CENTER FOR COMPUTATION
& TECHNOLOGY

SAGA

A Simple API for Grid Applications

Introduction to the SAGA API

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

SAGA

A Simple API for Grid Applications

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

SAGA

A Simple API for Grid Applications

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, ...)

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

SAGA

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

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

SAGA

A Simple API for Grid Applications

SAGA Intro: Example

```
// 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)

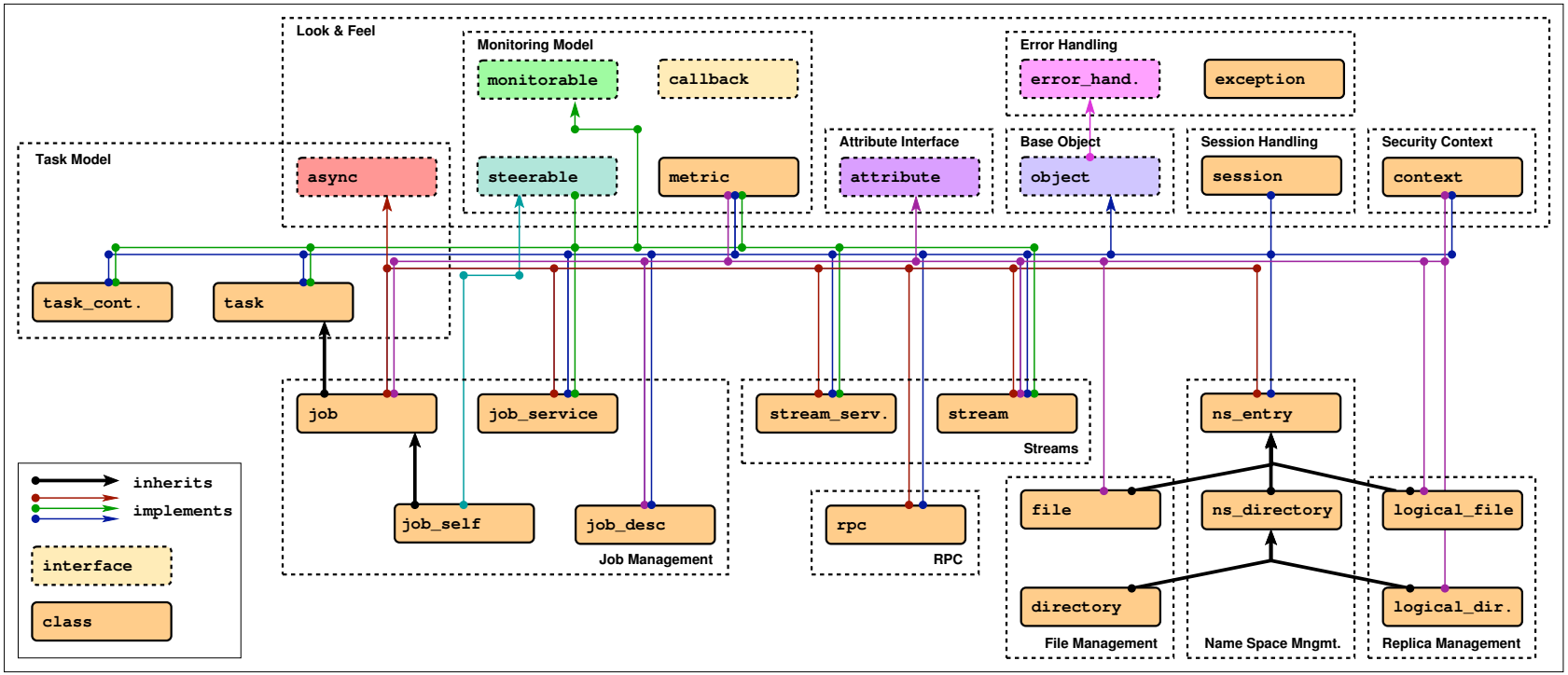
SAGA Intro: 10.000 feet

- 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

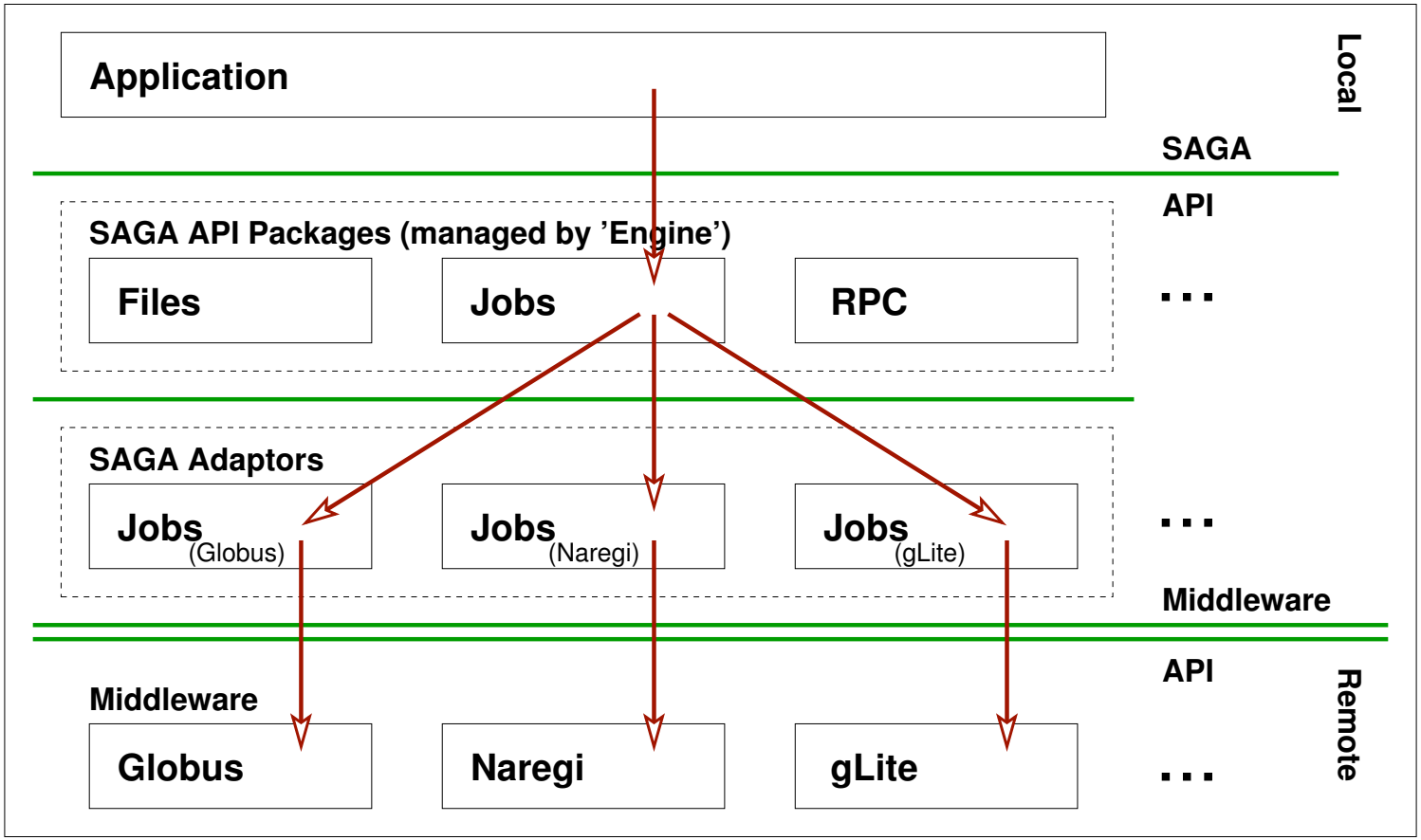
SAGA

A Simple API for Grid Applications

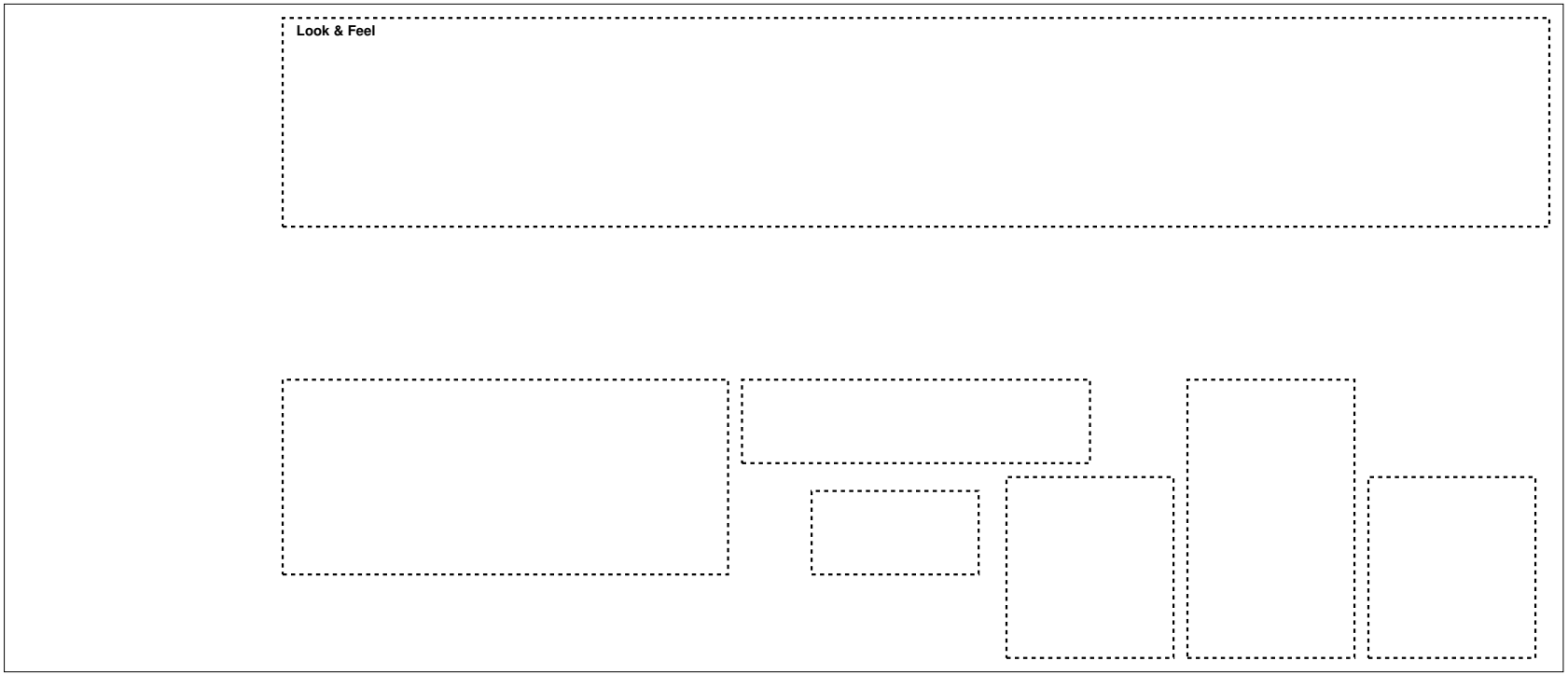
SAGA: Class hierarchy



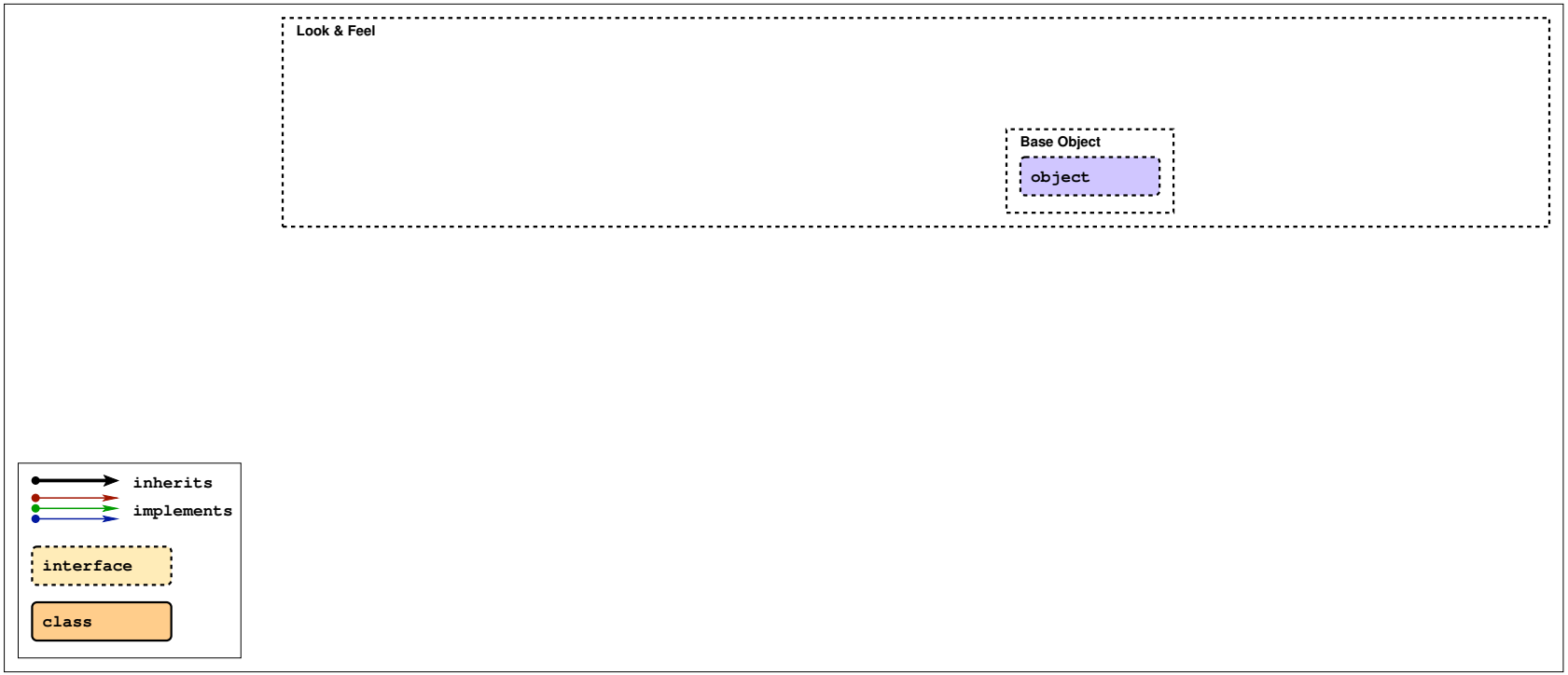
Implementation



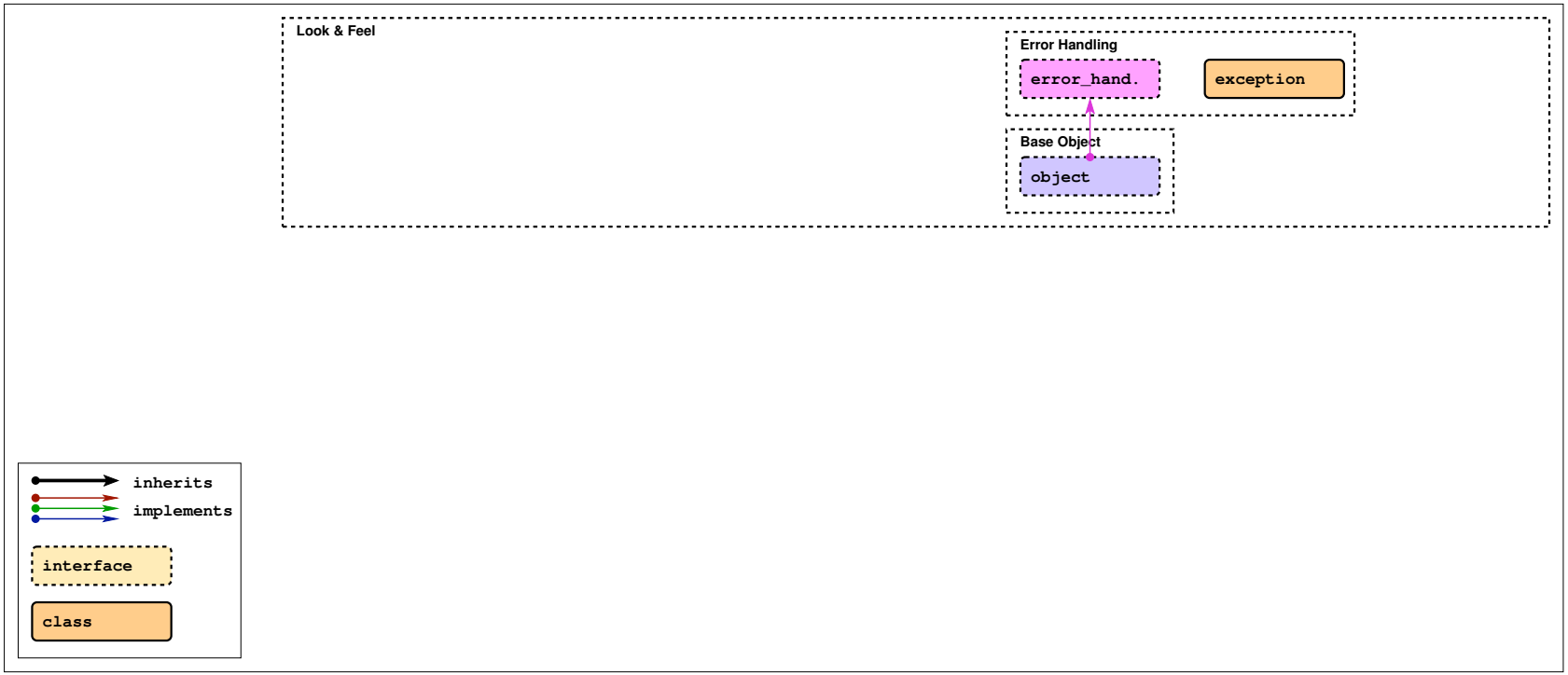
SAGA: Class hierarchy



SAGA: Class hierarchy



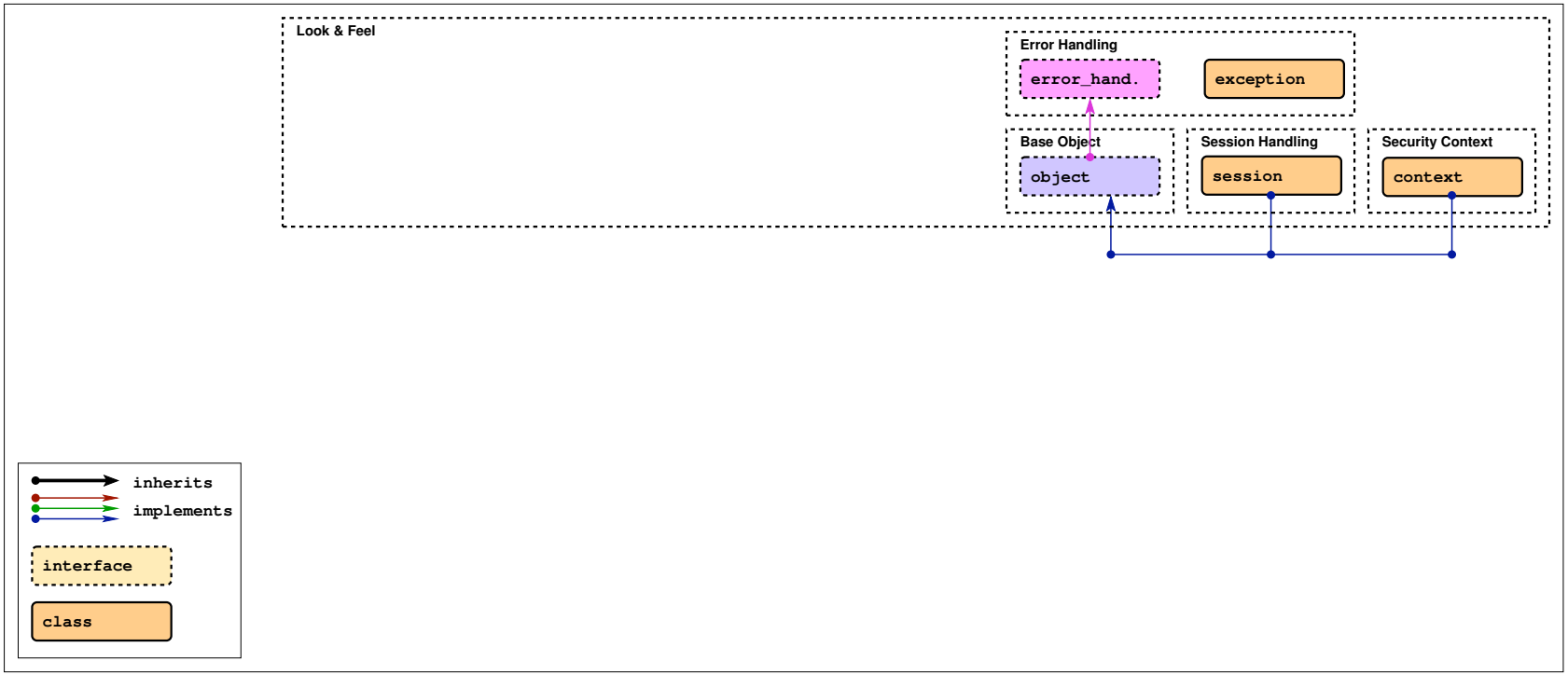
SAGA: Class hierarchy



SAGA

A Simple API for Grid Applications

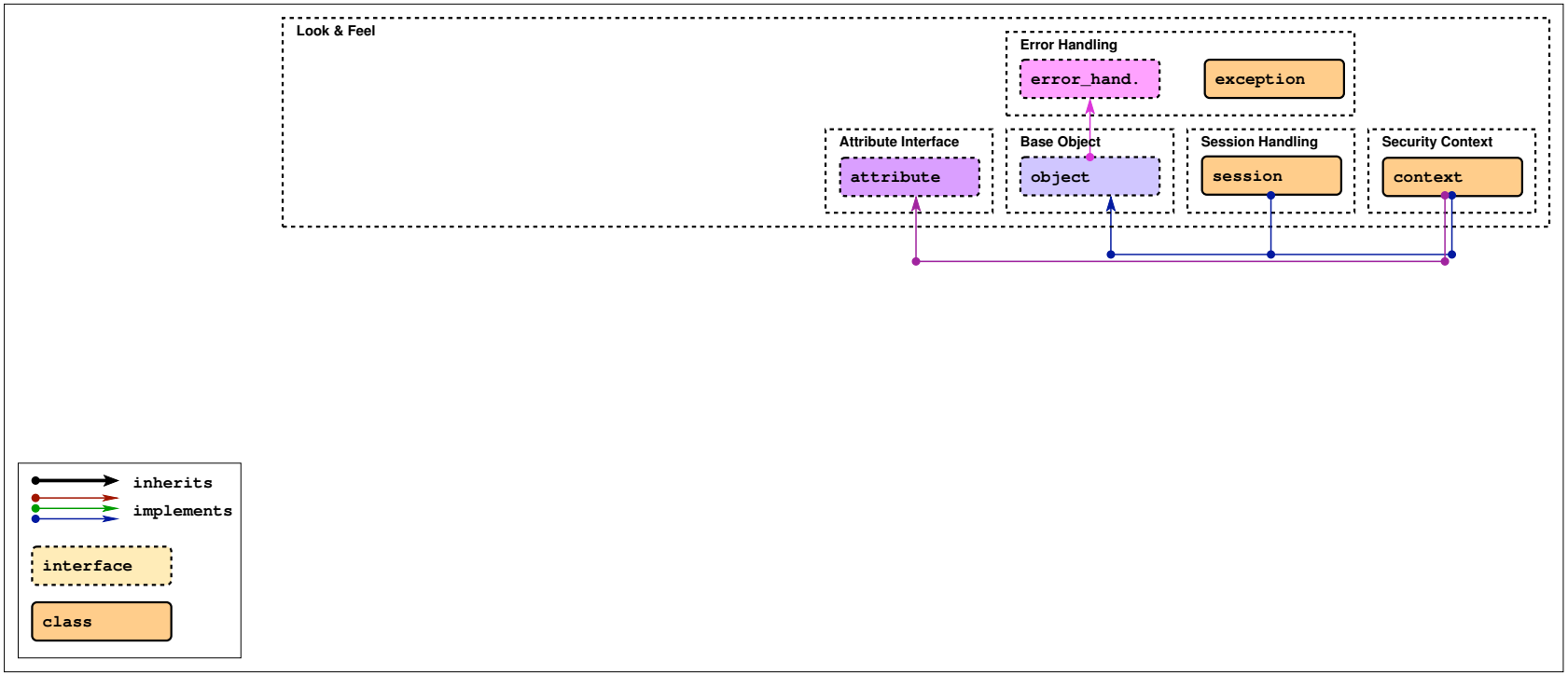
SAGA: Class hierarchy



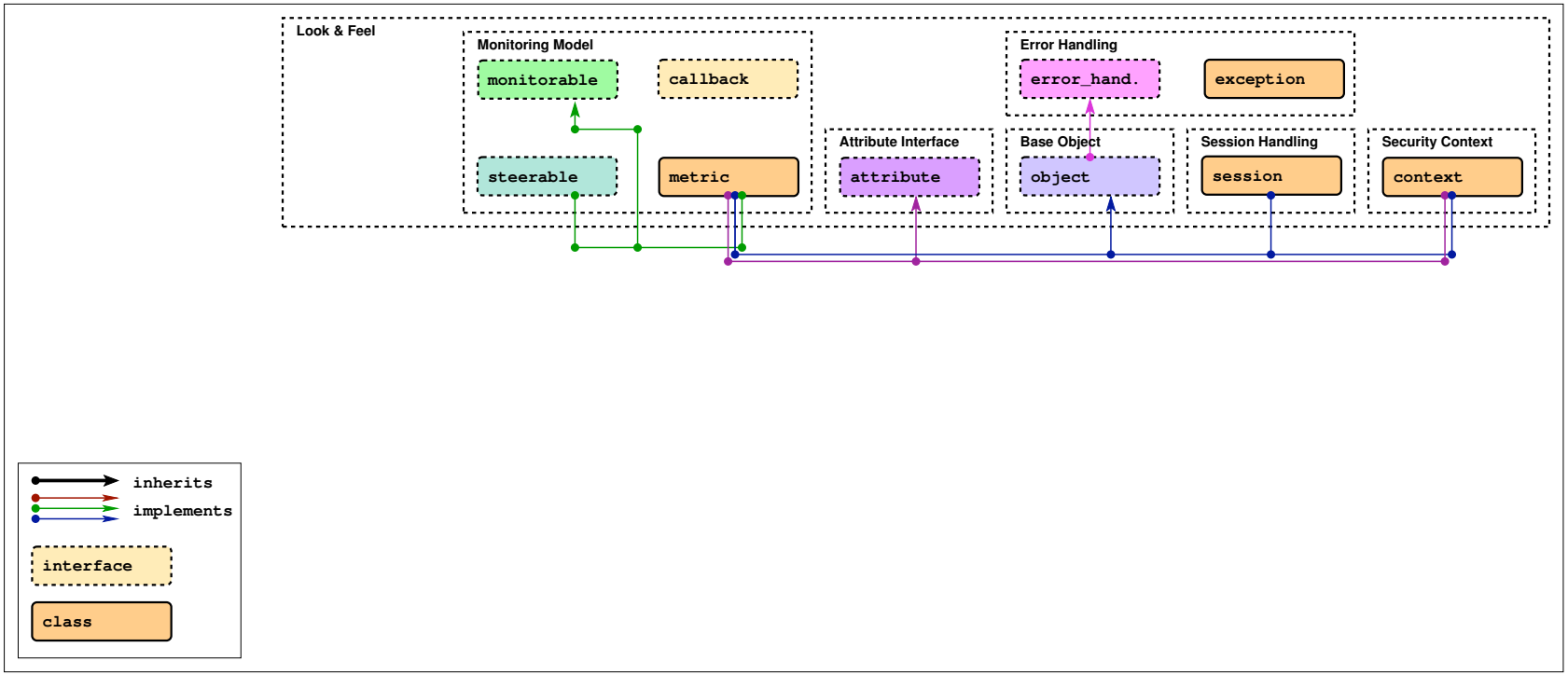
SAGA

A Simple API for Grid Applications

SAGA: Class hierarchy



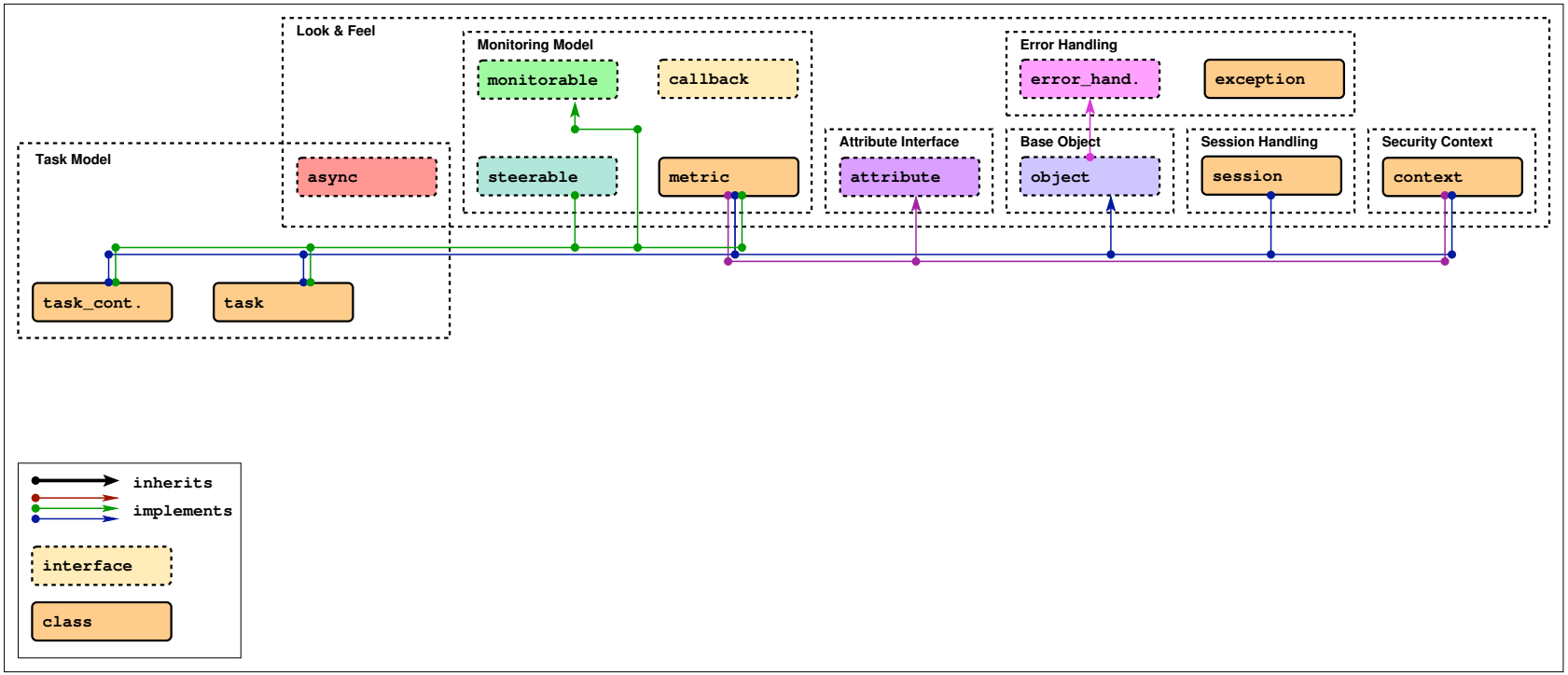
SAGA: Class hierarchy



SAGA

A Simple API for Grid Applications

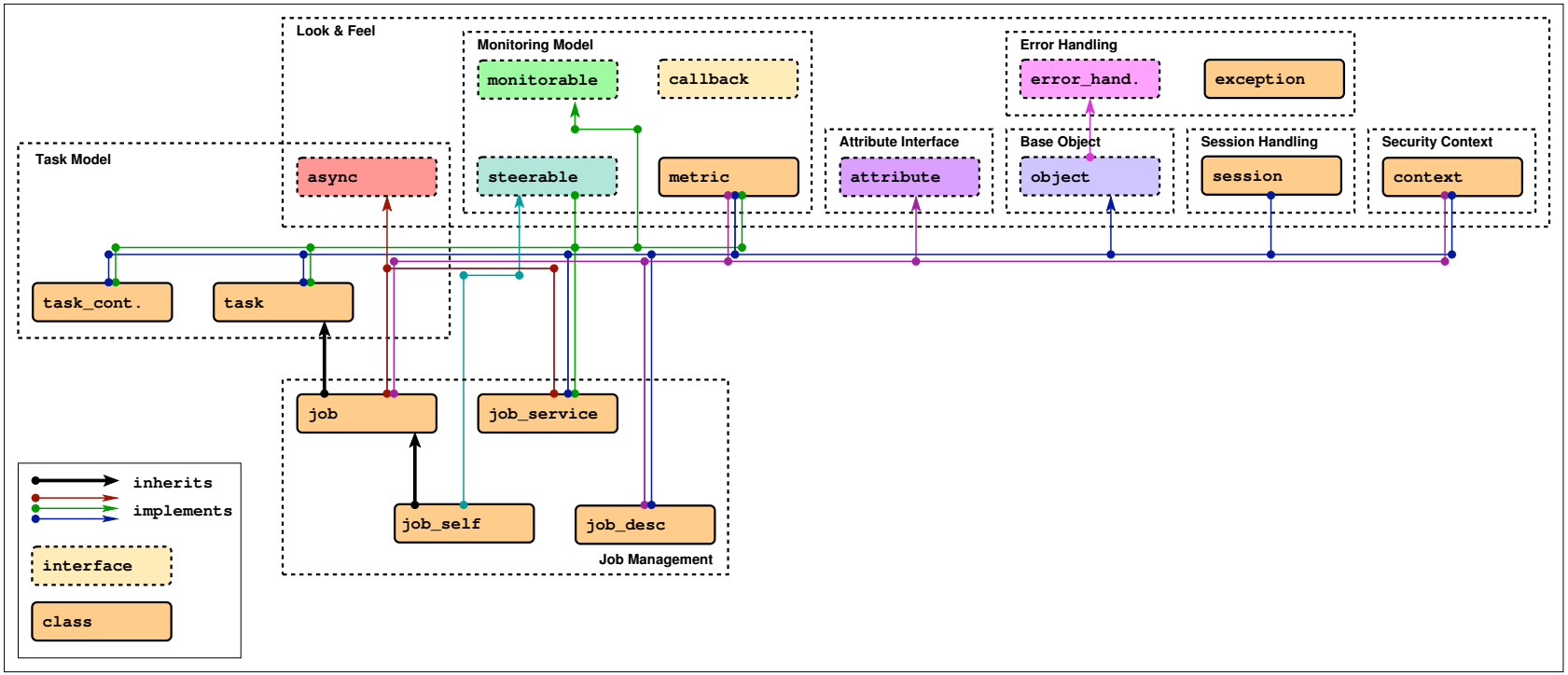
SAGA: Class hierarchy



SAGA

A Simple API for Grid Applications

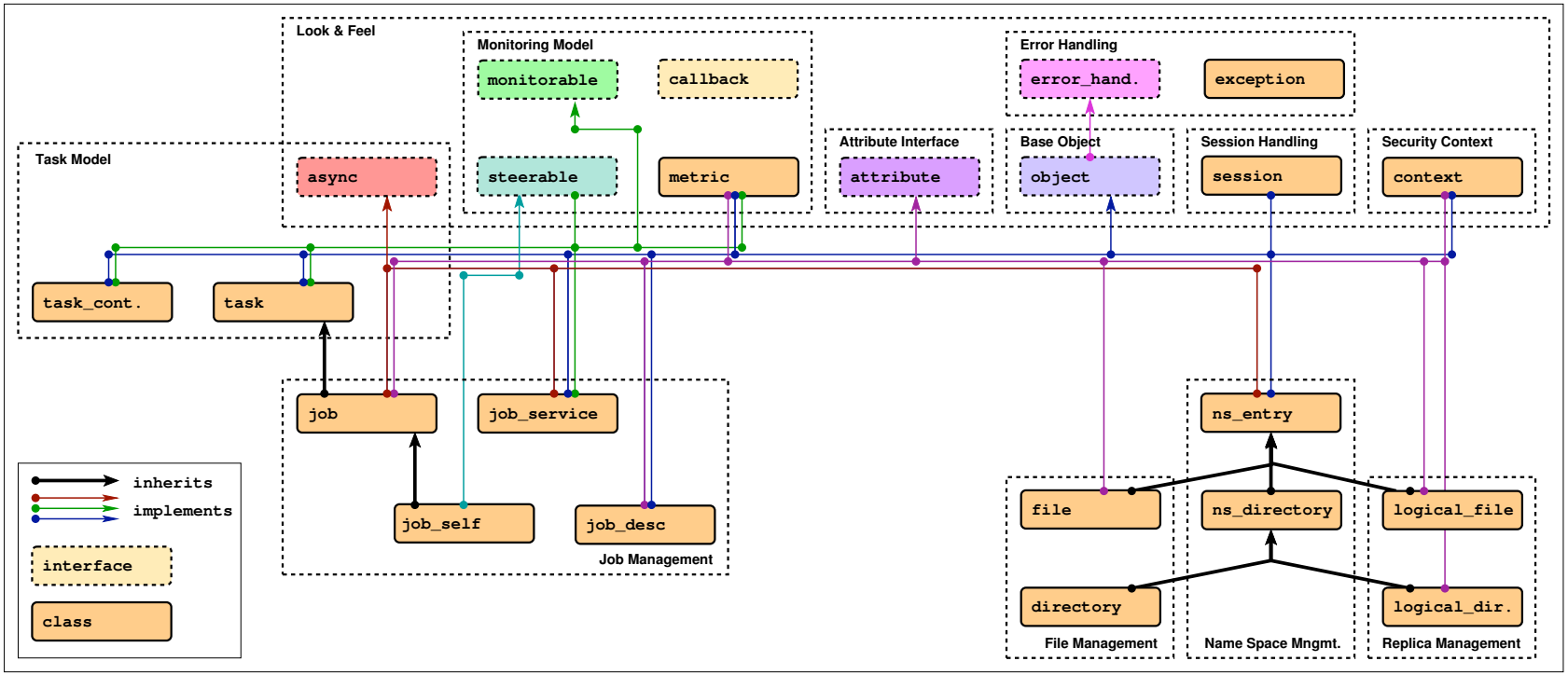
SAGA: Class hierarchy



SAGA

A Simple API for Grid Applications

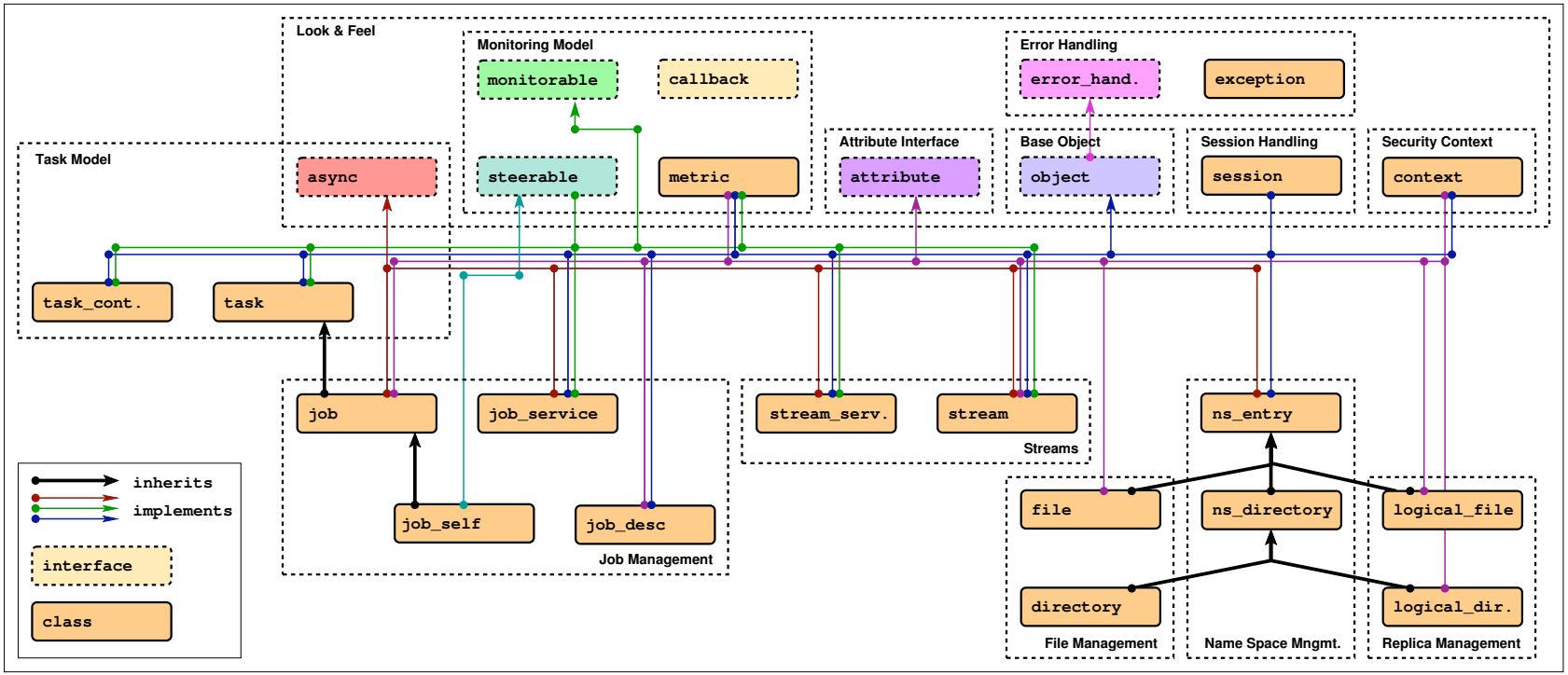
SAGA: Class hierarchy



SAGA

A Simple API for Grid Applications

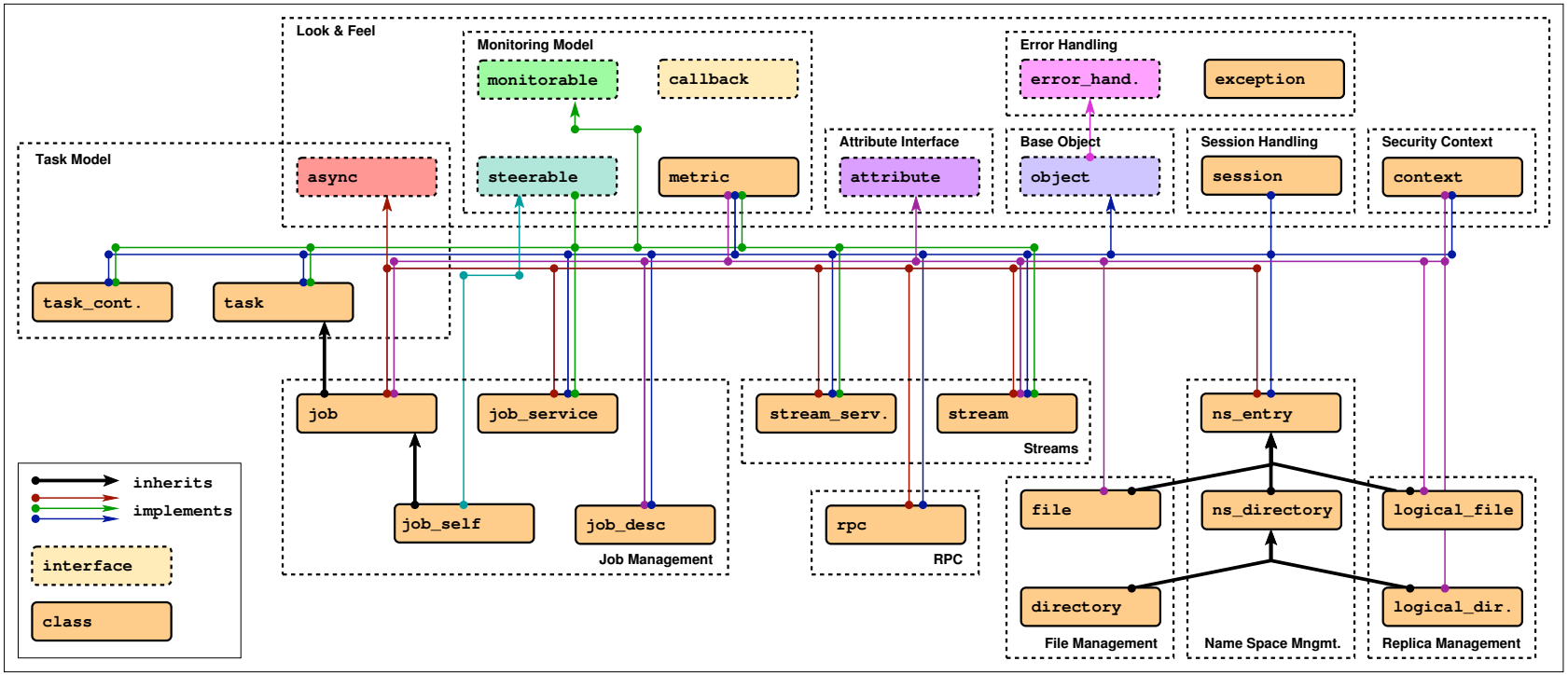
SAGA: Class hierarchy



SAGA

A Simple API for Grid Applications

SAGA: Class hierarchy



SAGA: Class hierarchy

SAGA: Class hierarchy