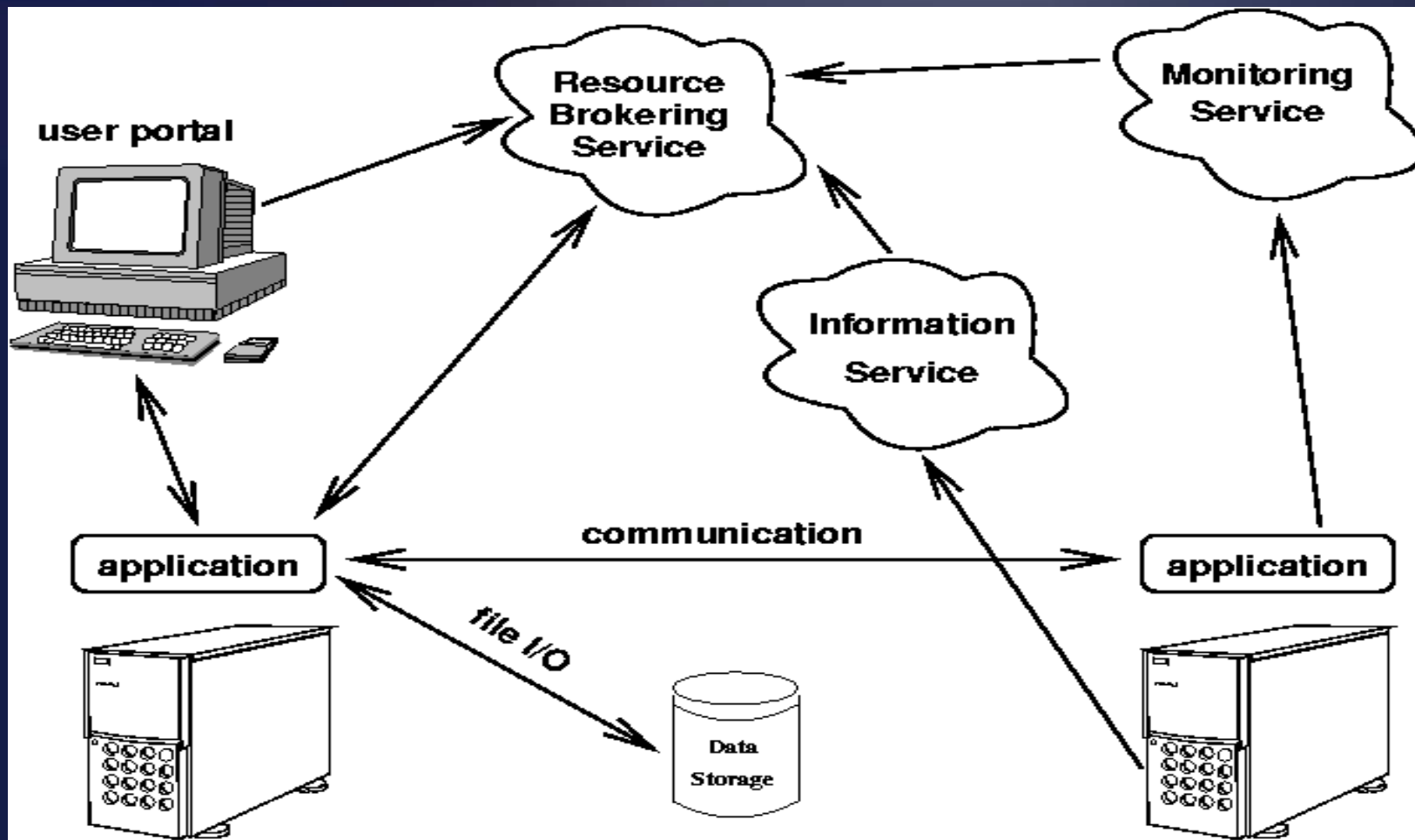


# The Simple API for Grid Applications (SAGA)

**Thilo Kielmann**  
**Vrije Universiteit, Amsterdam**  
**kielmann@cs.vu.nl**



# A Grid Application Execution Scenario



# Functional Properties of a Grid API

What applications need to do:

- **Access to compute resources, job spawning and scheduling**
- **Access to file and data resources**
- **Communication between parallel and distributed processes**
- **Application monitoring and steering**

# Non-functional Properties of a Grid API

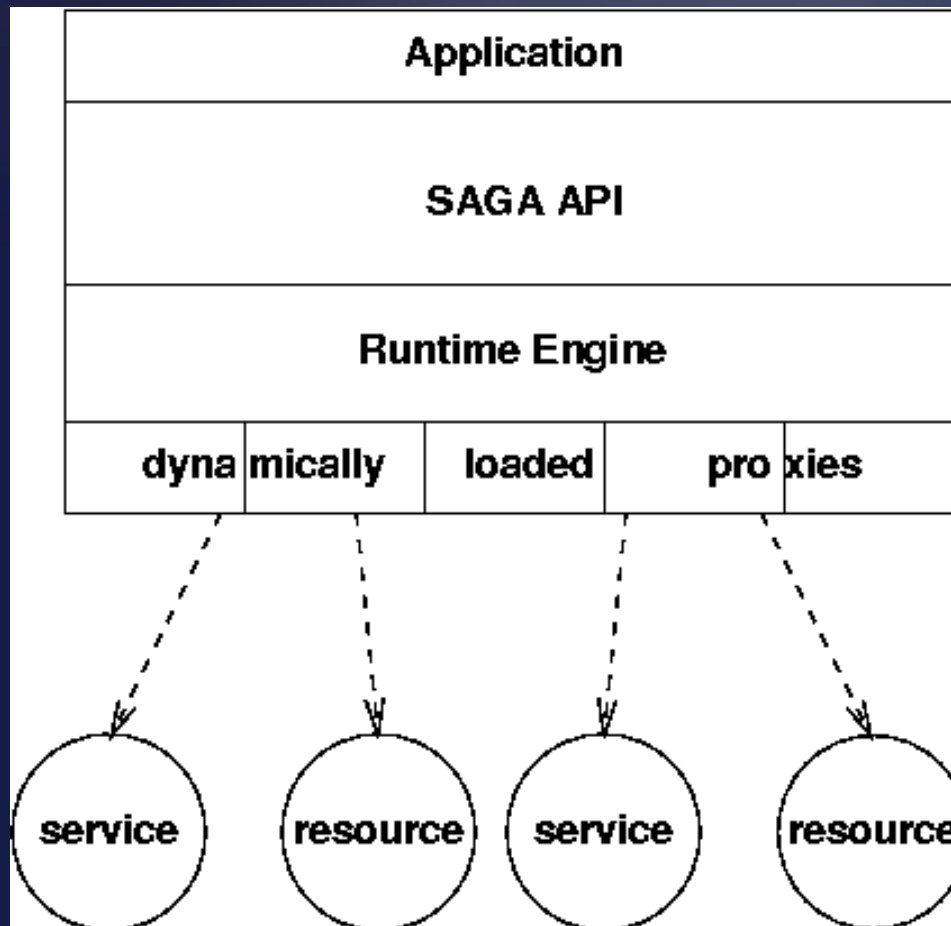
What else needs to be taken care of:

- Performance
- Fault tolerance
- Security and trust
- Platform independence

## Simple API for Grid Applications (SAGA)

- **SAGA- RG and SAGA- CORE- WG** within GGF are working on an upcoming standard for a simple Grid API
- **Design principles:**
  - **Simplicity (ease of use)**
  - **Strictly driven by application use cases**
- **API defines large part of a programming model for Grid- aware applications**

# Implementing SAGA: dynamically loaded proxies



## Functionality in SAGA

- **Jobs (submission and management)**
- **Files (and logical/ replicated files)**
- **Streams (sockets like)**
- **Application steering and monitoring**
- **Later in SAGA:**
  - **Workflow (task dependencies)**
  - **GridRPC (draft exists)**
  - **GridCPR (input pending from WG)**
- **Security (authentication)**
- **Error handling**
- **Asynchronous operations (tasks)**

## Example: Reading a Remote File (C++)

```
#include <string>
#include <iostream>
#include <saga.h>
int main () {
    try {
        // open a remote file
        saga::file f ("gsiftp://ftp.university.edu/pub/INDEX");
        // read data
        while ( string s = f.read (100) ){
            std::cout << s;
        }
    } catch (saga::exception e) {
        std::cerr << e.what() << std::endl;
    }
}
```

# SAGA Files

```

class File {
    void read      (in long      len_in,
                  out string    buffer,
                  out long      len_out );

    void write    (in long      len_in,
                  in  string    buffer,
                  out long      len_out );

    void seek     (in long      offset,
                  in  SeekMode  whence,
                  out long      position );

    void readV    (inout array<ivec>  ivec);
    void writeV   (inout array<ivec>  ivec);
                // plus more optimized (bulk) versions
}

```

(directories left out for brevity)

## SAGA Replicated Files

```
class LogicalFile {  
  
    void addLocation      (in name                );  
    void removeLocation  (in name                );  
    void listLocations   (out array<string,1> names );  
    void replicate       (in name                );  
  
}
```

(directories left out for brevity)

## Example: Running a Remote Job (C++)

```
#include <iostream>
#include <saga.h>
int main() {
    try{
        std::iostream in, out, err;
        saga::job_service js;
        saga::job job = js.runJob ("host.university.edu",
                                   "/bin/date", &in, &out, &err);
        while ( job.getJobState() != saga::job::Done &&
                job.getJobState() != saga::job::Failed ) {
            std::cout << out;
            std::cerr << err;
        }
        catch (saga::exception e){
            std::cerr << e.what() << std::endl;
        }
    }
}
```

# SAGA Jobs

```
interface Job {  
    void get_job_id          (out string          job_id);  
    void get_state          (out state          state);  
    void get_job_definition (out job_definition job_def);  
  
    void get_stdin          (out opaque          stdin);  
    void get_stdout        (out opaque          stdout);  
    void get_stderr        (out opaque          stderr);  
  
    void suspend            (void);  
    void resume             (void);  
    void checkpoint        (void);  
    void migrate            (in job_definition job_def);  
    void terminate          (void);  
    void signal             (in int             signum);  
}
```

# SAGA JobService

```
interface JobService {  
    void create_job (in string          res_mgr,  
                   in job_definition jobDef,  
                   out job             job);  
  
    void run_job   (in string          res_mgr,  
                  in string          commandline,  
                  out opaque         stdin,  
                  out opaque         stdout,  
                  out opaque         stderr,  
                  out job             job);  
  
    void list      (out array<string,1> job_ids);  
    void get_job   (in string          job_id,  
                  out job             job);  
  
    void get_self  (out job             job);  
}
```

# SAGA Security

```
enum contextType {
    GSI                = 0,
    MyProxy            = 1,
    SSH                = 2,
    Kerberos           = 3,
    UserPass           = 4,
    KeyStore           = 5
};

interface Context extends-all SAGA.Attribute {
    constructor (in contextType type);
    getType     (out contextType type);
}
```

- **Every SAGA object gets a Session with a Context as parameter to its constructor.**

## SAGA Error Handling

- Errors are signaled using exceptions
- 14 SAGA exceptions have been defined

## SAGA Tasks

- Asynchronous operations
- Bulk (async.) operations
- Single-threaded implementation support

```
package Task {
```

```
enum state {
```

```
    Unknown = -1,
```

```
    New      = 1,
```

```
    Running = 2,
```

```
    Done    = 3,
```

```
    Failed  = 4
```

```
};
```

```
...
```

# Tasks and Containers

```
interface Task {
    void run      ();
    void wait     (in double timeout,
                  out boolean finished);
    void cancel   ();
    void get_state (out State state);
}

class TaskContainer {
    void add      (in Task task);
    void remove  (in Task task);
    void run      ();
    void wait     (in double timeout,
                  out array<Task,1> finished);
    void cancel   ();
    void get_states (out array<State,1> states);
    void get_tasks (out array<Task,1> tasks);
}
```

# Instantiating Tasks

Have three versions of each operation:

- Synchronous
- Asynchronous (start immediately)
- Task (start explicitly)

```
                                d.mkdir      ("test/");  
saga::task t_1 = d.mkdir <sync>  ("test/"); // Done  
saga::task t_2 = d.mkdir <async> ("test/"); // Running  
saga::task t_3 = d.mkdir <task>  ("test/"); // New  
  
t_3.run ();  
  
t_2.wait ();  
t_3.wait ();
```

## Summary: SAGA

- Standardize a *simple* API for Grid applications
- Driven by user communities
- API completion Q2/ 2006
  - Currently nailing down last open issues
- Implementations:
  - C++ (SAGA- A) almost complete
  - Java
    - DEISA (Edinburgh): Job and Files
    - OMII- UK: almost complete

## Conclusions

- **SAGA: Simple API to various Grid systems and middlewares**
- **Work in progress:**
  - Finalizing API spec
  - Completing implementations
    - Proxies (“adaptors”)

***[wiki.cct.lsu.edu/saga/space/start](http://wiki.cct.lsu.edu/saga/space/start)***

***<https://forge.gridforum.org/projects/saga-rg/>***