Open Forum | Open Standards

OGF-NL Masterclass

Grid APIs for Grid Environments

Agenda

- Grid Application Programming
- established Grid APIs
  (Globus, gLite, CoG, GAT, OGF)
- SAGA as Unifying approach
  - Requirements
  - Structure
  - Tutorial
  - Future Developments
- Discussion

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
- no need for Grid APIs
- *not a topic for this masterclass*

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
- *not a topic for this masterclass*

Grid Aware Applications

- aware of distribution, heterogenuity, VOs, dynamicity etc.
- usually dynamic and adaptive
- use cases: collaboration, adaptivity, optimization, scalability
- favourite technique: depending on Grid middleware
- need for Grid APIs
- *topic for this masterclass :-)*

Grid APIs and Frameworks

- often target on legacy applications (Unicore, Globus, Condor, VMs)
- some are distribution aware (MPICH-G, Ninf-G, . . .)
- few **A**PIs exist for Grid aware applications
  - GridFTP
  - GRAM
  - gLite
  - CoG
  - GAT

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

GridFTP: Connection Setup

```
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_off_t    size    = GLOBUS_NULL;
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
- syncronous and asyncronous calls

GRAM Example

GRAM: Job Submit

```
globus_gram_client_callback_allow   (callback_func,
                                      (void *) &Monitor,
                                      &callback_contact);

rc = globus_gram_client_job_request (rm_contact,
                                      specification,
                                      job_state_mask,
                                      callback_contact,
                                      &job_contact);
```

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 (ii)

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 an own job / resource description language
- most middlewares invent their own languages
- requirements are interpreted in different places (resource broker, queue manager, . . .)

gLite Example

gLite: 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,
                 JDLBuffer,
                 "/tmp/x509up_u202",
                 uploadURL_and_jobID,
                 0, false);
client.Start    ("https://cream-ce-01:8443/.../CREAM",
                 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 (Application level WSDL)

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: Job Submit

```
ResourceBroker                rb ();
SoftwareDescription           sd (("location",  "/bin/ls")
                                  ("arguments", "-l"));

HardwareDescription           hd (("memory.size",  1024.f)
                                  ("disk.size",    10.f)
                                  ("machine.type", "i686"));

Job job = broker.SubmitJob (JobDescription (sd, hd));
```

GAT

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

Summary (i)

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

Grid APIs within OGF

- OGF focuses on services, but APIs are needed to access those
- OGF supports uptake of Grids: APIs needed!
  - Distributed Resource Management Application API (DRMAA)
  - Remote Procedure Calls (GridRPC)
  - Checkpoint and Recovery (GridCPR)
  - Job Submission and Description Language (JSDL)
- numerous service interfaces (WSDL etc)

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 *jobtemplatet;

if ( ! ( jobtemplate = create_job_template (job_path, 5, 0) ) )
{
  fprintf (stderr, "create_job_template failed\n");
  return 1;
}

while ( ( drmaa_errno = drmaa_run_job (jobid,
                                       sizeof (jobid)-1,
                                       jobtemplate,
                                       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];
grpc_function_handle_t handle;

grpc_initialize (argv[1]);

initMatA (N, A);   initMatB (N, B);   /* initialize */

grpc_function_handle_default (&handle, "mmul/mmul");

if ( grpc_call (&handle, N, A, B, C) != GRPC_NO_ERROR)
{
  fprintf (stderr, "Error in grpc_call\n");
  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 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)

OGF: JSDL

JSDL: Simple Job

```
 <jsdl:JobDefinition>
   <JobDescription>
      <Application>
         <jsdl-posix:POSIXApplication>
            <OpenDescriptorsLimit>64</OpenDescriptorsLimit>
         </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 surprisingly flat
- maps well to existing JDLs, but is ’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**

SAGA

Simple API for Grid Applications

SAGA overview

- SAGA API structure and scope
- planned extensions
- coding tutorial
- implementation status

OGF: APIs

- **SAGA**: **S**imple **A**PI for **G**rid **A**pplications
- OGF approach to a uniform API layer (facade, top-down)
- defines application level abstractions
- extensible (stable look & feel + API packages)
- major influences: GAT, CoG, DRMAA, GridRPC, LSF, OREP, JSDL, . . .
- simplicity versus control: 80:20 rule

SAGA Intro: Example 1

SAGA: File Management

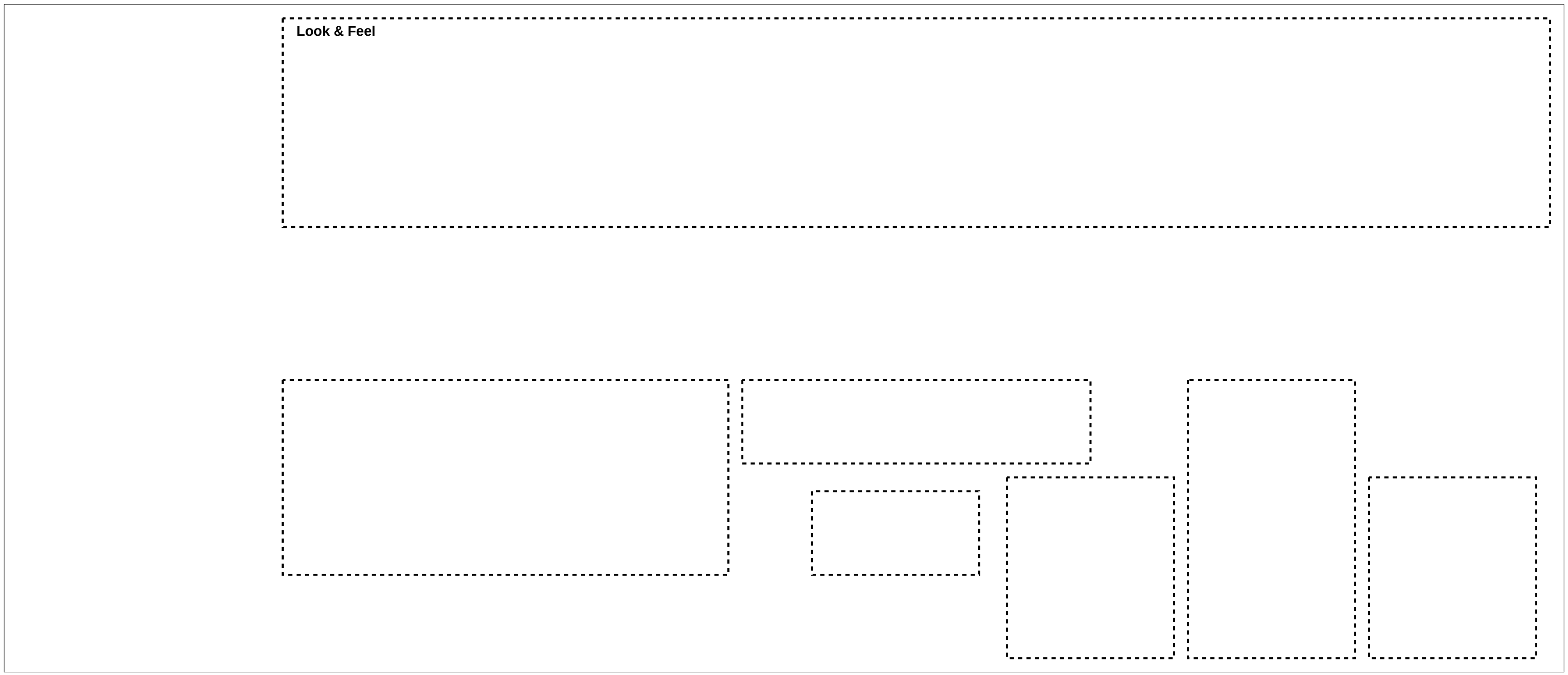
```
saga::directory dir ("any://remote.host.net//data/");

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");
```

SAGA Intro: Example 2

SAGA: Job Submission

```
saga::job_description jd;
saga::job_service     js ("any://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_get_attribute ("ExitCode") << endl;
```

SAGA: Class hierarchy



SAGA Look & Feel:

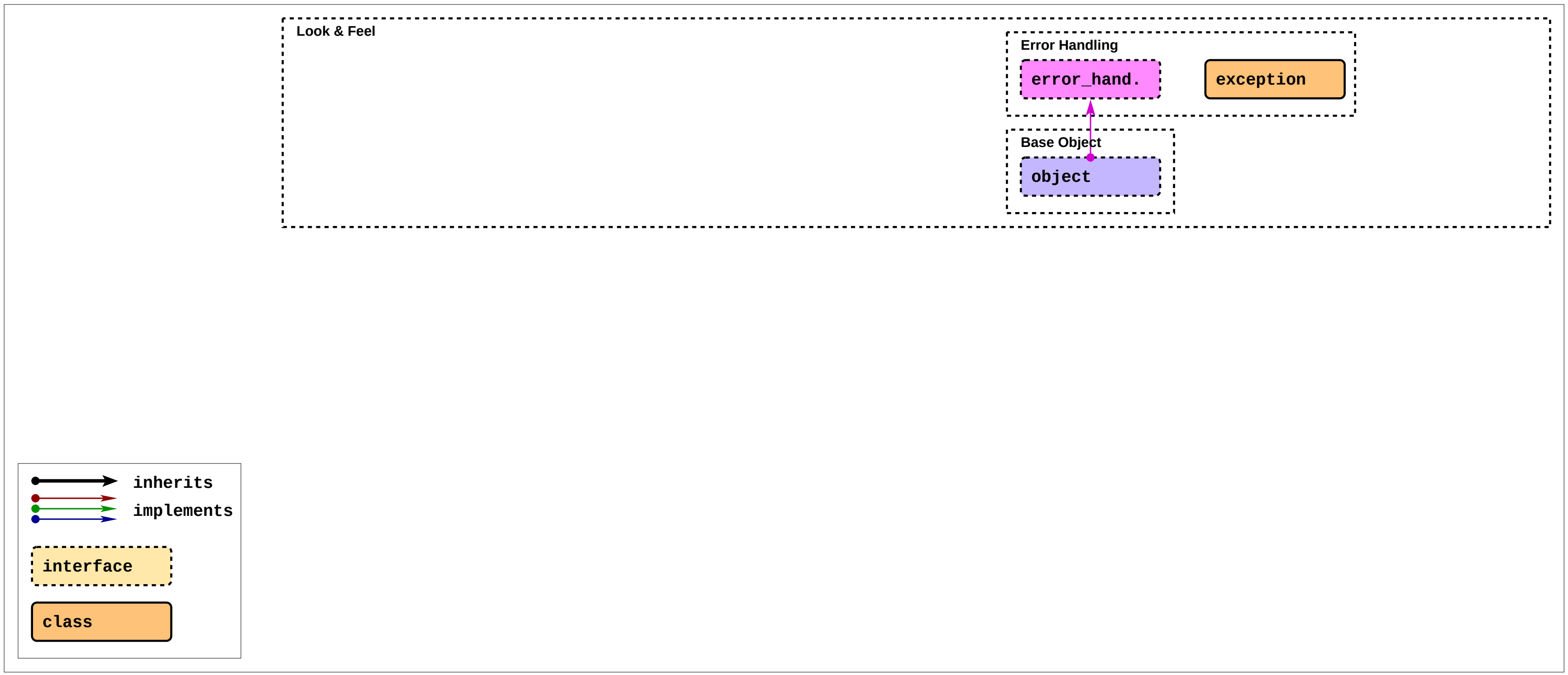
`saga::object` allows for object uuids, `clone()` etc.

SAGA: Class hierarchy

SAGA Look & Feel:

errors are based on exceptions or error codes.

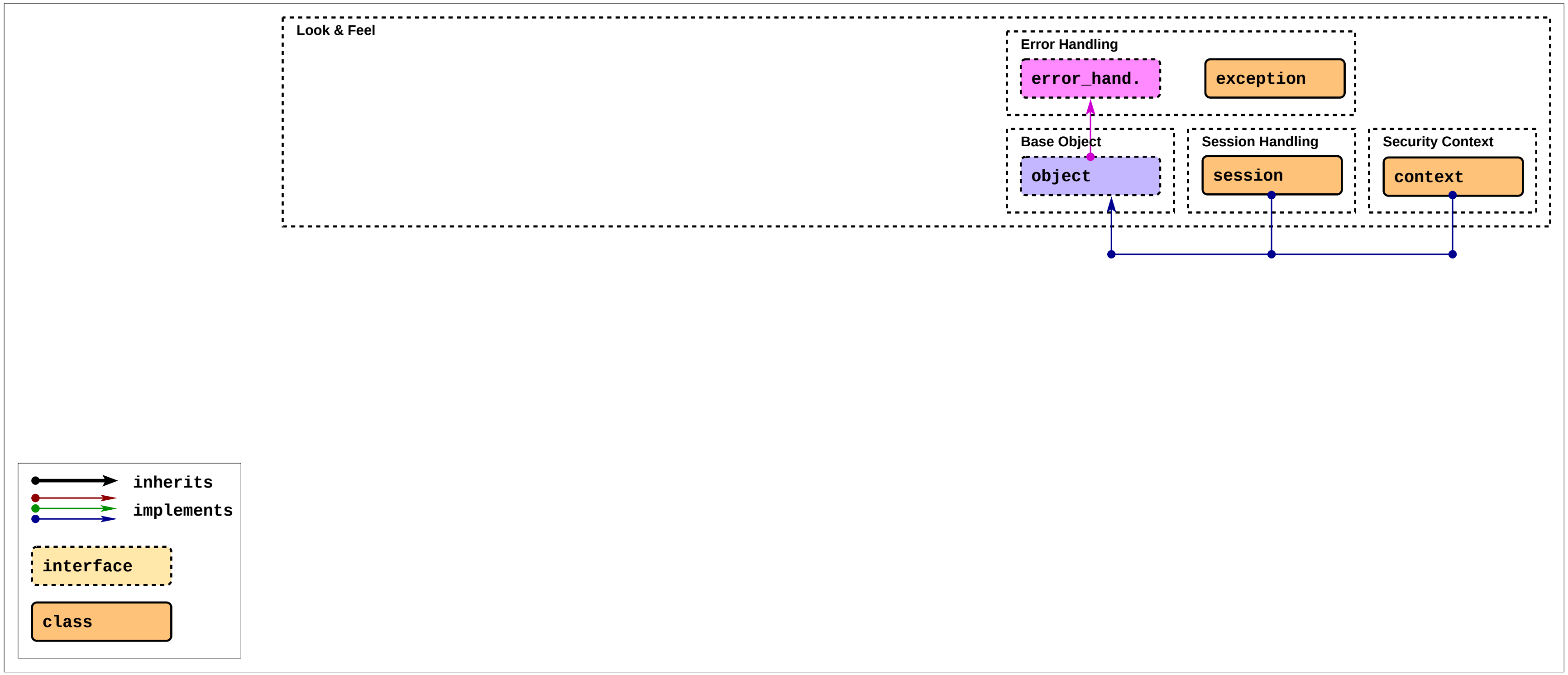
SAGA: Class hierarchy



SAGA Look & Feel:

session and credential management is hidden.

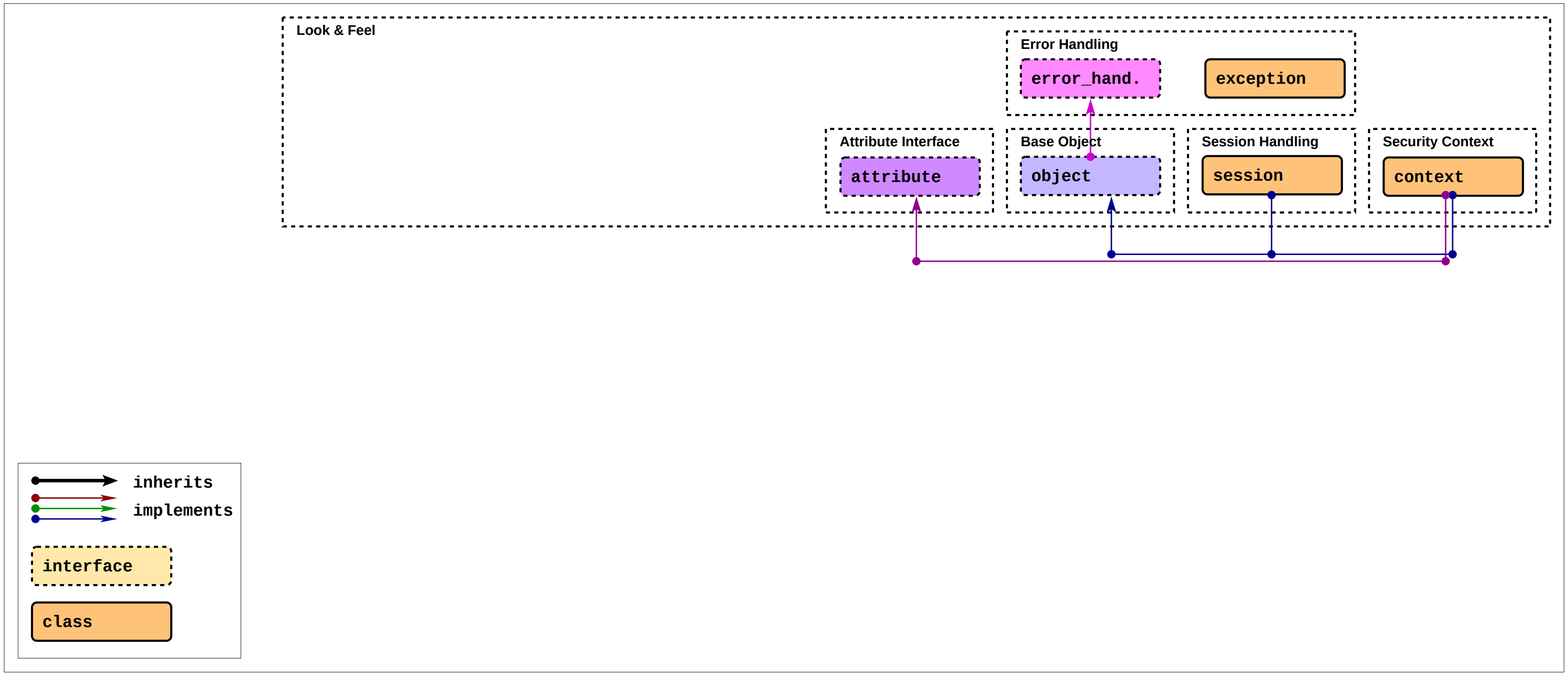
SAGA: Class hierarchy



SAGA Look & Feel:

Attribute interface for meta data.

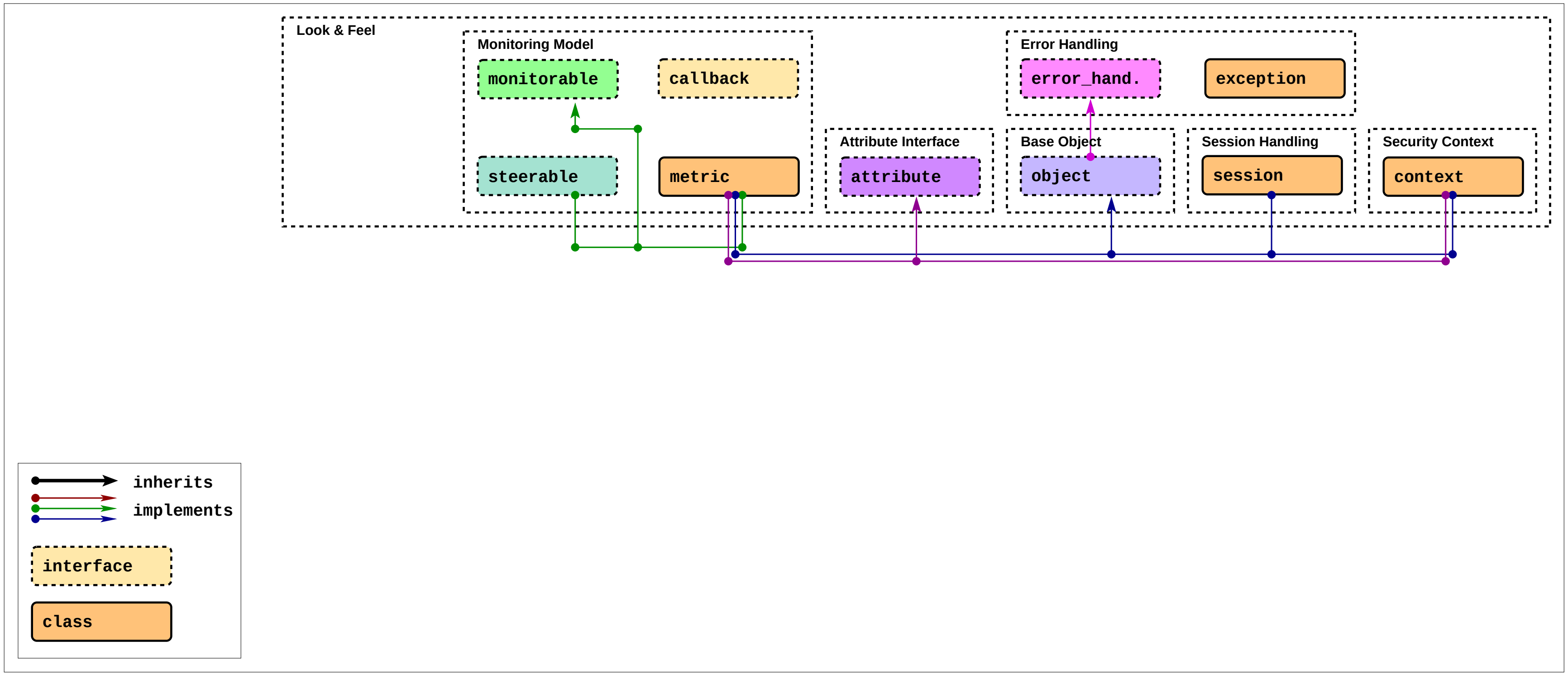
SAGA: Class hierarchy



SAGA Look & Feel:

Monitoring includes asynchronous notifications.

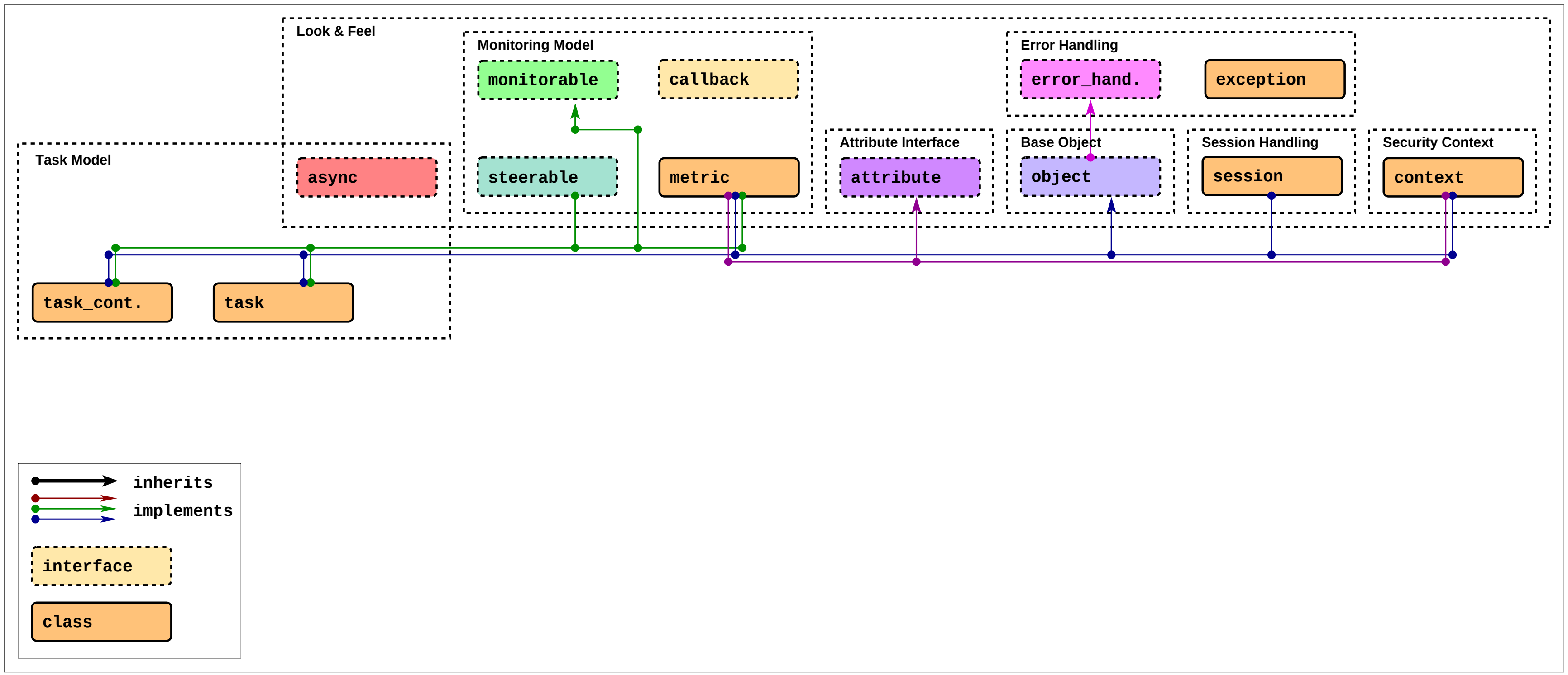
SAGA: Class hierarchy



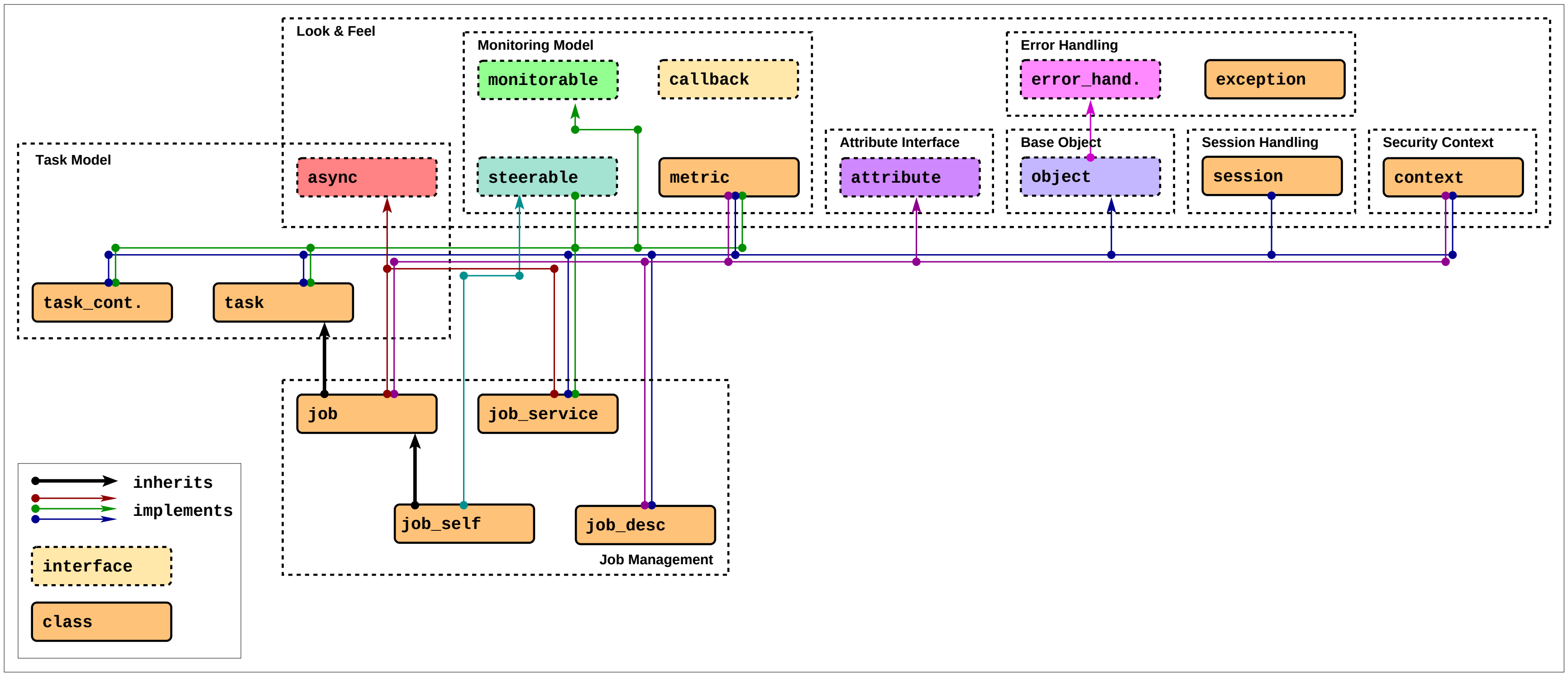
SAGA Look & Feel:

the task model adds asynchronous operations.

SAGA: Class hierarchy



SAGA: Jobs

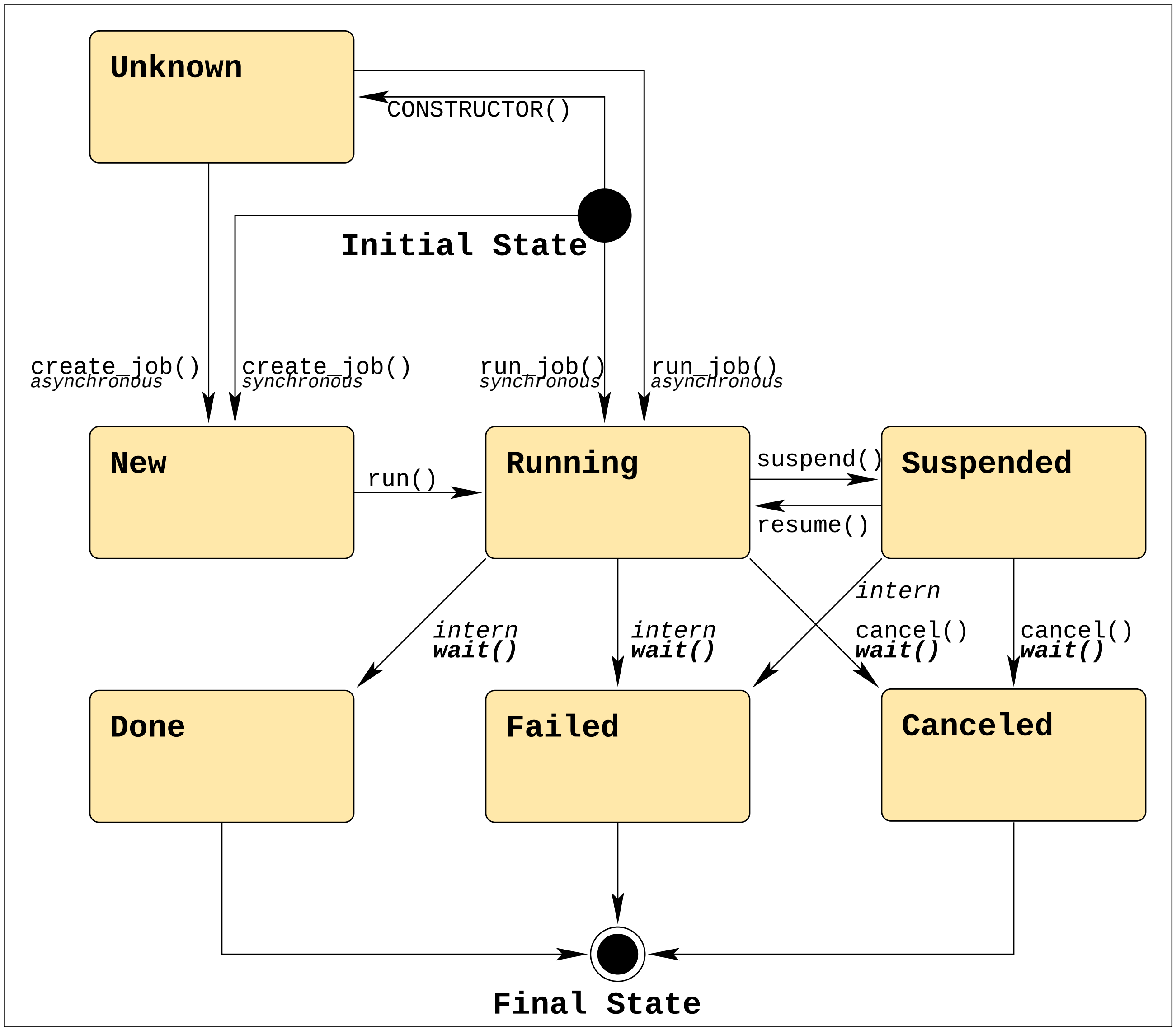


Look & Feel
Monitoring Model
monitorable
callback
steerable
metric
Error Handling
error_hand.
exception
Task Model
async
Attribute Interface
attribute
Base Object
object
Session Handling
session
Security Context
context
task_cont.
task
job
job_service
job_self
job_desc
Job Management
inherits
implements
interface
class

SAGA: Jobs

- `job_service` used `job_description` to create `job`
- `job_description` attributes are based on JSDL
- state model is based on 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_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_get_attribute ("ExitCode") << endl;
```

SAGA Examples: Jobs

jobs (cont.)

```
saga::job j = js.create_job (jd);

j.run ();

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

j.checkpoint ();

j.migrate (jd);

j.signal (SIGUSR1);

j.cancel ();
```

SAGA Examples: Jobs

jobs (cont.)

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

self.checkpoint ();

self.migrate (jd);

self.signal (SIGUSR1);

self.cancel ();
```

SAGA Examples: Jobs

jobs (cont.)

```
list<string> ids = js.list ();

while ( ids.size () )
{
  string    id = list.pop_front ();
  saga::job j  = js.get_job (id);

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

SAGA Examples: Job Descr.

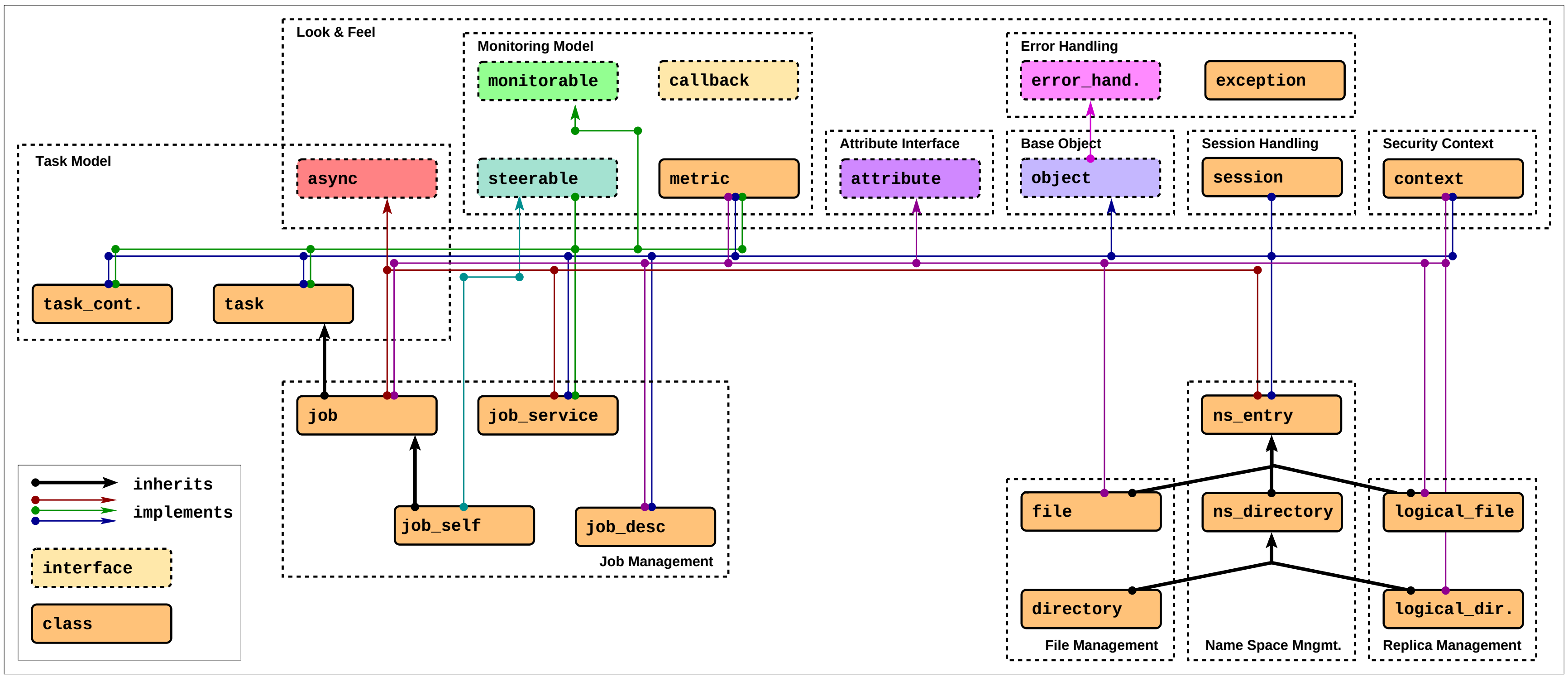
job description - JSDL based

```
saga::job_description jd;

jd.set_attribute ("Executable",        "/bin/tail");
jd.set_attribute ("Arguments",         "-n, 20, -f, all.log");
jd.set_attribute ("Environment",       "TMPDIR=/tmp/");
jd.set_attribute ("WorkingDirectory", "data/");
jd.set_attribute ("FileTransfer",      "last.log >> all.log");
jd.set_attribute ("Cleanup",           "False");
```

SAGA Examples: Job Descr.

Executable	Arguments	Environment
WorkingDirectory	JobInteractive	
Input	Output	Error
JobContact	JobName	FileTransfer
Cleanup	JobStartTime	Deadline
WallTimeLimit	WallclockSoftLimit	CPUTimeLimit
TotalCPUCount	TotalPhysicalMemory	CPUArchitecture
OperatingSystemType	CandidateHosts	Queue
JobID	ExecutionHosts	

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

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);
```

SAGA Examples: Files

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);
}
```

SAGA Examples: Files

file access - scattered I/O

```
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);
}
```

SAGA Examples: Files

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);
}
```

SAGA Examples: Files

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

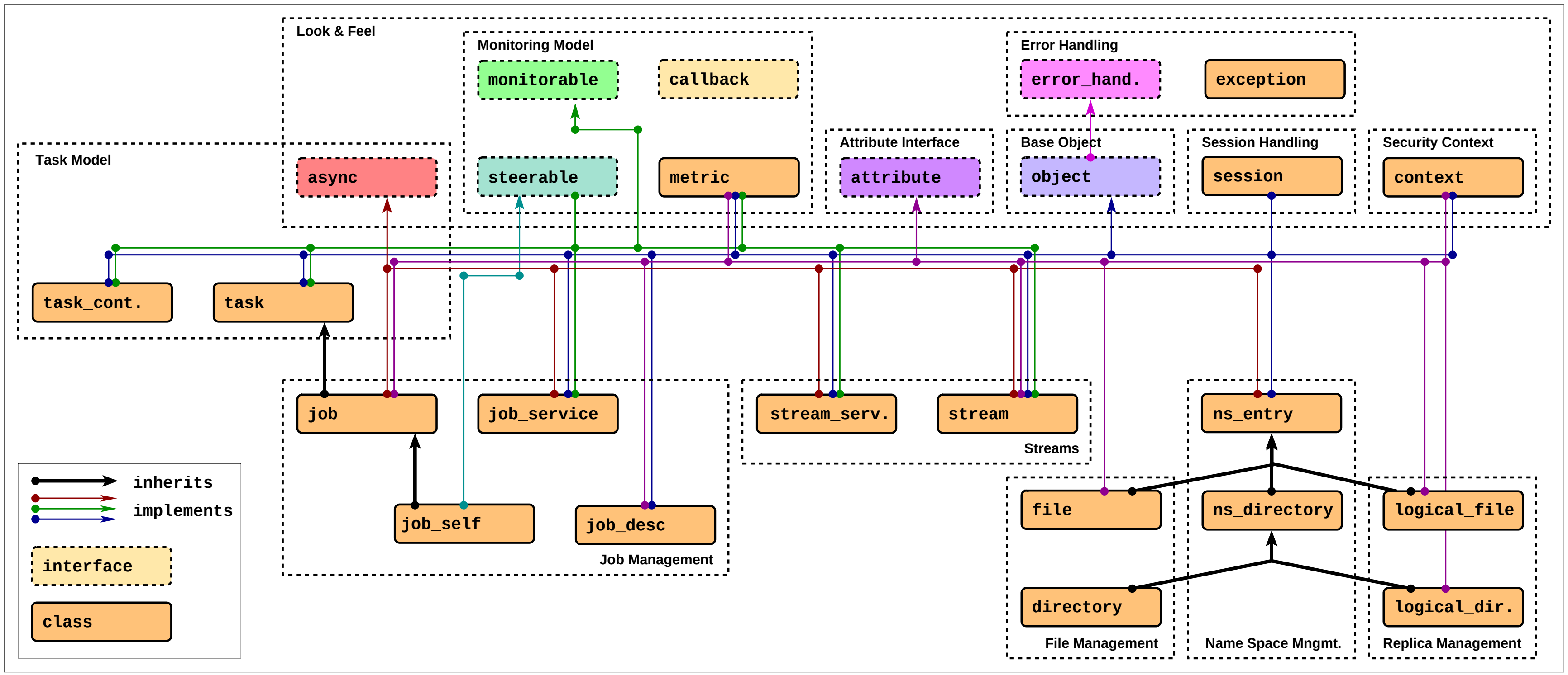
```
saga::logical_directory dir ("raptor://remote.host.net/data/");

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

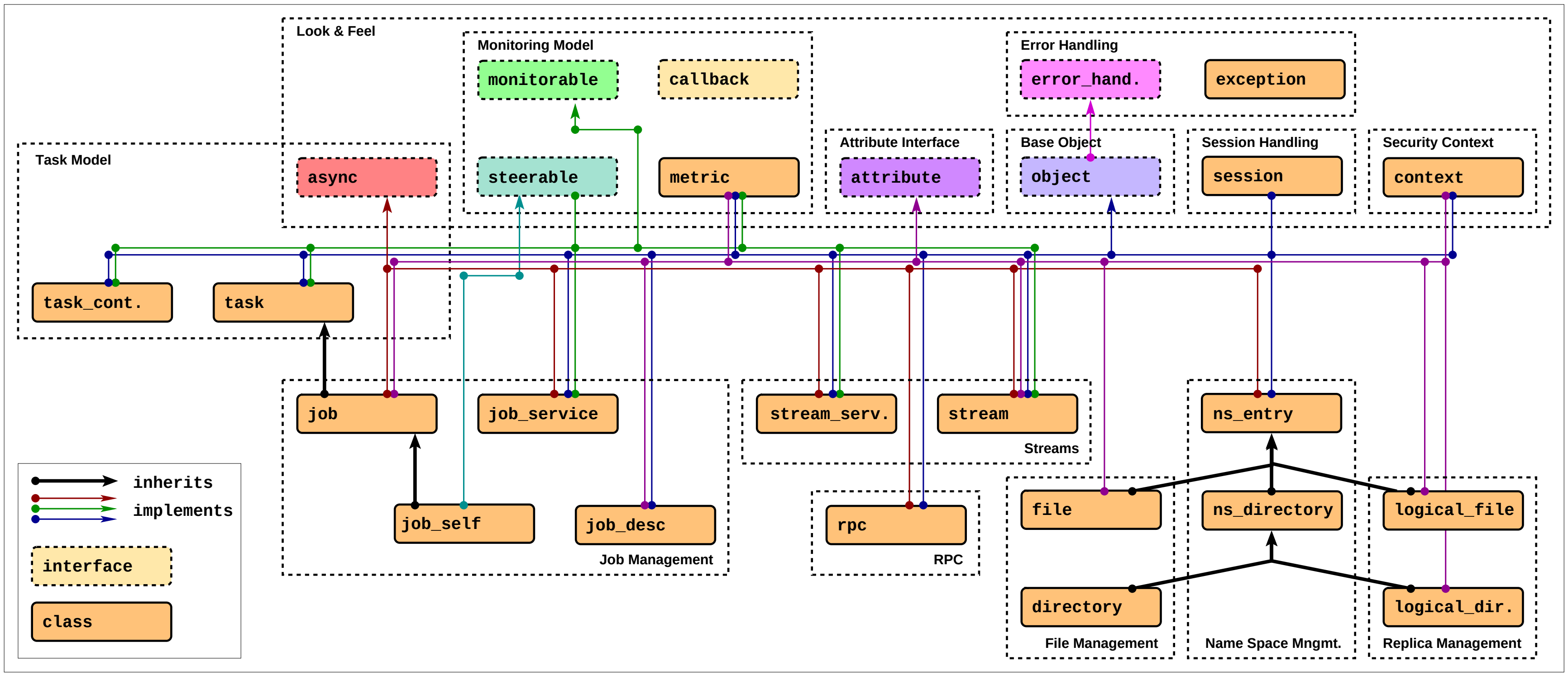
stream server

```
saga::stream_service ss ("tcp://localhost:1234");

saga::stream_client sc = ss.serve ();

sc.write ("Hello client", 13);
```

stream client

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

sc.connect ();
sc.read    (buf, 13);

cout << buf << endl;
```

SAGA: RPC



Look & Feel
Monitoring Model
monitorable
callback
steerable
metric
Error Handling
error_hand.
exception
Task Model
async
Attribute Interface
attribute
Base Object
object
Session Handling
session
Security Context
context
task_cont.
task
job
job_service
stream_serv.
stream
Streams
ns_entry
inherits
implements
job_self
job_desc
Job Management
rpc
RPC
file
ns_directory
logical_file
interface
class
directory
logical_dir.
File Management
Name Space Mngmt.
Replica Management

SAGA: RPC

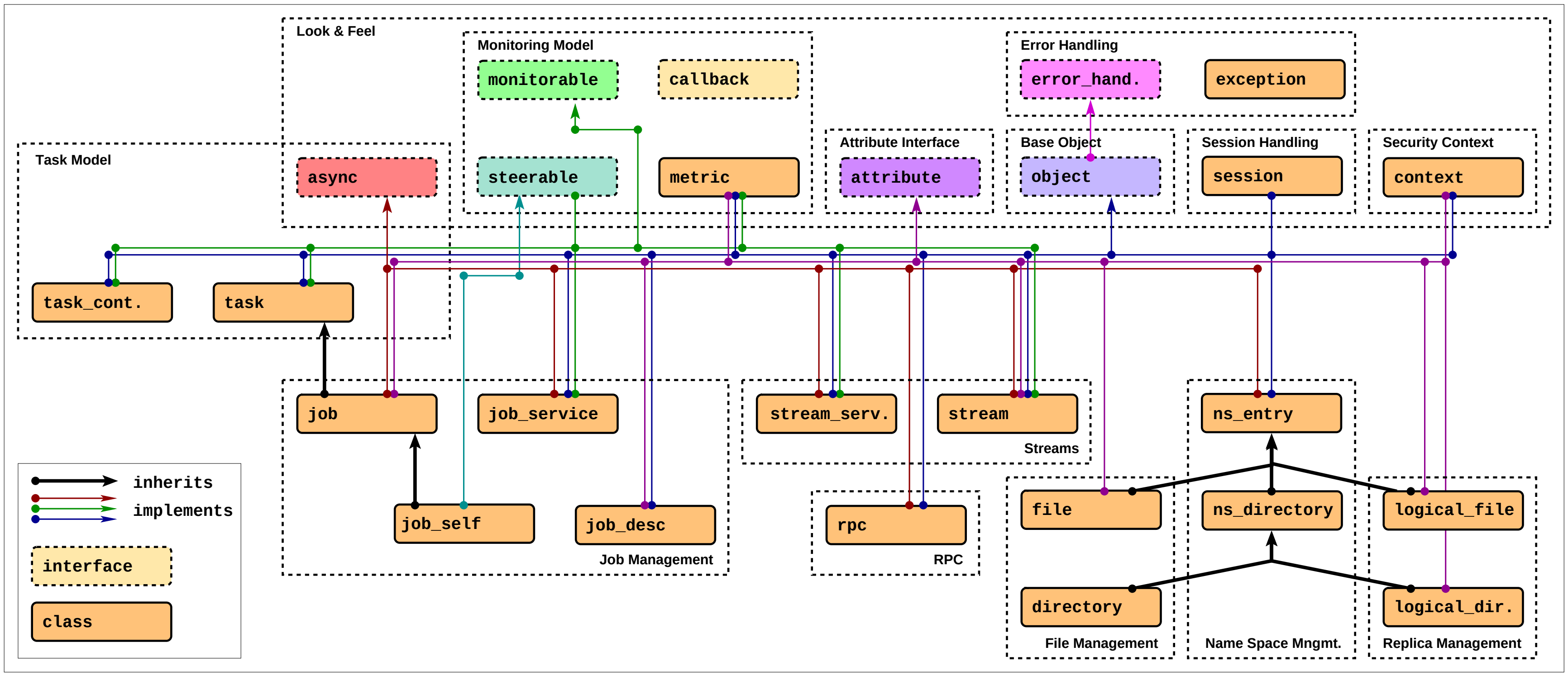
- maps GridRPC standard into the SAGA look & feel
- parameters are stack of structures (similar to scattered I/O)
- future revision will work on optimized data handling

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 ());
```

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

SAGA Examples: Session

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);
```

SAGA Examples: Session

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/");
```

SAGA Examples: Session

session inheritance

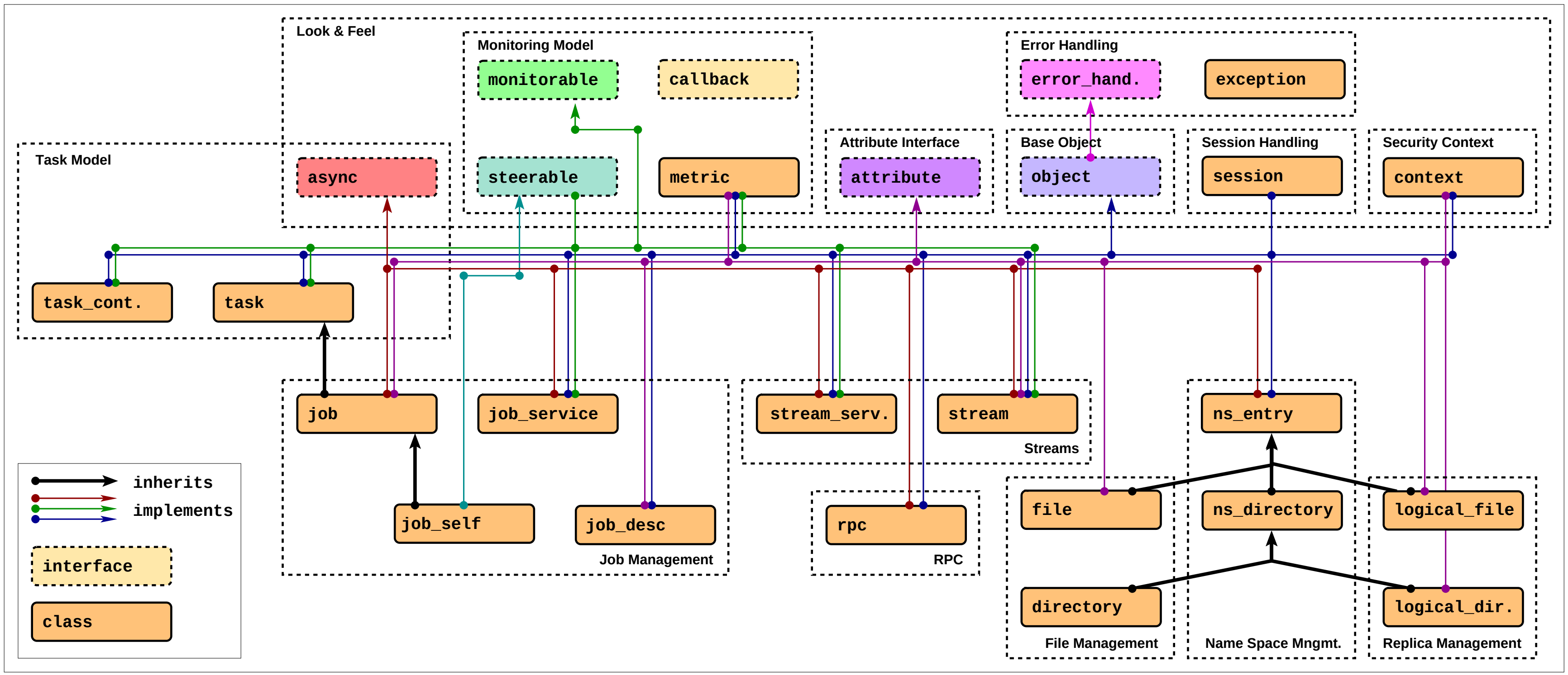
```
saga::dir  dir (s, "gridftp://remote.host.net/data/");

saga::file file = dir.open ("data.bin");

s.remove_context (c1);
s.remove_context (c2);

file.copy ("data.bin.bak");   // works - state is sticky!
```

SAGA Examples: Session

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

pull monitoring

```
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;
  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;
      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

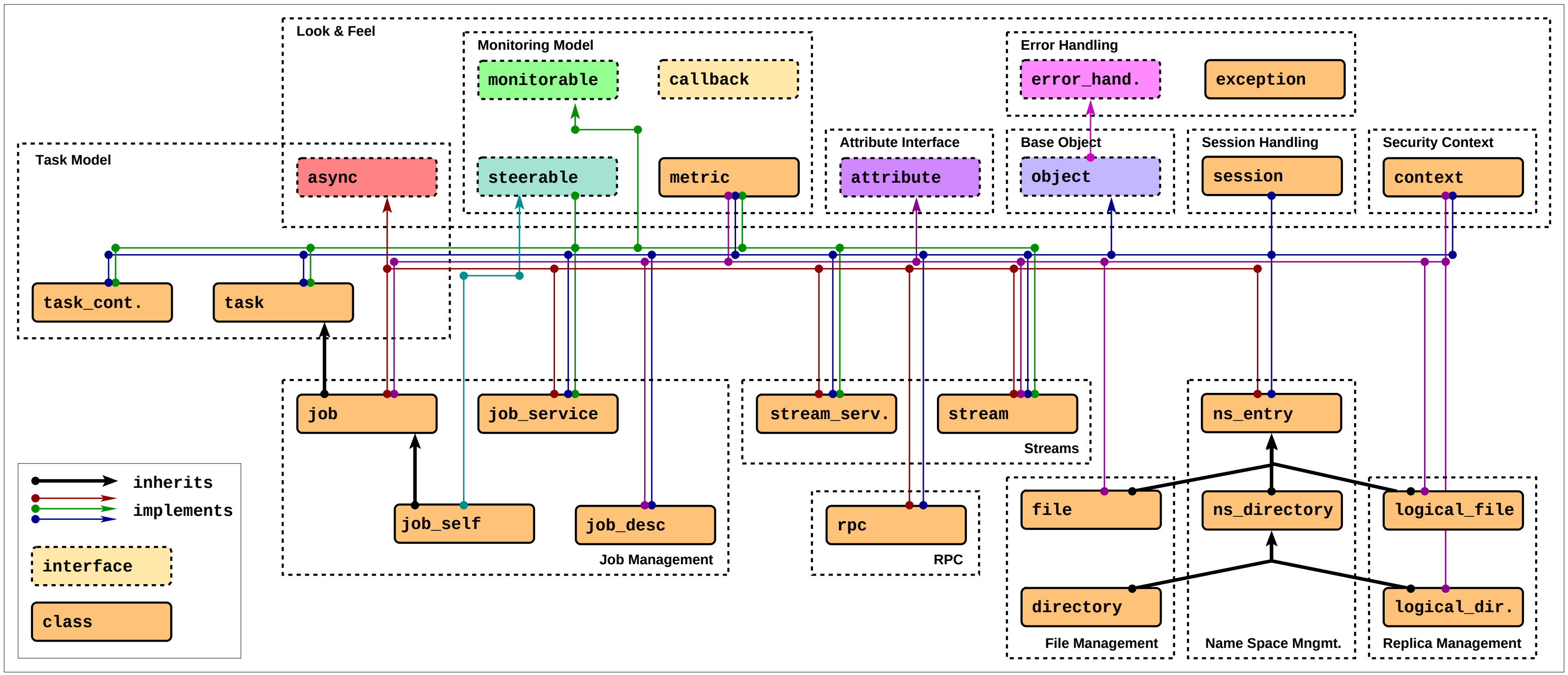
```
class my_cb : public saga::callback
{
  public:
    bool cb (saga::monitorable obj,
             saga::metric      m,
             saga::context     c)
    {
      cout << "Memory Usage: " << m.get_value () << endl;
      return (true);
    }
};

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;
      return (true);
    }
};

list <string> metrics = job.list_metrics ();

while ( metrics.size () )
{
  job.add_callback (metrics.pop_front (), cb);
}
```

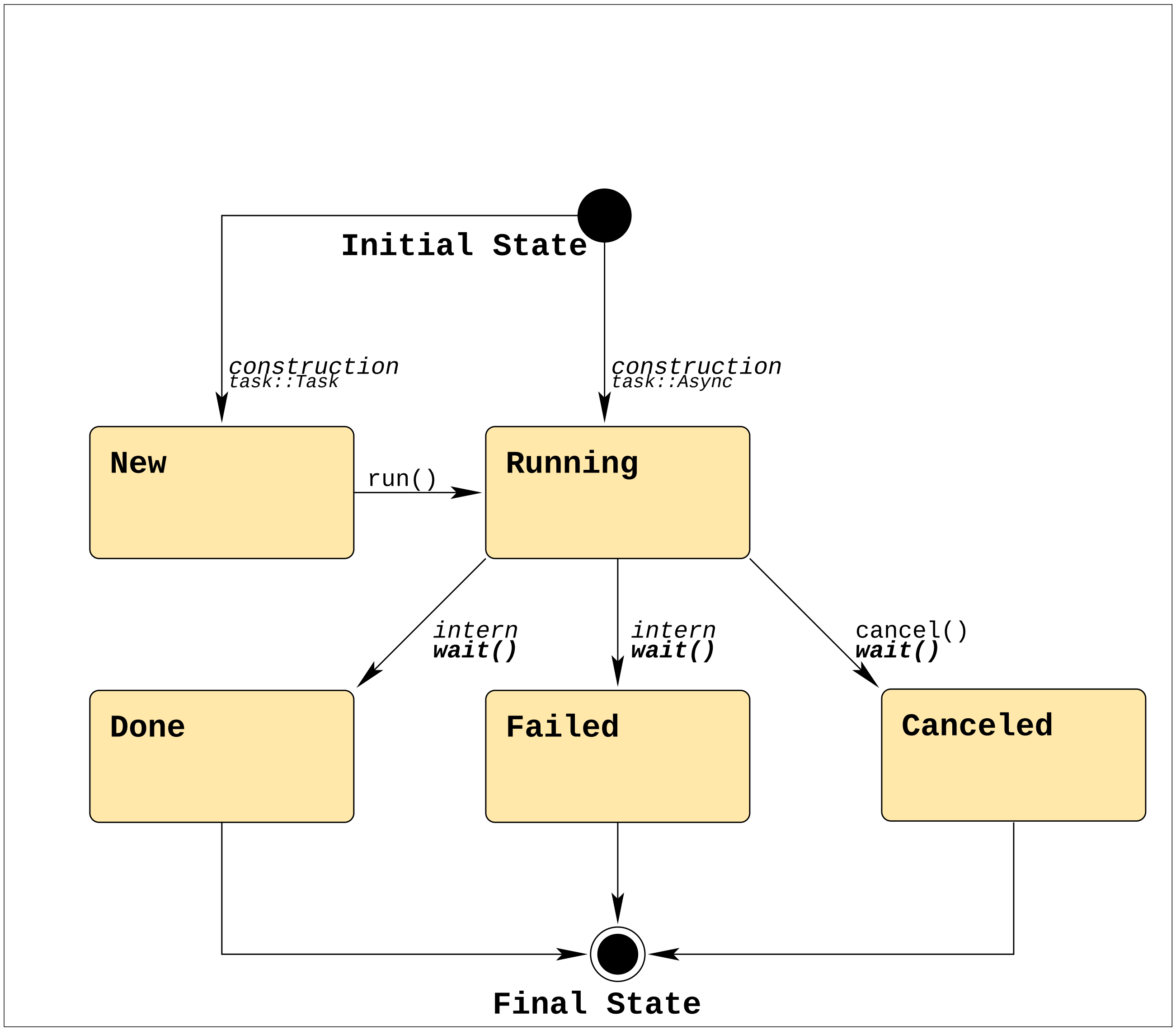
SAGA: Tasks


Look & Feel
Monitoring Model
monitorable
callback
steerable
metric
Error Handling
error_hand.
exception
Attribute Interface
attribute
Base Object
object
Session Handling
session
Security Context
context
Task Model
async
task_cont.
task
job
job_service
job_self
job_desc
Job Management
stream_serv.
stream
Streams
rpc
RPC
ns_entry
file
ns_directory
logical_file
directory
logical_dir.
File Management
Name Space Mngmt.
Replica Management
inherits
implements
interface
class

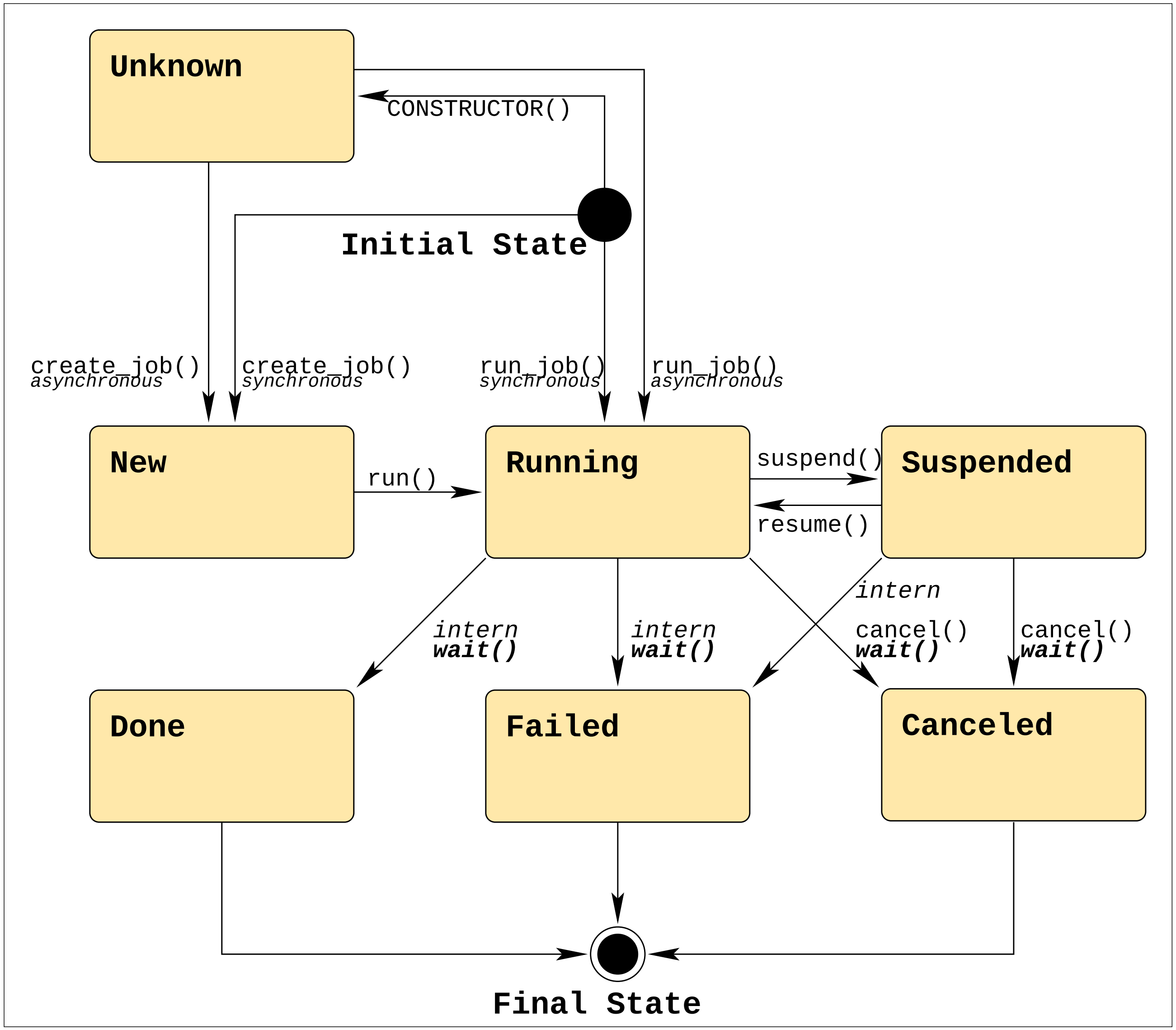
SAGA: Tasks

- 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)

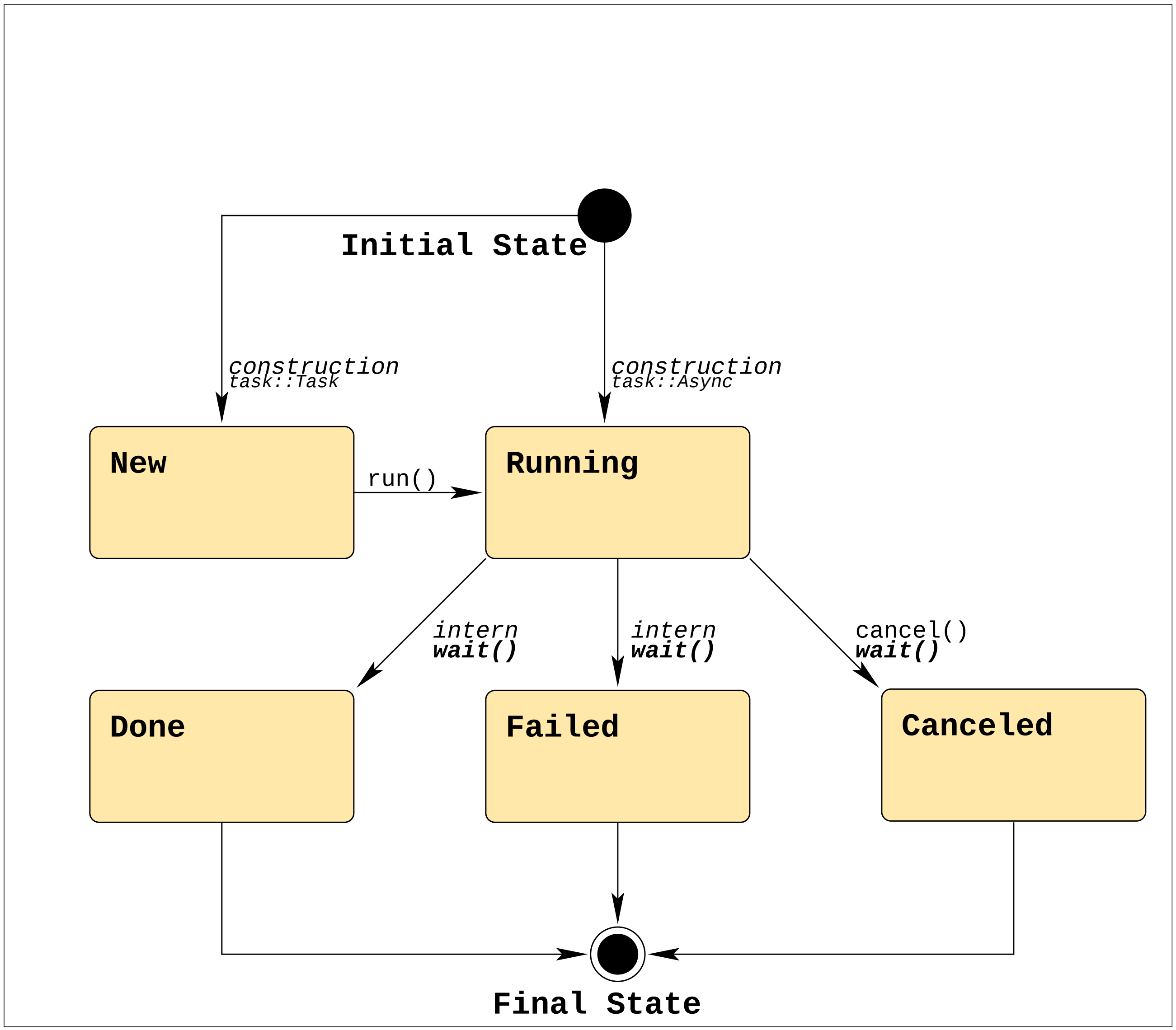
SAGA: Task States



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);
```

Messaging client

```
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

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");
```

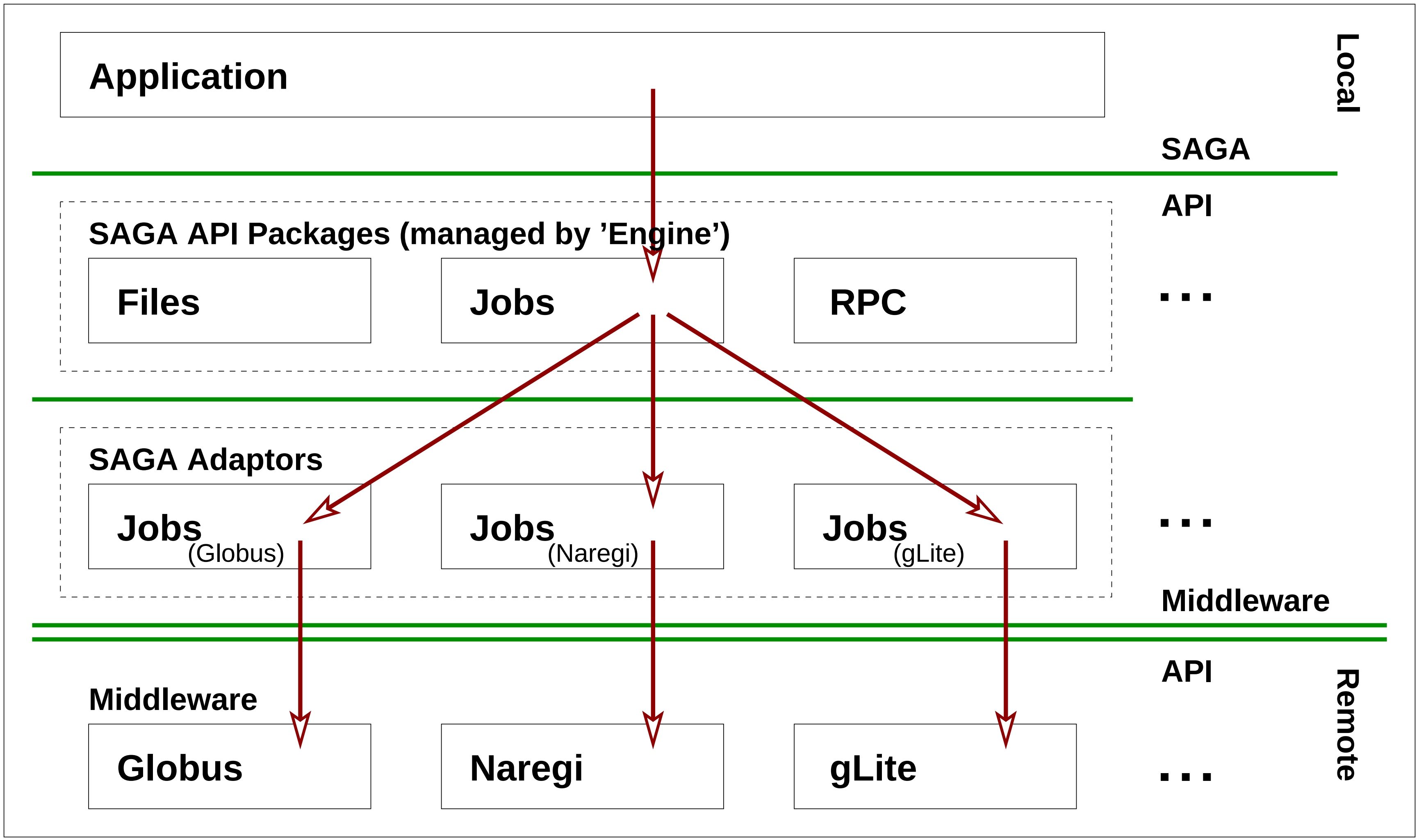
SAGA v2: CPR

- 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`

$\longrightarrow$ wiki, CVS details

Questions?