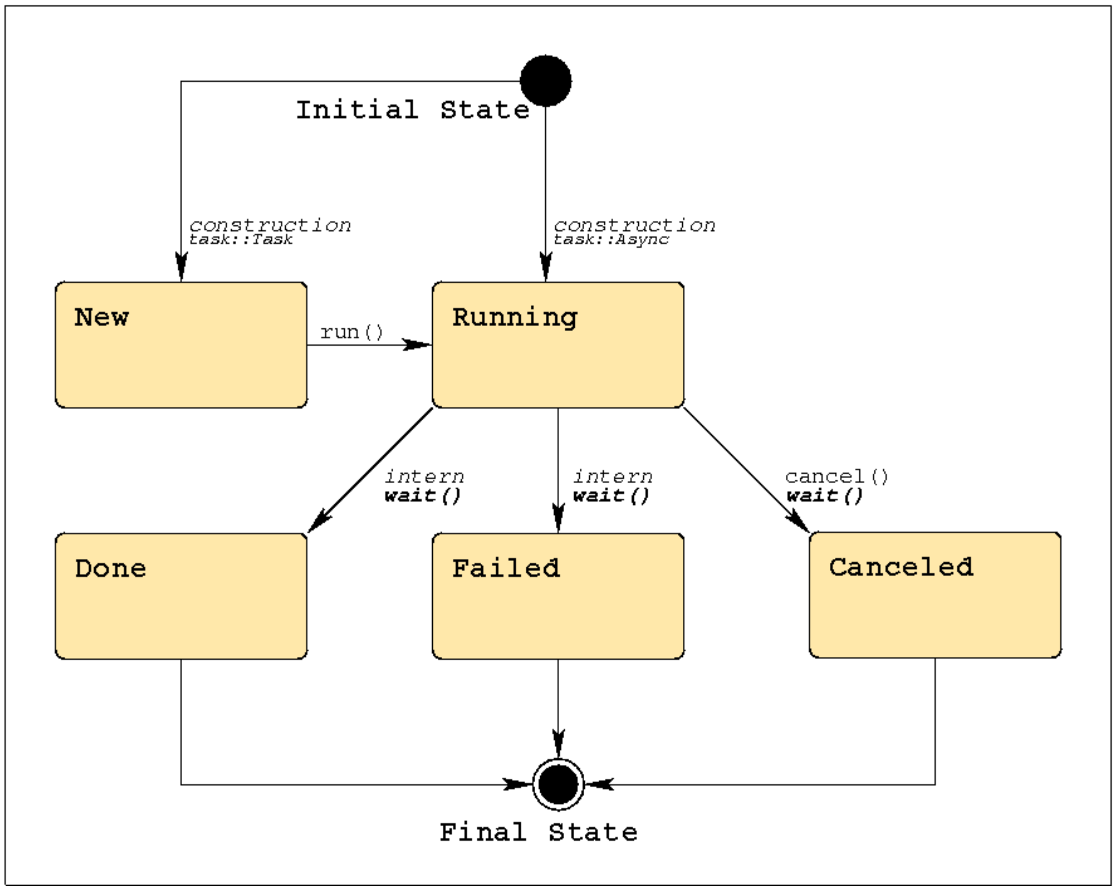
SAGA Tasks

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

SAGA

A Simple API for Grid Applications

SAGA Tasks: States



SAGA
A Simple API for Grid Applications

SAGA Tasks

- different versions for each method call: `Sync, Async, Task`
- signature basically the same
- differ in state of the task returned by that method
  - `Sync:` task is `Done`
  - `Async:` task is `Running` -> `wait();`
  - `Task:` task is `New` -> `run(); wait();`
- delayed exception delivery

```
if ( saga::task::Failed == task.get_state () )
{
    task.rethrow ();
}
```

SAGA

A Simple API for Grid Applications

SAGA Task: Example

```
 // normal method call, synchronous
 /* void */ file.copy ("data.bak");

 // async versions, never throw (use ’rethrow()’ on failure)
 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

 t3.run ();    // t3 now Running, too

 t2.wait ();
 t3.wait ();

// t1, t2, t3 are final (Done or Failed)
```

SAGA

A Simple API for Grid Applications

SAGA Task: Example

```
// normal method call, synchronous
size_t s = file.get_size ();

// async versions, never throw (use 'rethrow()' on failure)
saga::task t1 = file.get_size <saga::task::Sync>  ();
saga::task t2 = file.get_size <saga::task::Async> ();
saga::task t3 = file.get_size <saga::task::Task>  ();

// get_result: implies wait() and rethrow(), and thus can throw!
size_t s1 = t1.get_result <size_t> ();
size_t s2 = t2.get_result <size_t> ();
size_t s3 = t3.get_result <size_t> ();
```

SAGA Task Container: Example

```
// create task container
saga::task_container tc;

// add tasks
tc.add (t1);
tc.add (t2);
tc.add (t3);

// collective operations on all tasks in container
tc.run ();

saga::task done_task = tc.wait (saga::task::Any);

tc.wait (saga::task::All);
```

SAGA Task Container: Tasks and Jobs

```
// NOTE:
// class saga::job : public saga::task
// task container can thus manage tasks *and* jobs:

saga::task task = file.copy <saga::task::Async> ("b");
saga::job  job  = js.run_job ("remote.host.net", "/bin/date");

saga::task_container tc;

tc.add (task);
tc.add (job);

tc.wait (saga::task::All);
```