SAGA – a Simple API for Grid Applications

Introduction to the API

Agenda

- Background Recap

- SAGA API structure and scope

- walkthrough

- some implementation details

- SAGA extensions

Grid APIs and Frameworks

- coding applications for Grids is notoriously difficult:

  - complex infrastructure,
  - diversity of middlewares
  - invironment evolves very quickly

- diversity of Grid Middleware implies diversity of APIs

- difficult to keep up with MW development, and to stay **simple**

- few APIs try to generalize Grid programming concepts

OGF: Summary

- some API standards 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 APIs tend to be more powerful

- top-down APIs tend to be simpler and more concise

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

SAGA

Simple API for Grid Applications

SAGA Design Principles

- **SAGA**: **S**imple **A**PI for **G**rid **A**pplications

- OGF approach to a uniform API layer (facade)

- **governing principle: 80:20 rule**
  simplicity versus control!

- **top-down approach:** use case driven!
  $\rightarrow$ defines **application level** abstractions

- **extensible:** stable look & feel + API packages

- **influenced by:** DRMAA, GridRPC, OREP, JSDL,
  POSIX, GAT, CoG, LSF, Globus, . . .

- API Specification is **Language Independent** (IDL)
  Renderings exist in C++, Python, Java
  Examples here are in C++

SAGA Intro: Example 1

```
                        —— SAGA: File Management ——
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
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 Intro: Example 2'

```
                         SAGA: Job Submission

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;
```
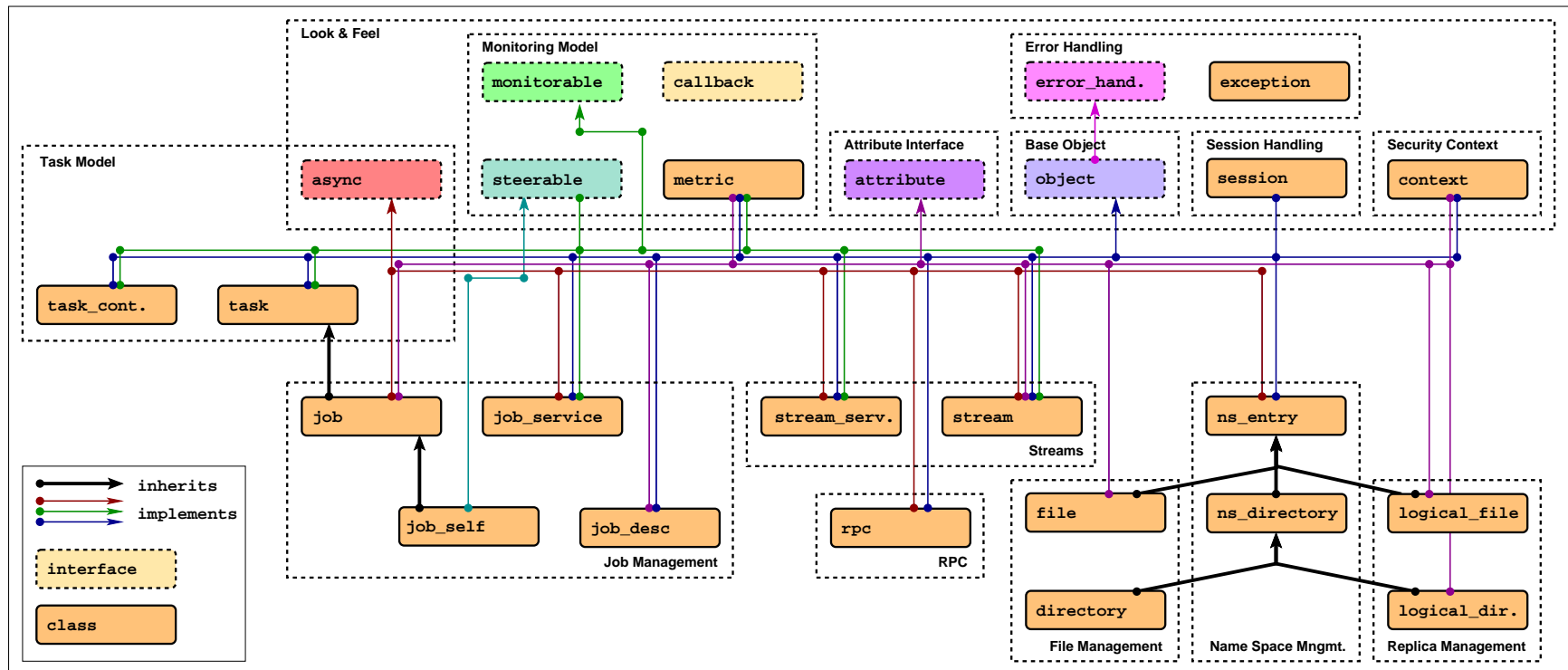
SAGA Intro: Example 2

- stateful objects!

- yet another job description langauge? `:-(`

- many hidden/default parameters (to keep call signatures small)

- `'any://'` again!

- TIMTOWTDI (there is more than one way to do it)
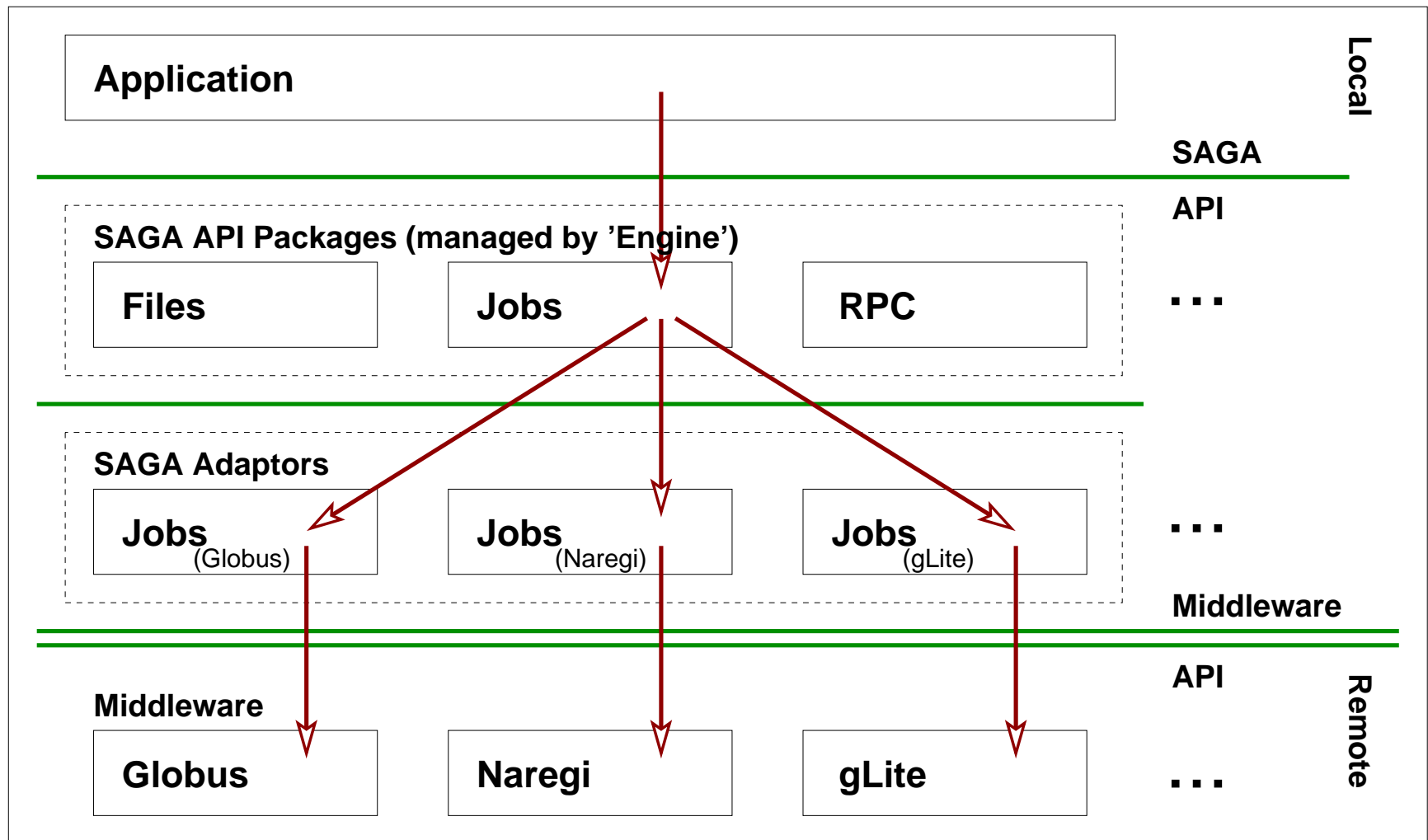
SAGA Intro: 10.000 feet

- **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
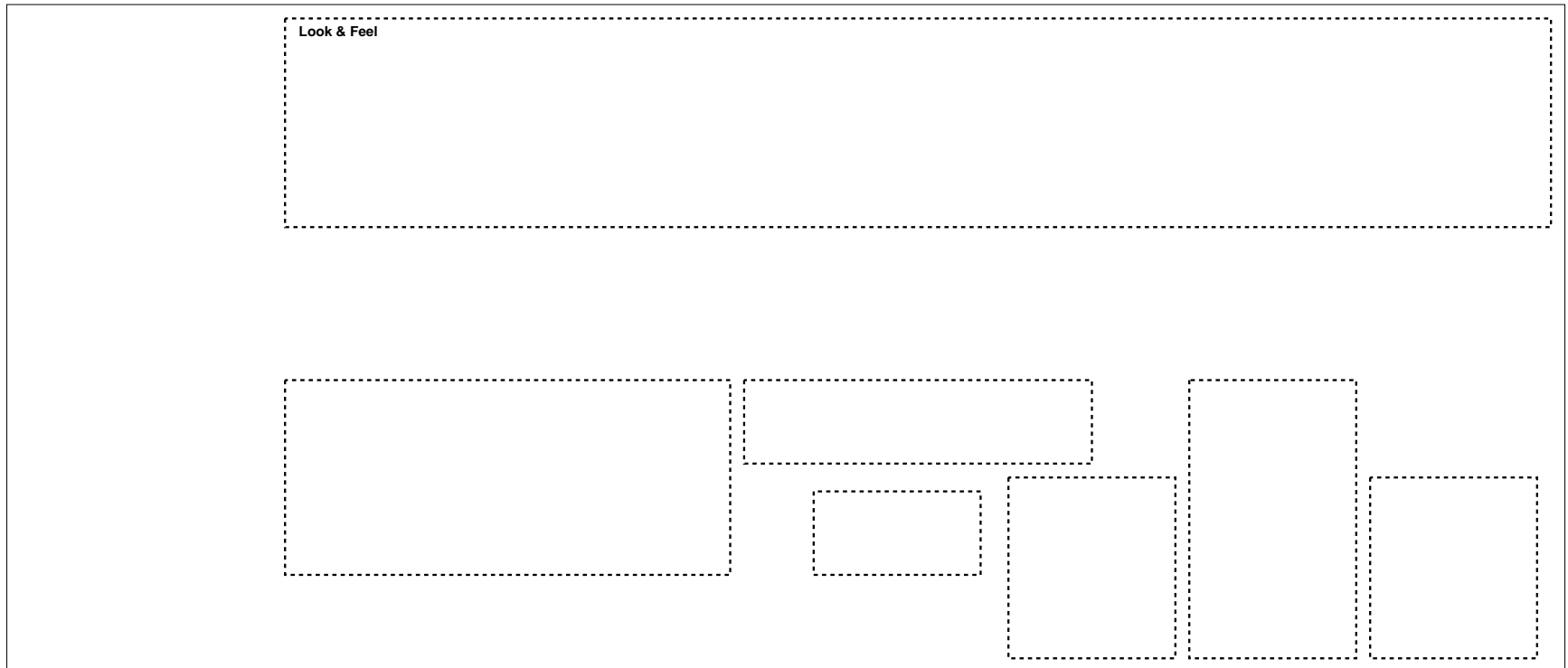 implementations

SAGA: Class hierarchy

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

Implementation

Local

Application

SAGA

API

SAGA API Packages (managed by 'Engine')

| Files | Jobs | RPC | . . . |

SAGA Adaptors

| Jobs (Globus) | Jobs (Naregi) | Jobs (gLite) | . . . |

Middleware

API

Remote

Middleware

| Globus | Naregi | gLite | . . . |

SAGA: Class hierarchy

Look & Feel

SAGA: Class hierarchy

SAGA Look & Feel:

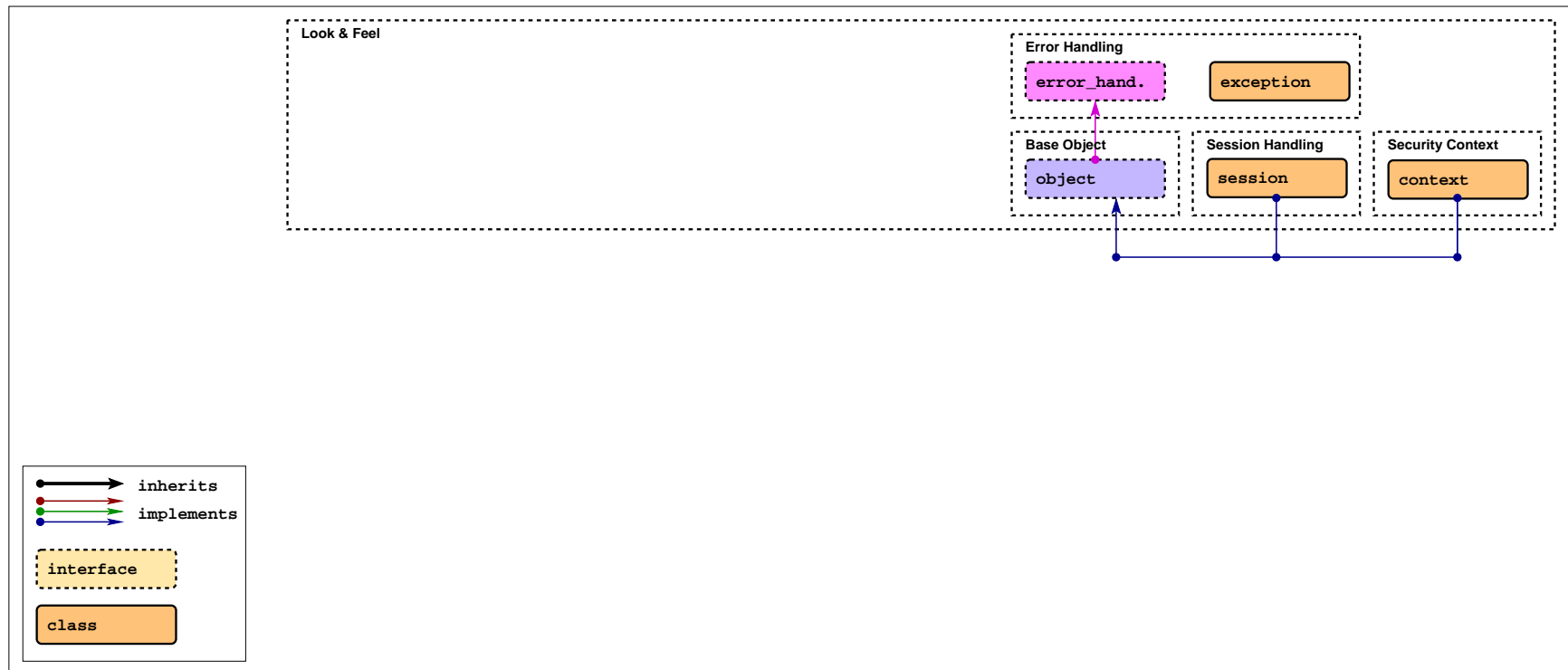`saga::object` allows for object uuids, `clone()` etc.

SAGA: Class hierarchy
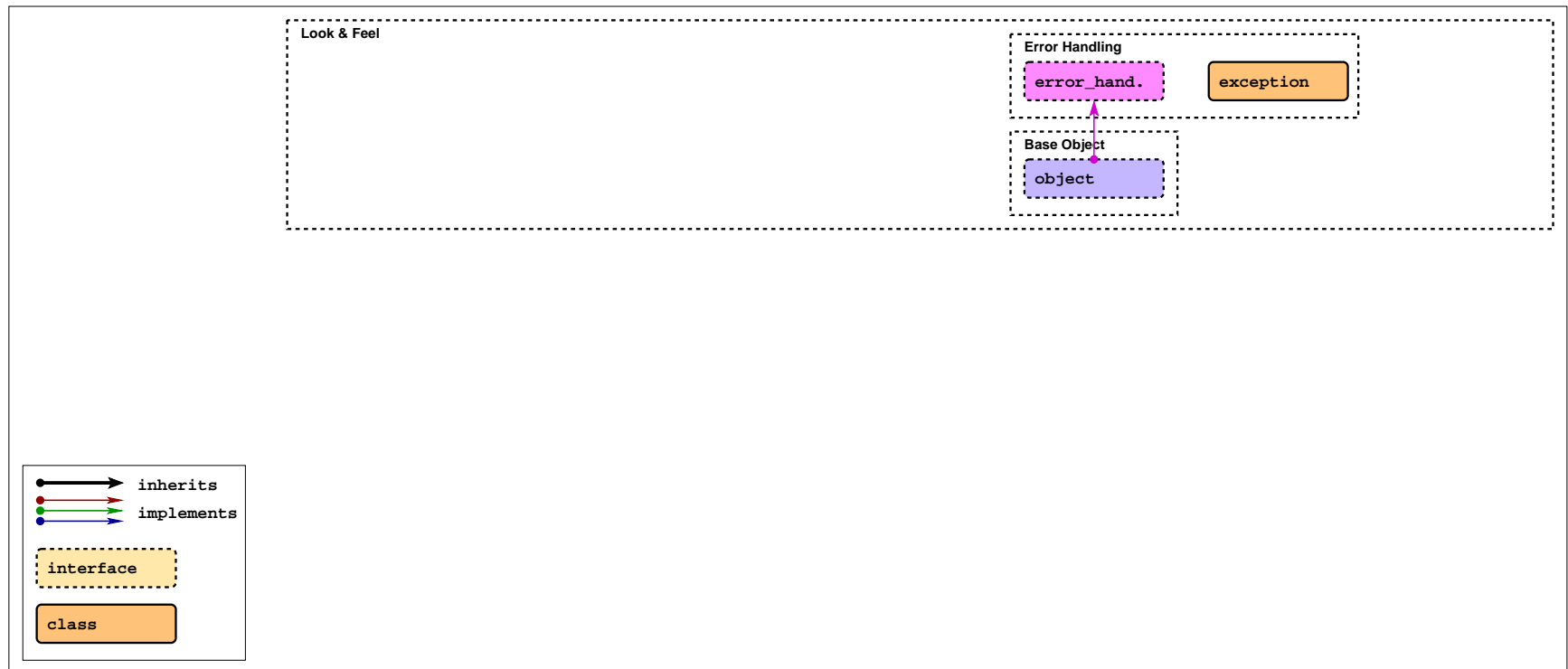
SAGA Look & Feel:

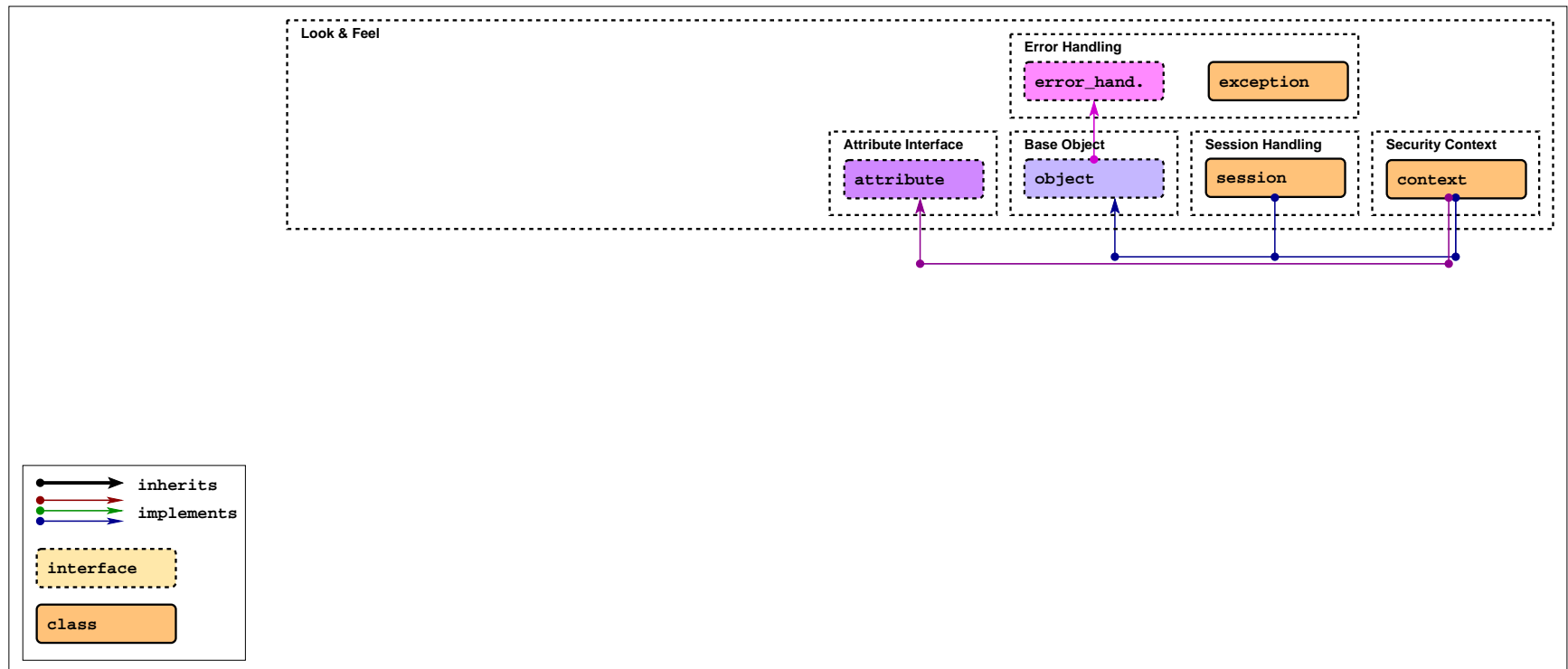errors are based on exceptions or error codes.

SAGA: Class hierarchy

SAGA Look & Feel:

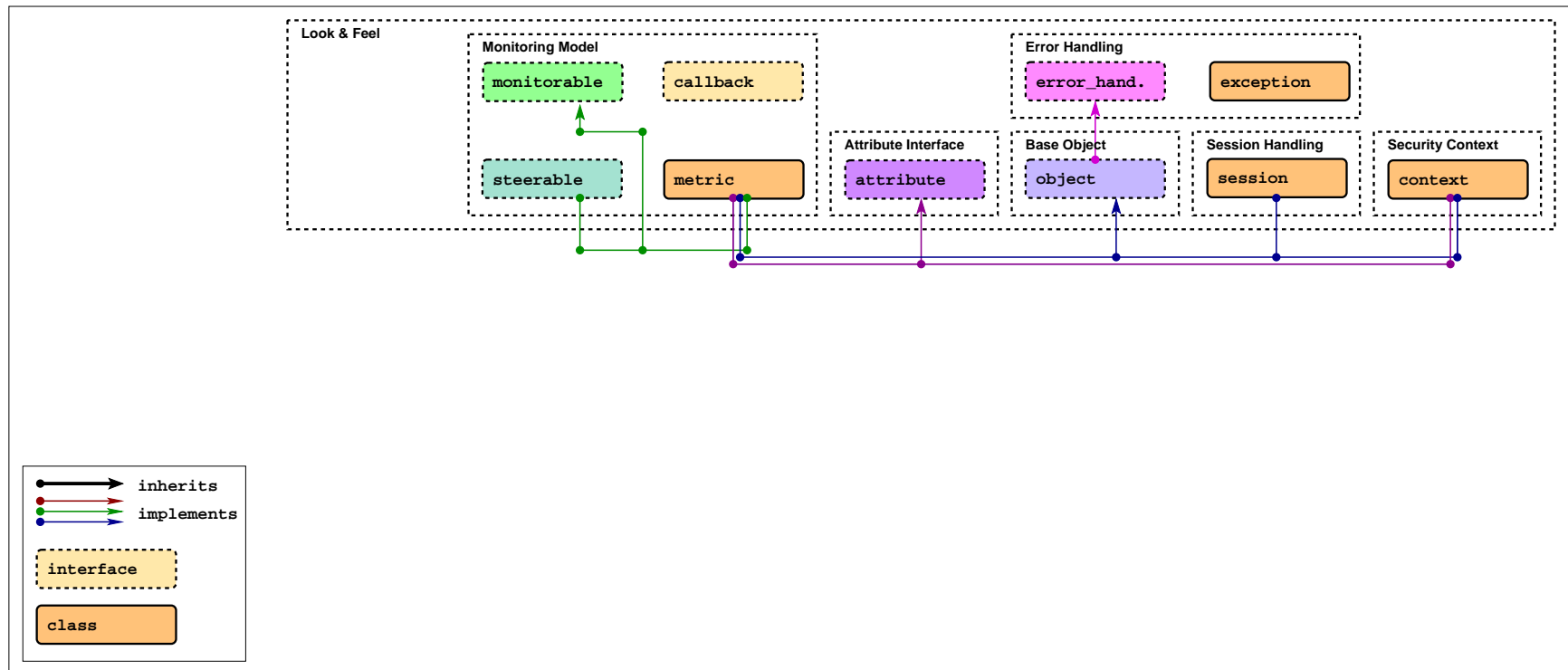session and credential management is hidden.

SAGA: Class hierarchy

SAGA Look & Feel:
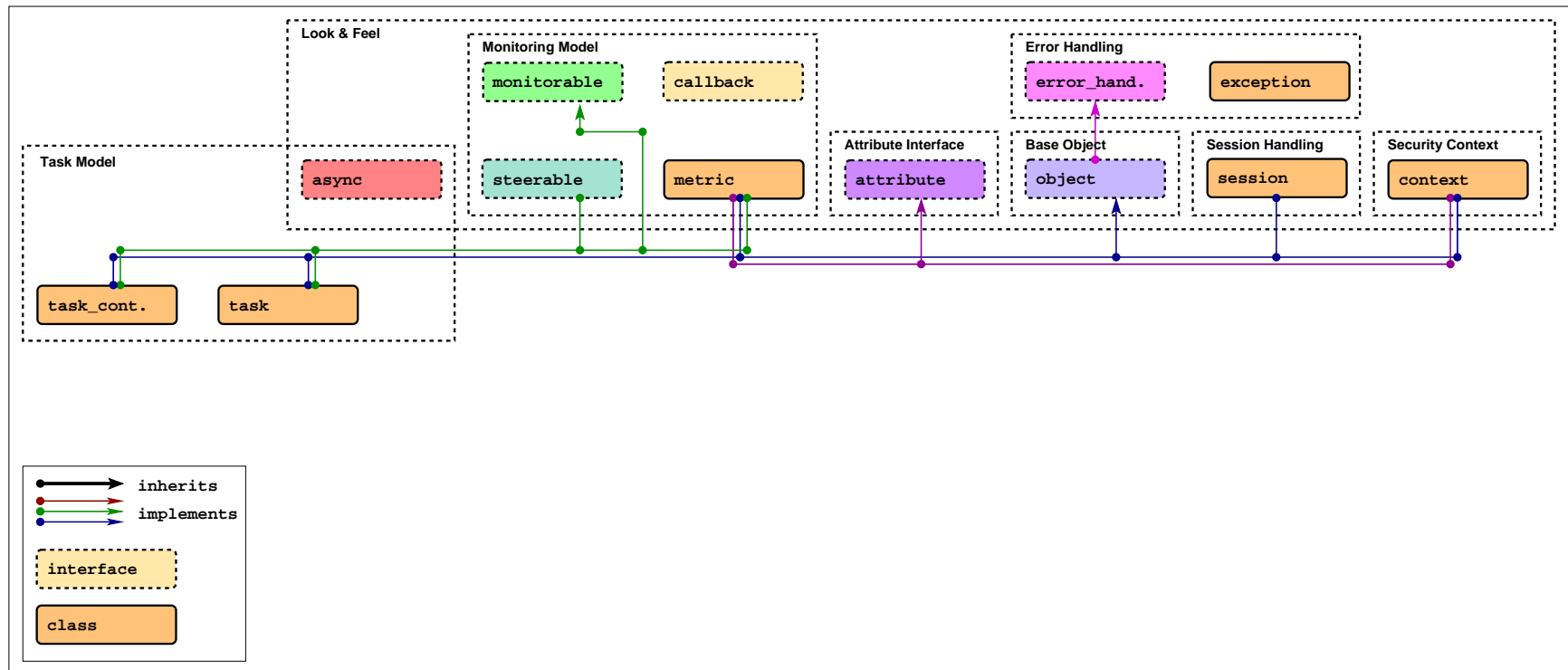
Attribute interface for meta data.

SAGA: Class hierarchy

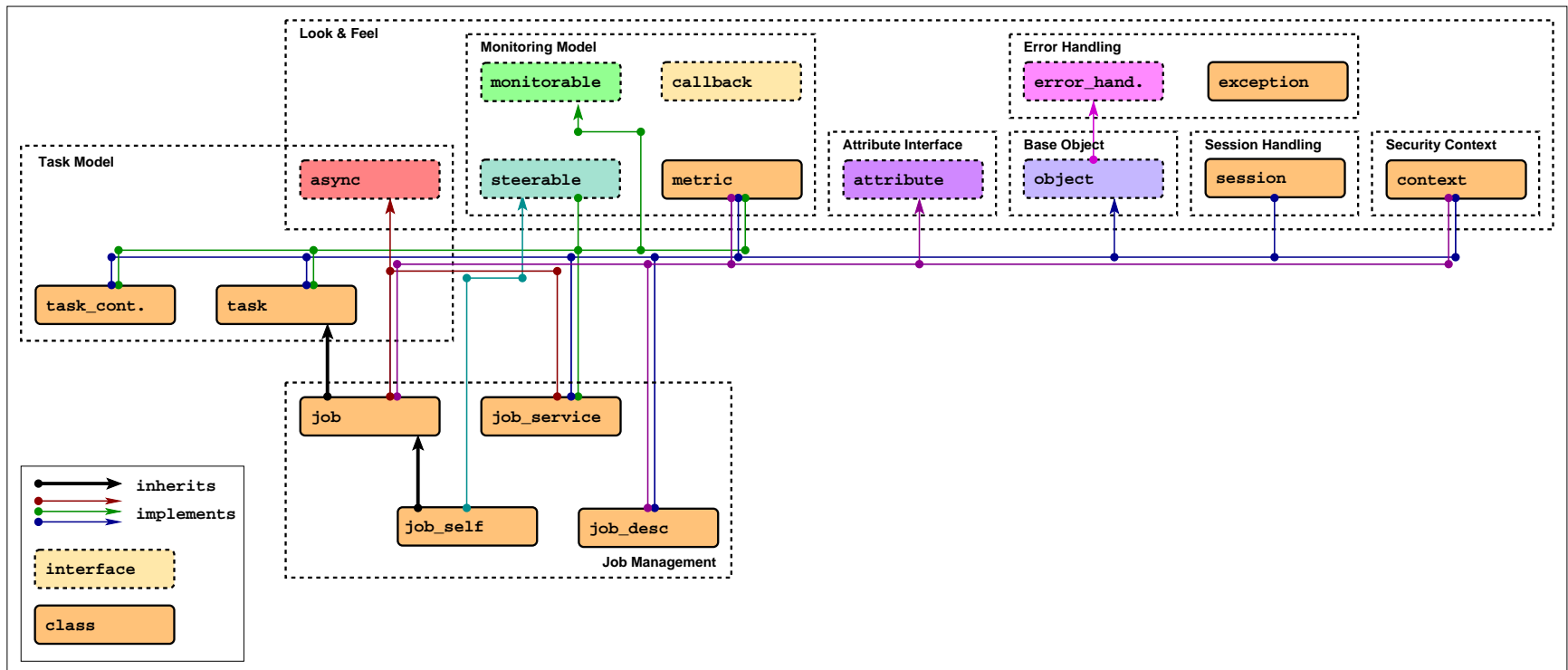## SAGA Look & Feel:

Monitoring includes asynchronous notifications.

SAGA: Class hierarchy

SAGA Look & Feel:

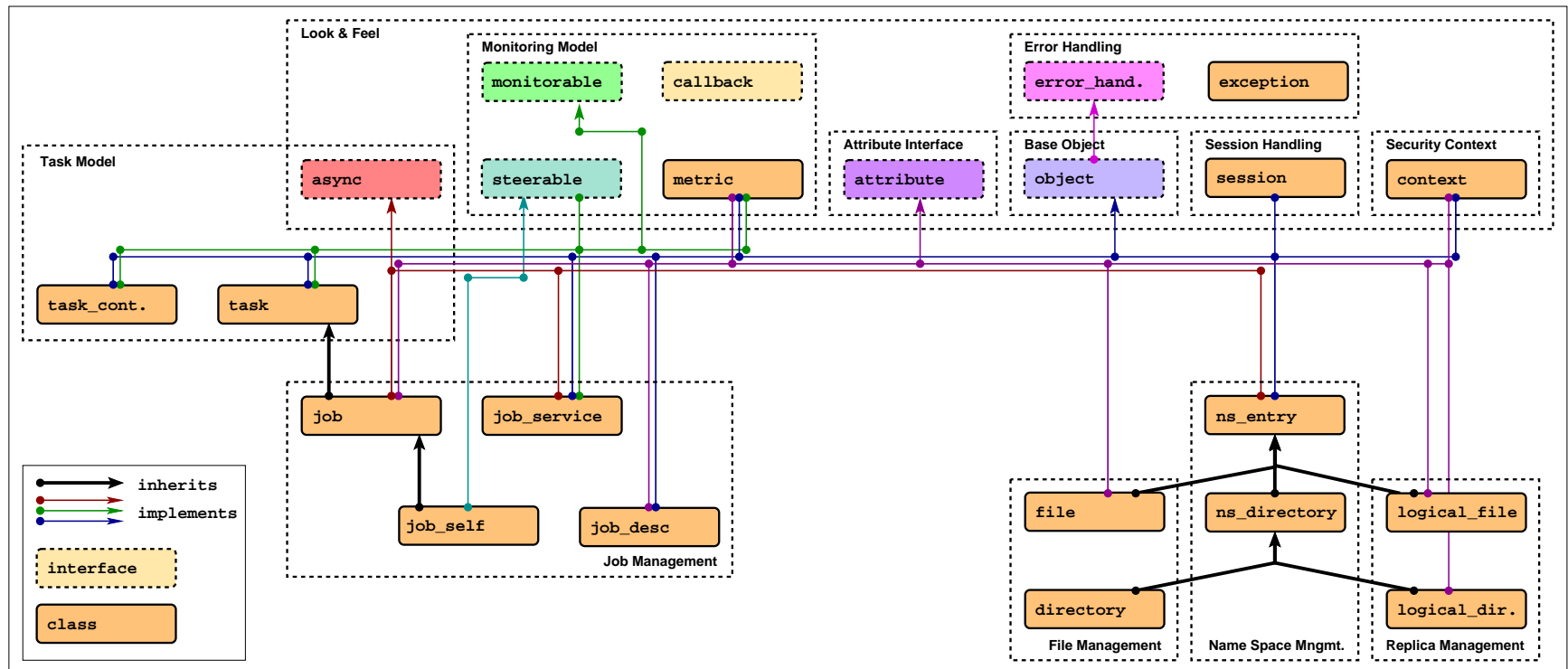the task model adds asynchronous operations.

SAGA: Class hierarchy

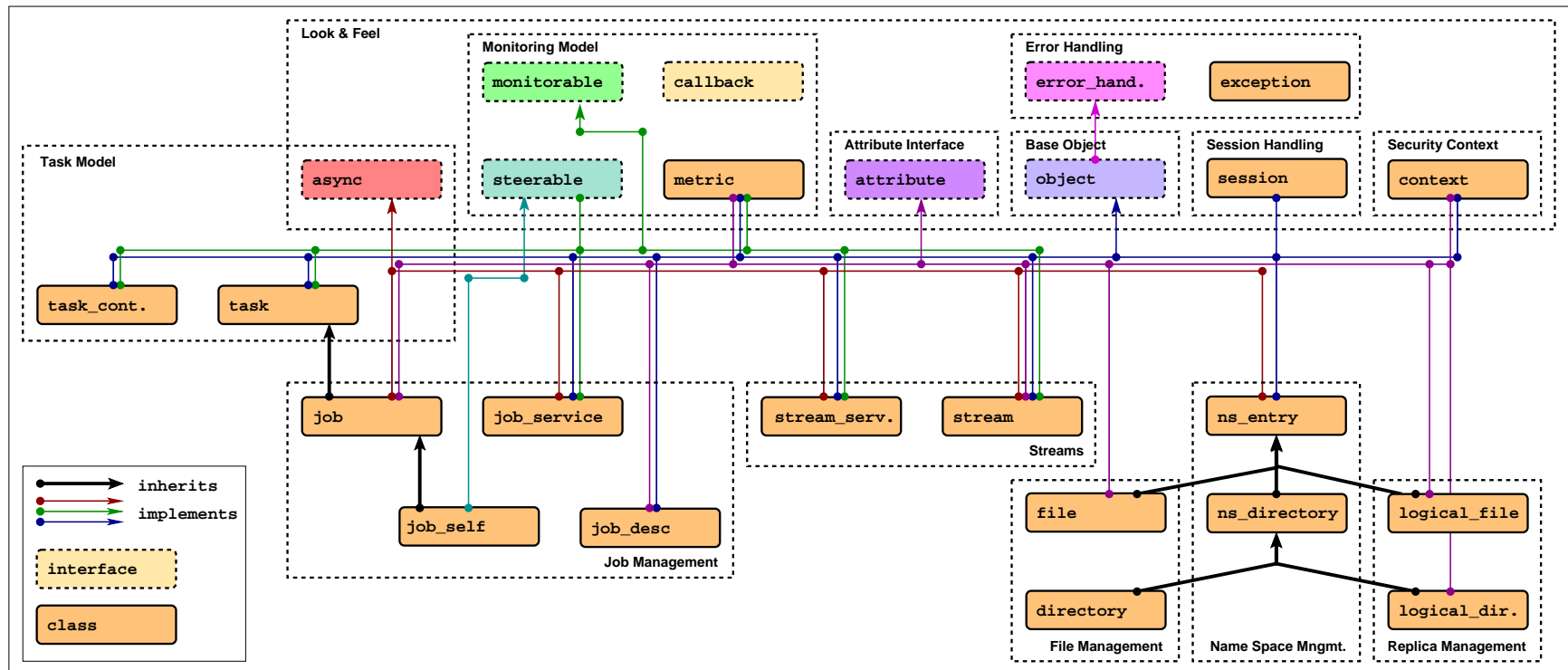## SAGA API Package 'job':

create and manage remote processes.

SAGA: Class hierarchy

SAGA API Package 'name_spaces':

manage files, replicas, etc.

SAGA: Class hierarchy

SAGA API Package 'stream':

SAGA rendering of BSD streams.

SAGA: Class hierarchy
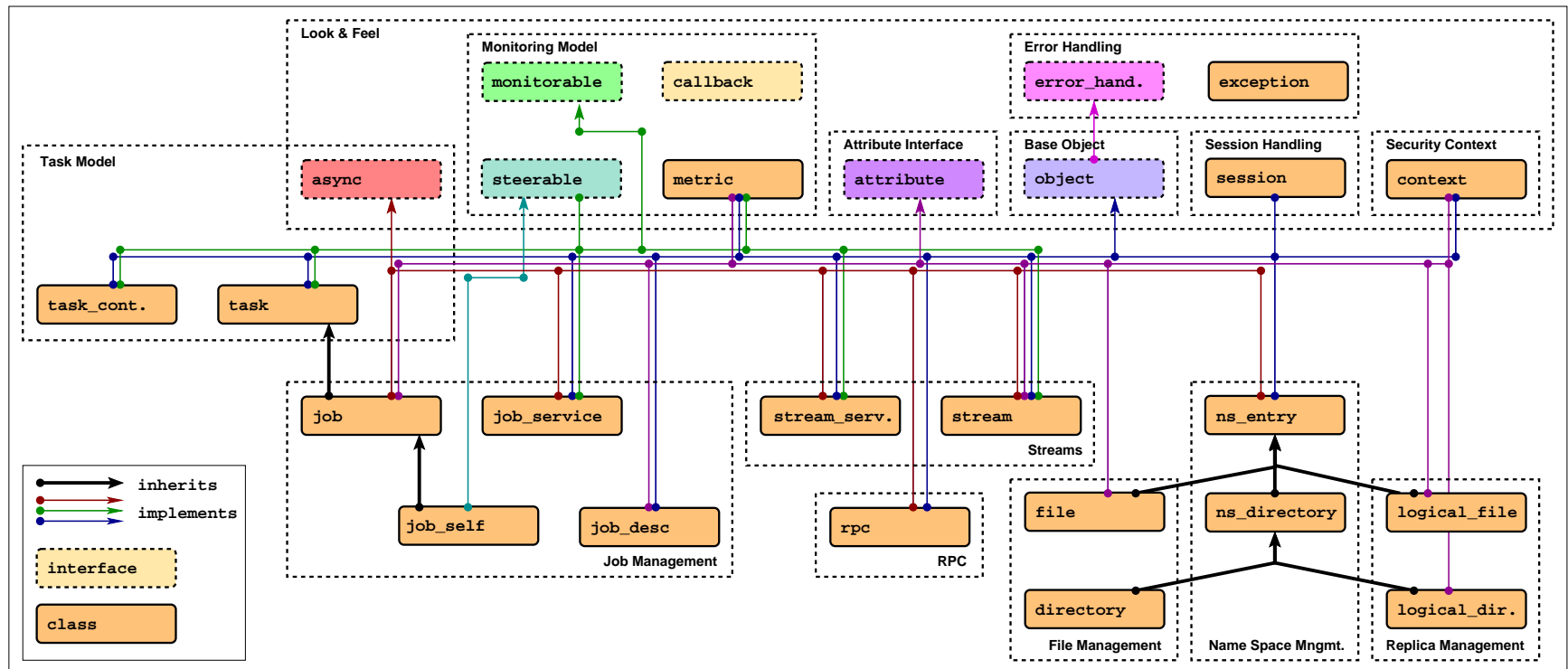
SAGA API Package 'rpc':

remote procedure calls.

Functional API Packages

SAGA: Jobs
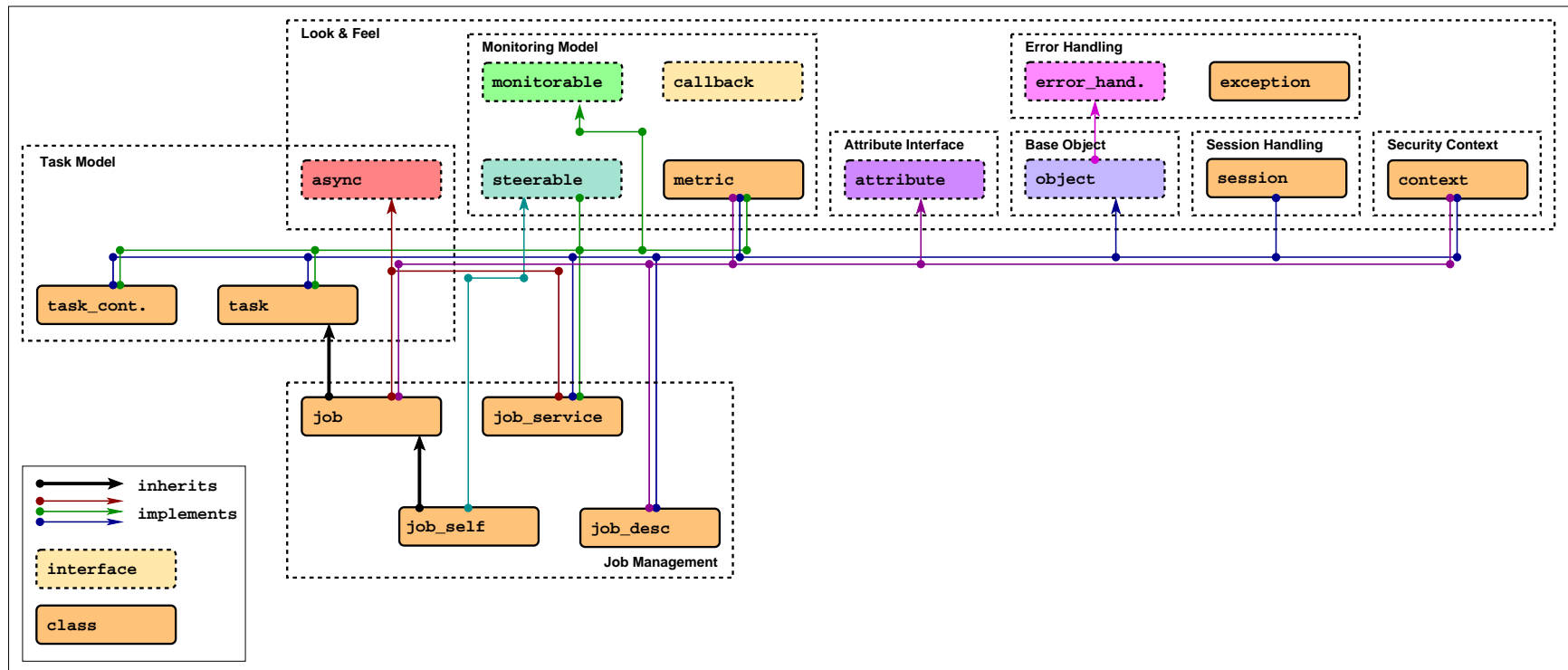
SAGA: Jobs - Overview

- `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
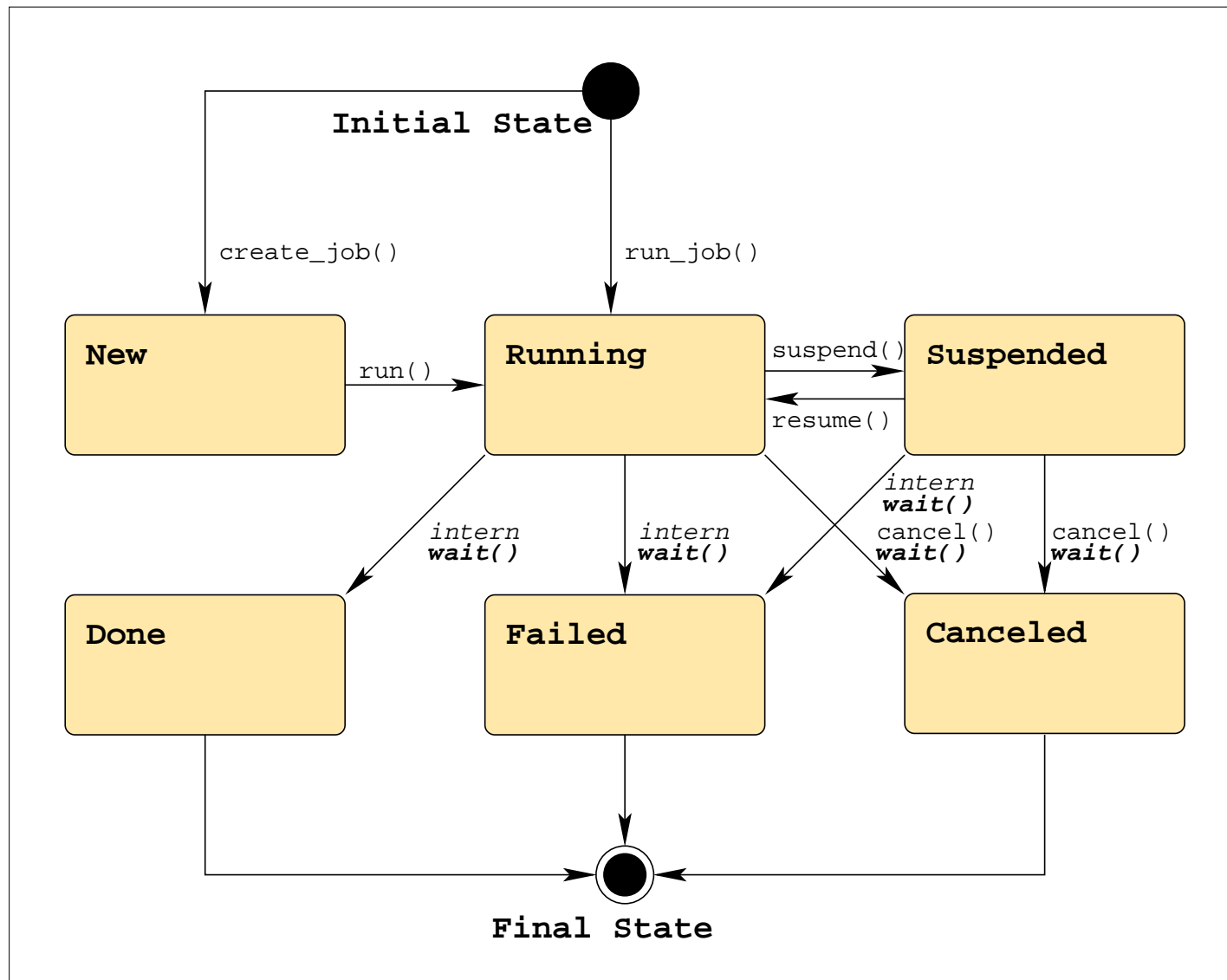
SAGA Examples: Jobs

```
────────────────── job submission ──────────────────

saga::job::service    js ("gram://headnode.gram.net");
saga::job::job        j = js.run_job ("/bin/sleep 10",
                                   "clusternode-2.gram.net");


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


j.wait ();


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

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_attribute ("ExitCode") << endl;
```

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");


// 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 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;
}
```

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: Name Spaces etc.
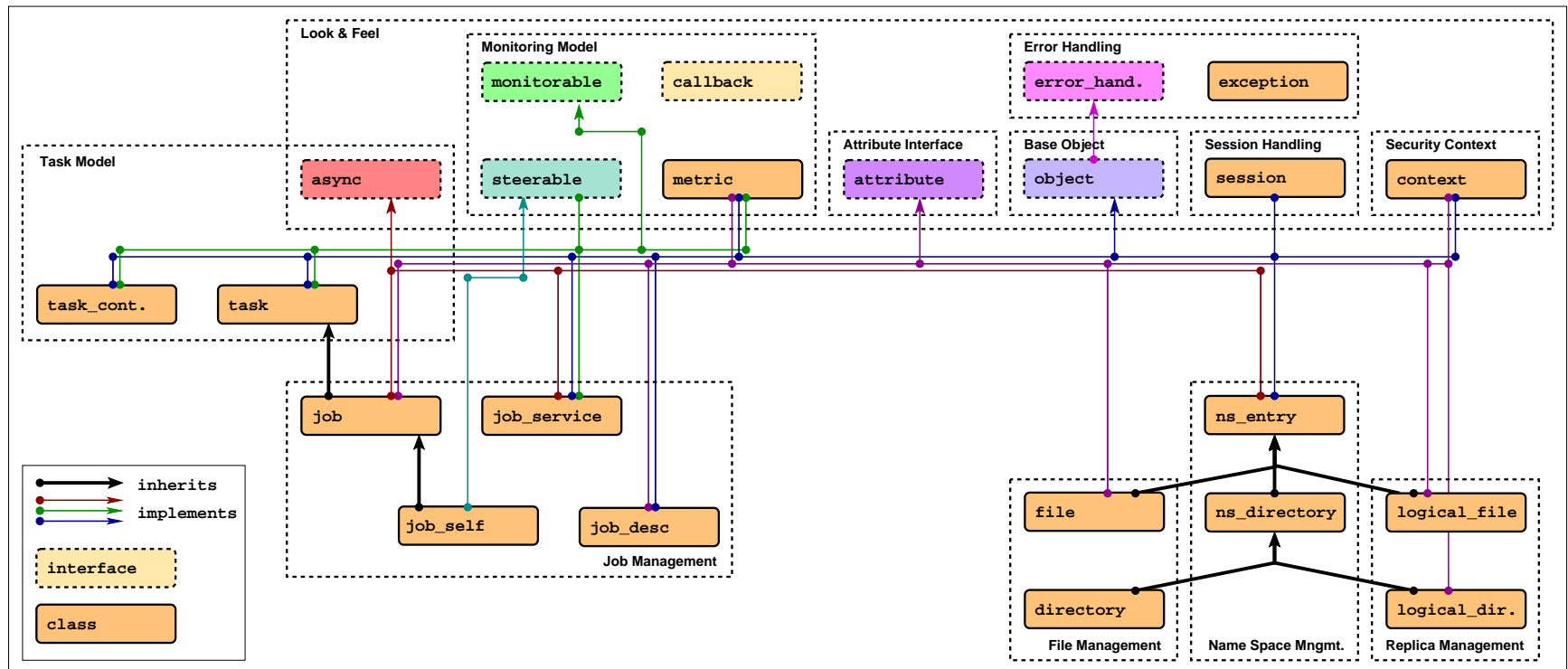
SAGA: Name Spaces

- 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: Adverts

- implements name space interface, and adds access to arbitrary key/value pairs on each entry.

- entries also allow to store a **serialized** SAGA Object!

- utterly useful for application bootstrapping, communication between different application modules, application persistency, etc etc.

- **not part of the SAGA Specification**, but is getting standardized as an extension

```
                    ─── name space management ───

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 Name Spaces

- 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 (c'tor, open), copy, link, move, remove`

- **permissions:**
  `permissions_allow, permissions_deny`

- wildcards are supported (remember POSIX influence...)

SAGA Examples: Files

```
                           file access
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 Filesystem

- 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 by the engine

- several incarnations of read/write: posix style, scattered, pattern based

SAGA Name Spaces: Flags

```
┌──────────────────── 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        = 204    // only on filesystem │
│ }                                              │
└────────────────────────────────────────────────┘
```

SAGA Examples: Replicas

```
                   ── replica management ──
saga::replica::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::replica::file file = dir.open ("tmp/data.txt");
list <string>  locations = file.list_locations ();


file.replicate ("gridftp://other.host.net/tmp/a.dat");
```

SAGA Replica

- provides access to the **content** of replica system entries (list of physical locations, plus attributes)

- saga attribute interface is used for entry meta data. Meta data are maintained by application, and/or backend.

- `replicate()` creates a new copy, and adds new location to list

SAGA Examples: Replicas

```
───────────────── replica meta data ─────────────────

saga::replica::directory dir ("raptor://remote.host.net/data/");


list <saga::url> files = dir.find ("*", "type=jpg");


while ( file.size () )
{
  saga::logical_file lf (file.pop_front ());

  lf.replicate ("file://localhost/data/images/",
                saga::replica::Overwrite);
}
```

SAGA Adverts

- persistent storage of application level information

- semantics of information defined by application

- allows storage of serialized SAGA objects (object persistency)

SAGA Examples: Adverts

```
                              ── 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 ("description") << std::endl;
}
```
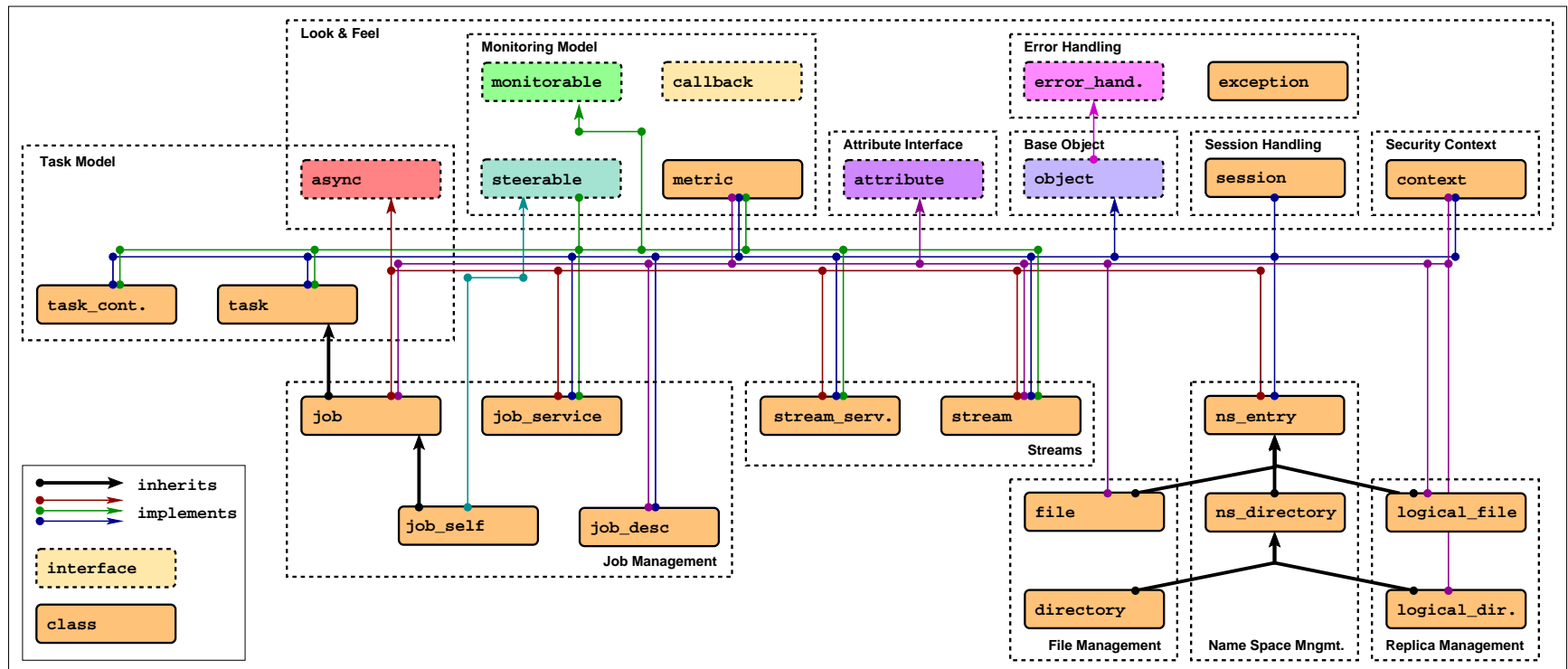
SAGA Examples: Adverts

```
                        Persistent SAGA Objects

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


ad.store_object (f);


------------------------------------------------------------------


saga::advert ad ("any//remote.host.net/files/my_file_ad");
saga::file   f = ad.retrieve_object ();


```

SAGA: Streams

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: Streams

- simple and BSD socket oriented

- not supposed to replace MPI etc, but allows for simple application level communication

- will be superceded by message package

SAGA: RPC
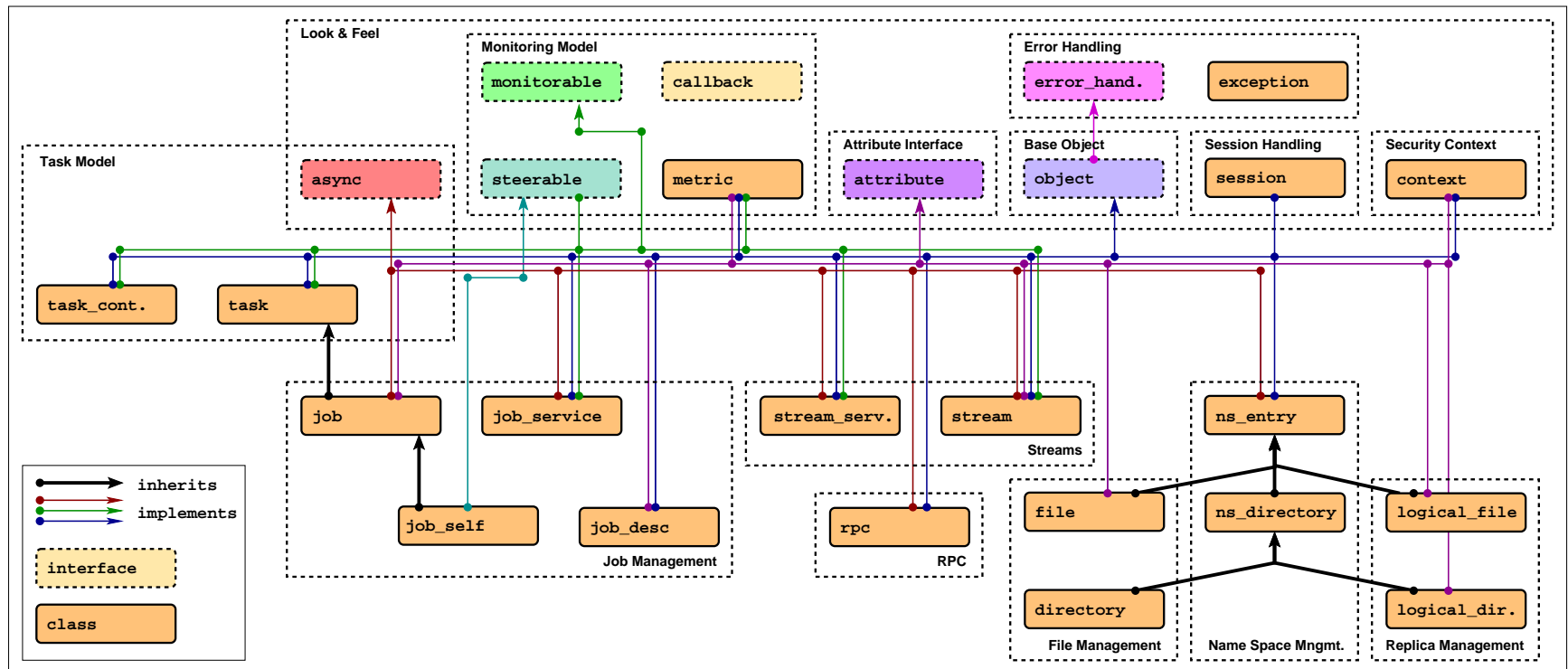
SAGA Examples: RPC

```
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 ());
```
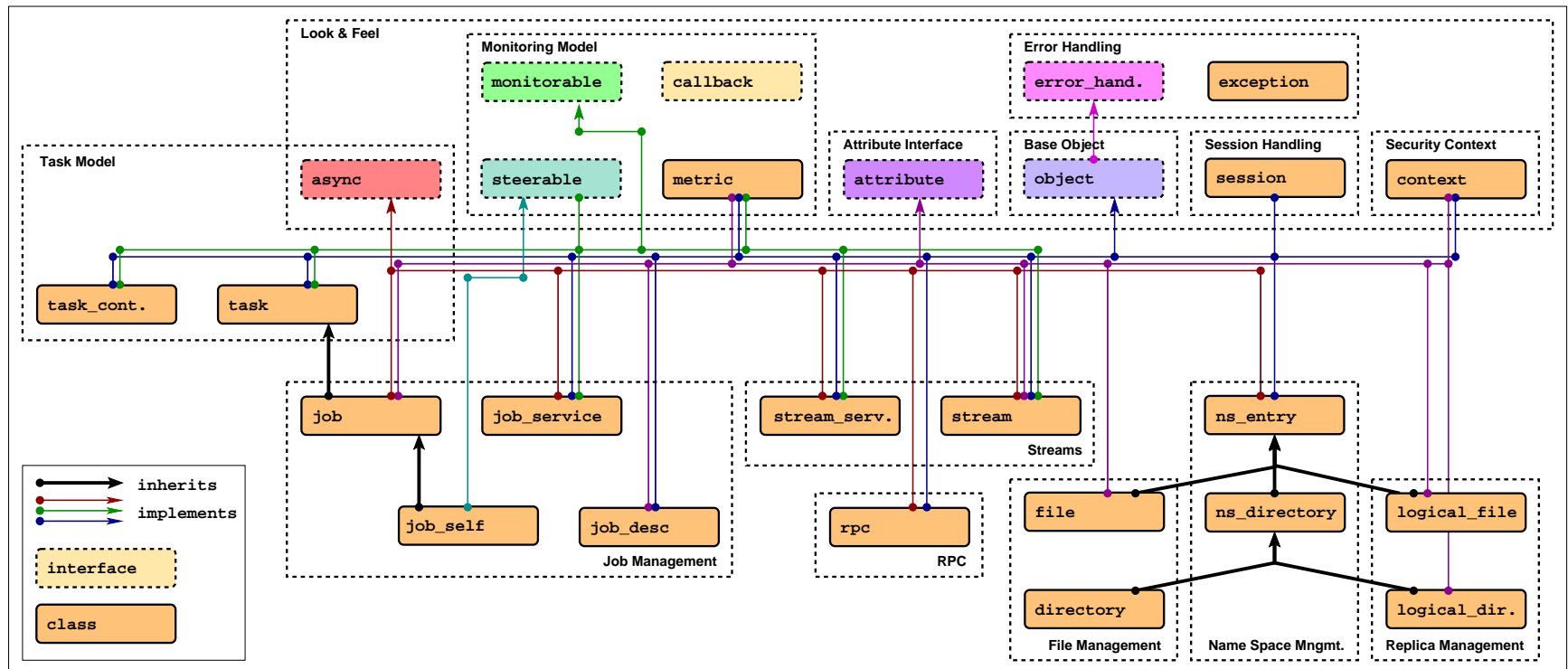
SAGA: RPC

- 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

Non-Functional API Packages

SAGA: Session and Context

SAGA Examples: Session

```
                          ── default sessions ──

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

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

SAGA Examples: Session

```
                        ─ context management ─

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 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

```
                      ── 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!
```
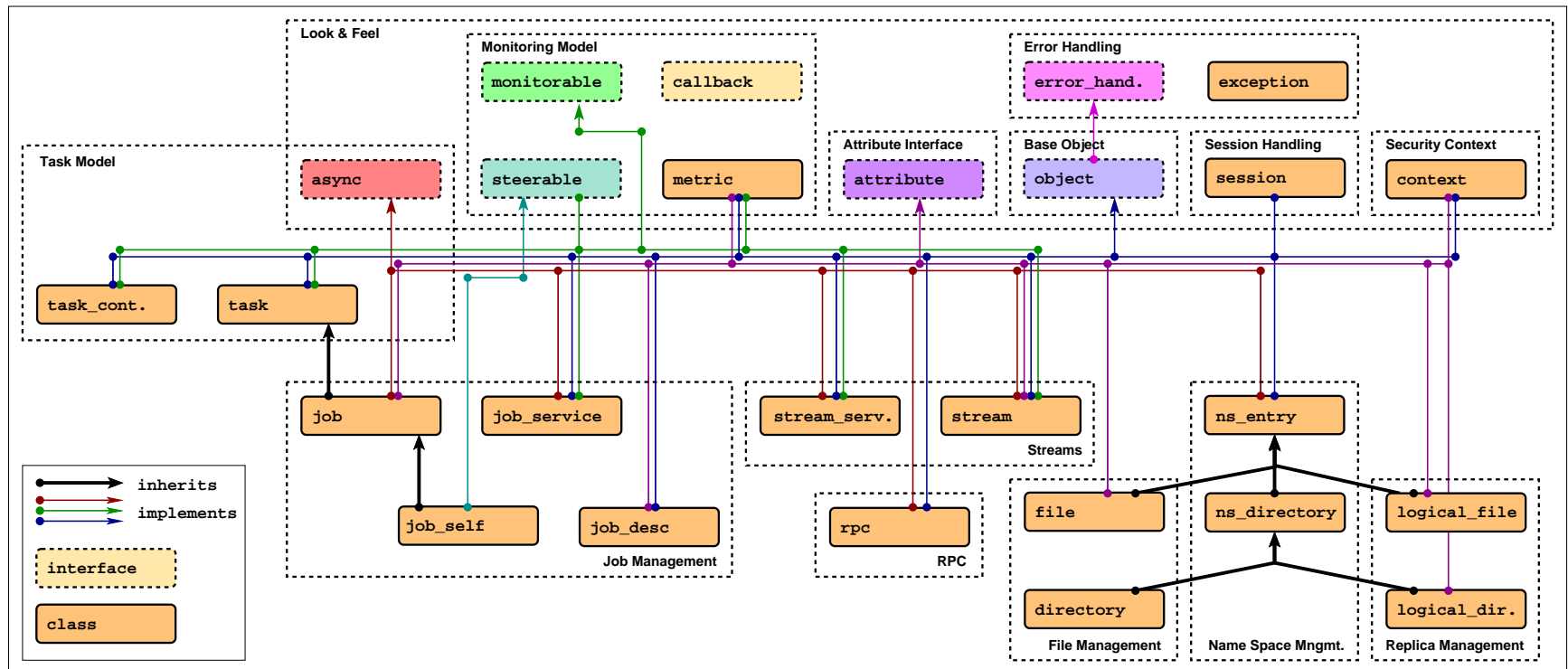
SAGA Examples: Contexts

```
                        ━━ 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 j = js.create_job (jd);

j.run ();

saga::metric m = j.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 j = js.create_job (jd);
j.run ();


saga::metric m = j.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 j = js.create_job (jd);
j.run ();



j.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 = j.list_metrics ();


while ( metrics.size () )
{
  j.add_callback (metrics.pop_front (), cb);
}
```
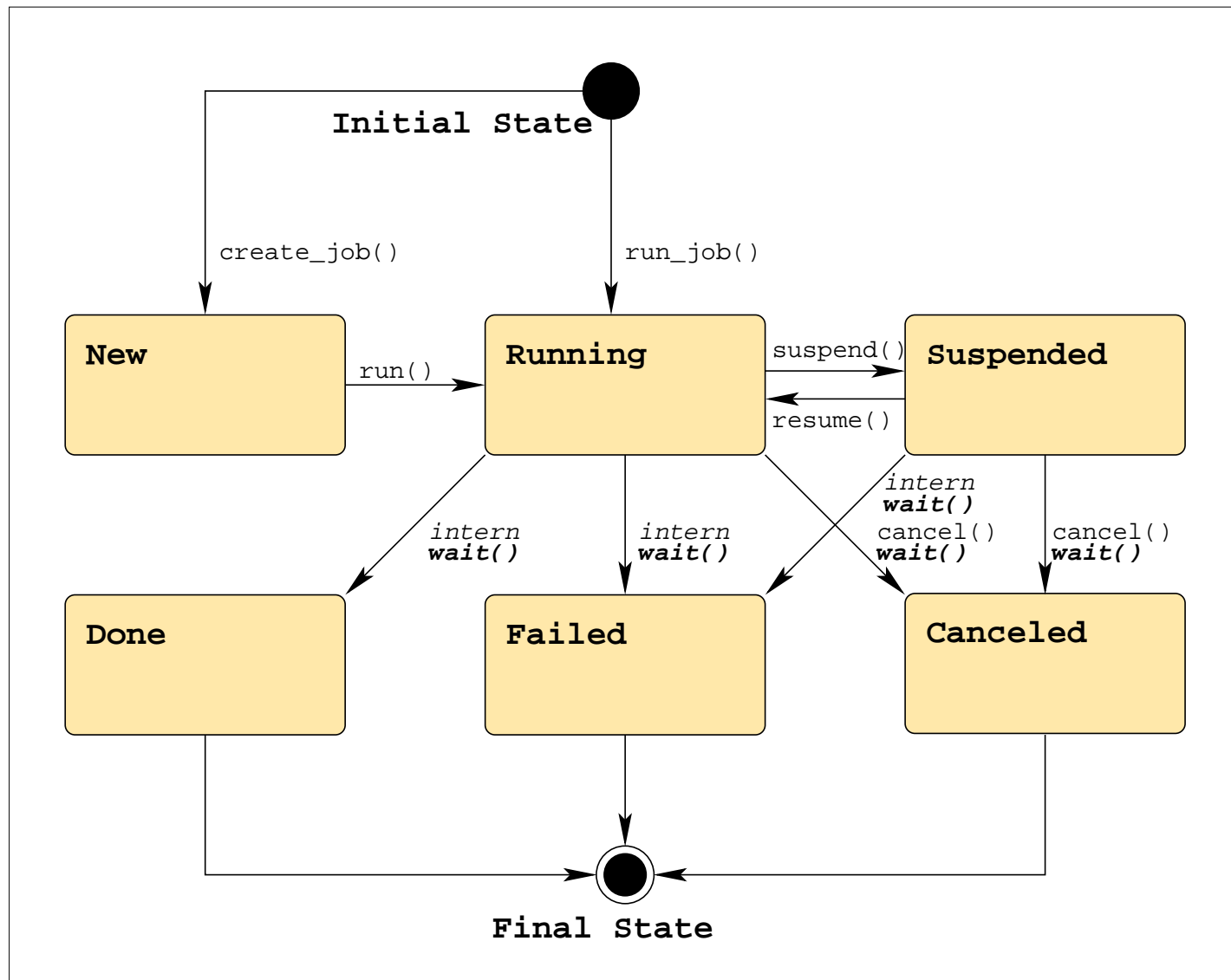
SAGA: Tasks
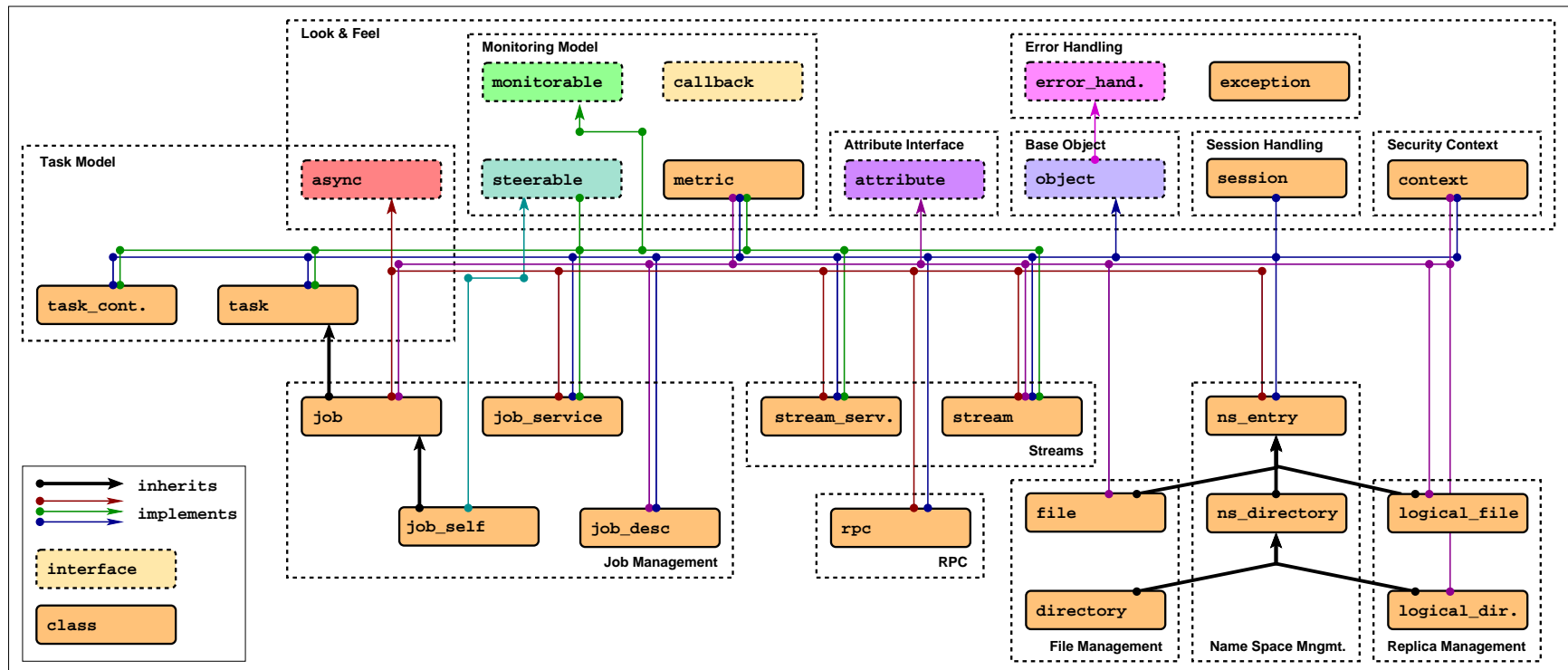
SAGA: Tasks

- 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
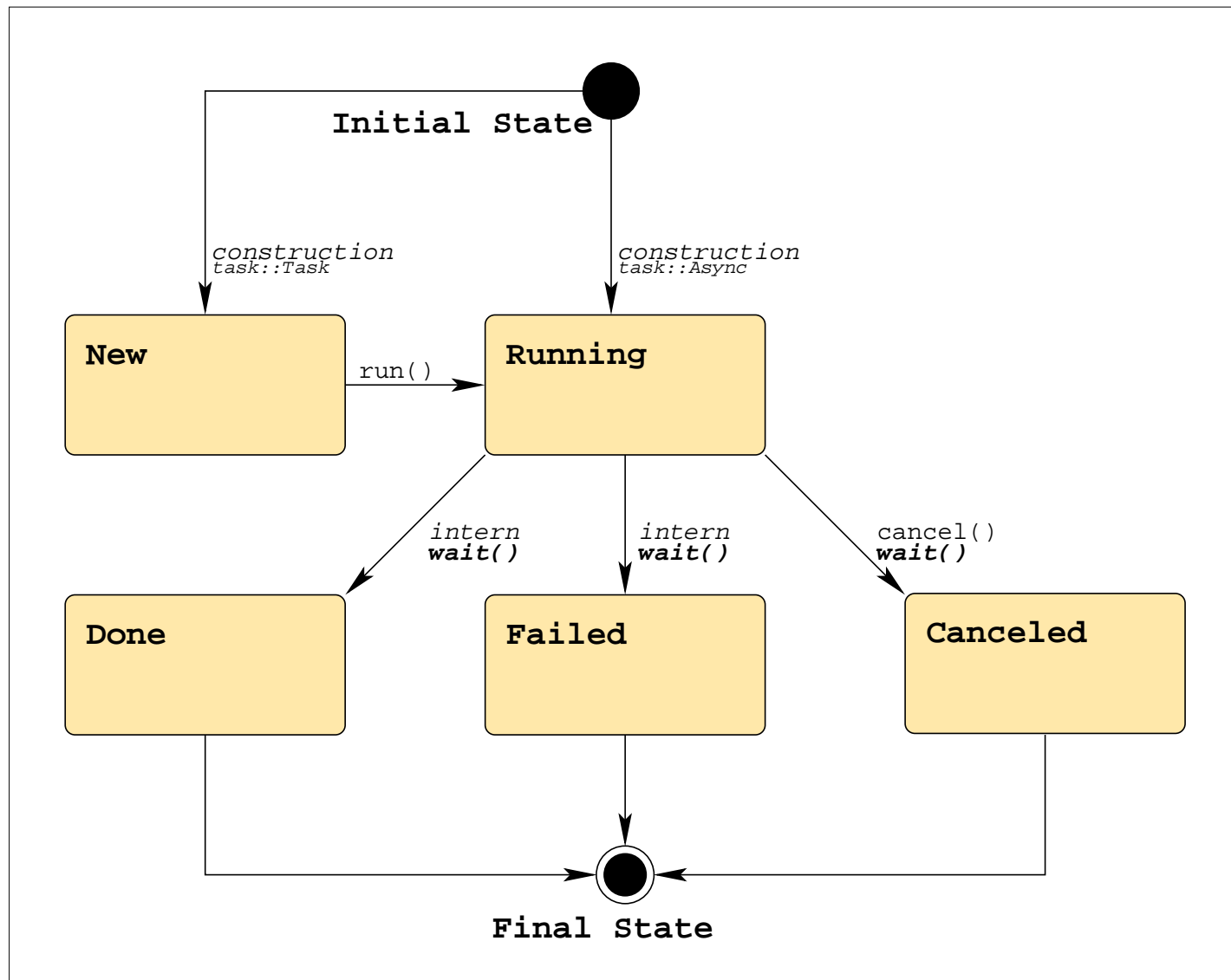
- tasks are stateful (similar to jobs)

SAGA: Task States
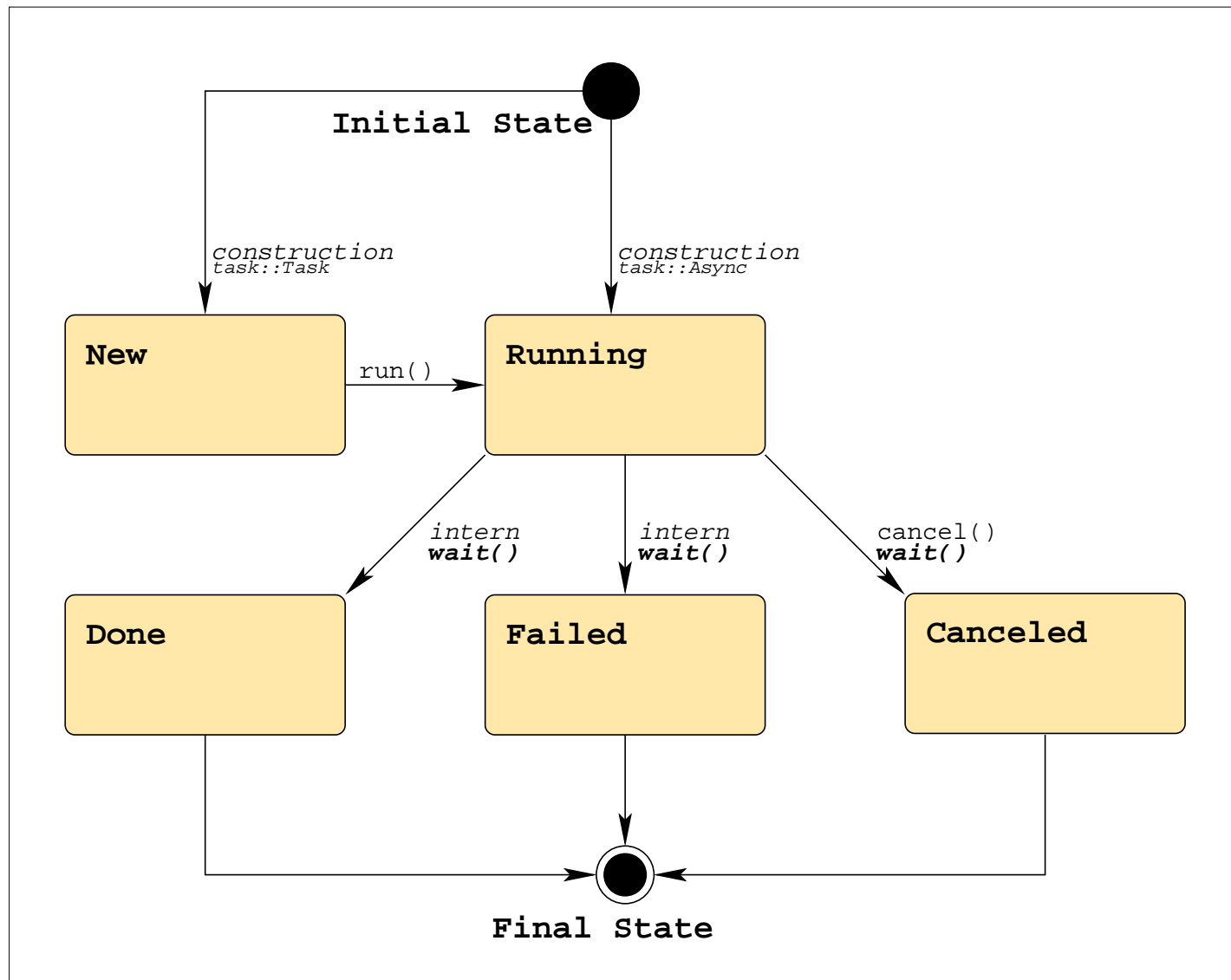
SAGA: Job States

SAGA: Task States

SAGA: Tasks

- different versions for each method call: sync, async, task

- signature basically the same

- differ in state of task returned by that method

SAGA Examples: Tasks

```
                         ─── tasks (i) ───

saga::file file ("gsiftp://remote.host.net/data/data.bin");


// normal, synchronous
file.copy ("data.bak");   // void


// 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
```

SAGA Examples: Tasks

```
                           ──── tasks (ii) ────

saga::file file ("gsiftp://remote.host.net/data/data.bin");


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


// async versions
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 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 Examples: Tasks

```
                        ── tasks (iii) ──
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 and jobs ───

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

- service discovery

- message based communication

- information service (Advert Service)

- resource discovery and management

- checkpoint & recovery (GridCPR)

Questions?