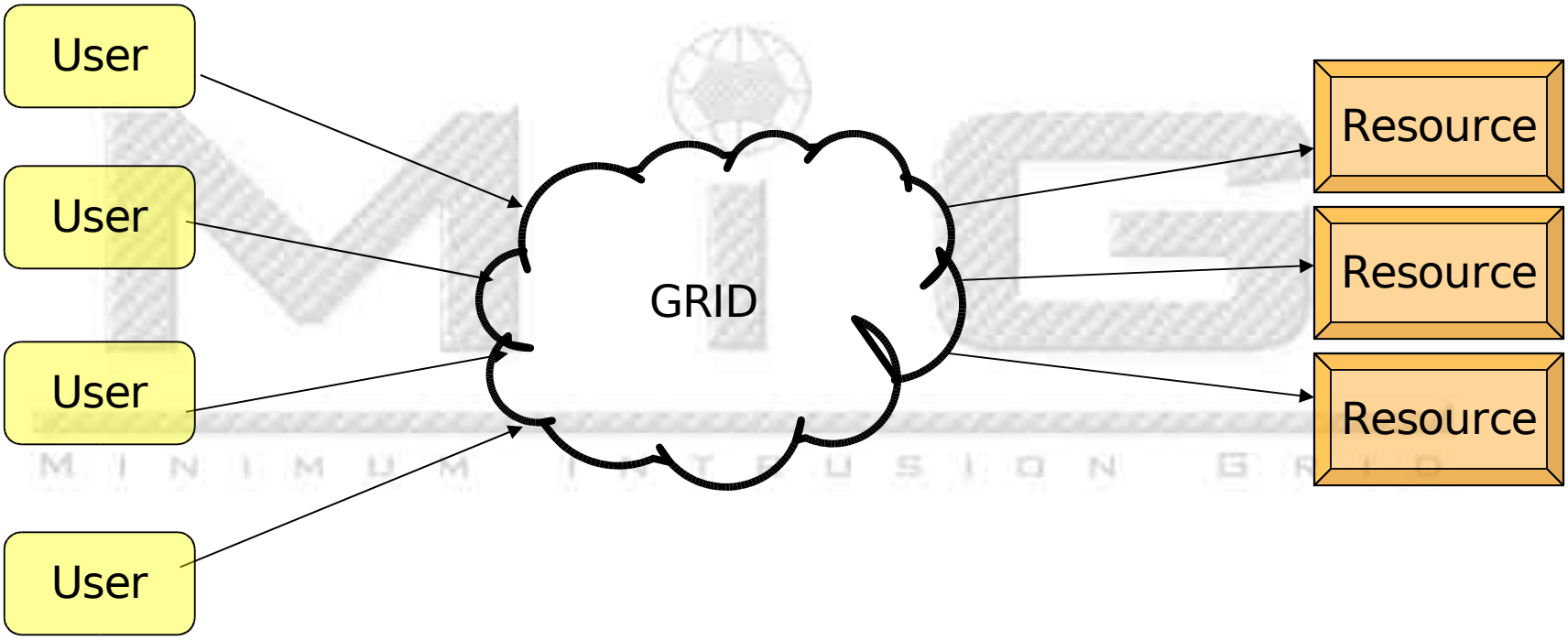
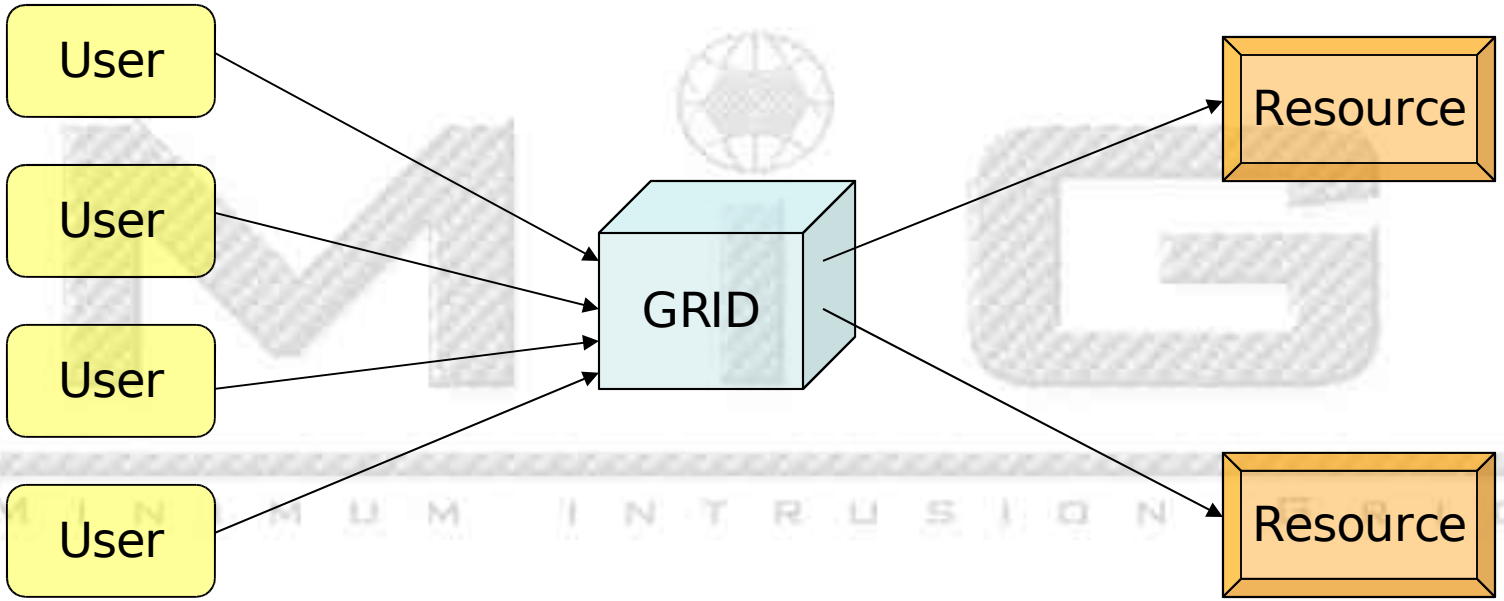
The Simple Model

Design and implementation

The abstract MiG model

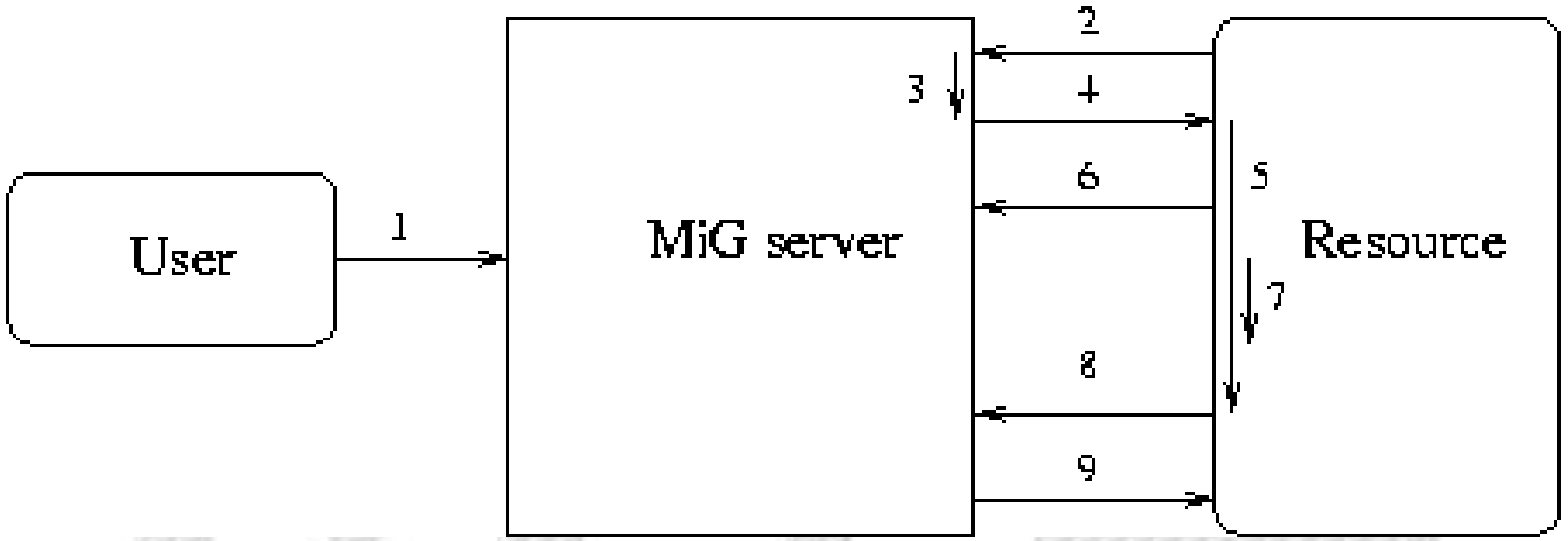


The simple MiG model



- Grid cloud in abstract model is replaced with a single grid server
- Full featured grid solution
- Obvious problems: single point of failure, performance

Simple model - design



1. User communicates with the MiG server using HTTPS and certificates.
2. Resource requests a new job to execute.
3. The MiG server creates the job script.
4. MiG server sends the job to the resource using SCP.
5. The resource starts the job script.
6. Resource requests the needed inputfiles from the MiG server
7. The job is being executed
8. Resource sends outputfiles to the MiG server
9. MiG server cleans up the resource using SSH (files and processes).

New concepts

- A job also consists of getting the inputfiles and sending the outputfiles (Rasmus' solution)
- Home directory on grid server
  Inputfiles must be in the users personal home directory and outputfiles is send to the home directory

MiG from the users POV

-Browser (x.509 certificate and HTTPS)
  Manage files in grid home directory
  Submit jobs, view jobstatus etc.
-MiGscripts. Wrappers around “curl”.
  File handling:
    MiGput, MiGget, MiGremove,
    MiGlist, MiGcat
  Job handling:
    MiGsubmit, MiGstatus, MiGallstatus
    (MiGkill)

MiG from the users POV

- http://mig-1.imada.sdu.dk
- Certificate in x.509 p.12 format.
  Import in browser.

MiG user scripts

- Very simple. As much code as possible is placed on the MiG server.

- Example (MiGlist.sh):

```
search=$1
size=false
with_html=false
curl --cert $certfile --key $key --pass $pass $migserver/cgi-bin/myfiles.py?
search=$search\&amp\;size=$size\&amp\;with_html=$with_html

karlsen@adina:~/mig/user> ./MiGlist.sh "*.txt"
README.txt
txtfile.txt
```

mRSL job specification

- Globus RSL and Nordugrid xRSL too complex.
- Most keywords are similar to those in RSL/xRSL:

EXECUTE,INPUTFILES,OUTPUTFILES,
EXECUTABLES,CPUTIME,MEMORY,
DISK,RUNTIMEENVIRONMENT,JOBNAME,
NOTIFY,ARCHITECTURE,ENVIRONMENT,
CPUCOUNT,NODECOUNT,MAXPRICE

mRSL example

```
::EXECUTE::
echo "Hello World"
uname -a
cat inputfile >> outfile

::NOTIFY::
jabber: karlsen@jabbernet.dk
karlsen@imada.sdu.dk

::INPUTFILES::
inputfile

::OUTPUTFILES::
outfile

::MEMORY::
128

::DISK::
10

::MAXPRICE::
30

::CPUTIME::
1000

::JOBNAME::
myjobname
```

MiG from the resource POV

- Configuration on MiG server
  Updated using browser or script (HTTPS and certificate)

Scheduler: architecture, disk, memory, cpucount, nodecount, runtimeenvironment

Pricing: minprice

Other: scriptlanguage (sh or python) hosturl, miguser,hostkey (scp/ssh to the resource).

MiG from the resource POV

miniscript.sh requests and executes a single job. Pseudo code:

newjob = curl mig.server.url/cgi-bin/requestnewjob?cputime=$cputime
chmod +x newjob
./newjob

Resource without queue system: no_queue.sh. Pseudo code:

while [1] ; do
./miniscript.sh
done

The central MiG server

- Apache server with HTTPS and x.509 certificates
- Cgi-scripts: jobstatus.py, removefile.py, requestnewjob.py etc.
- grid_script.py:
  When a new job is received from a user and it is parsed successfully the script is notified
  The same script is notified when a resource requests a new job to execute.

The central MiG server

- First Fit scheduler
- Job script generator

scriptlanguage in res. configuration

createJobDirectory
cdToJobDirectory
getInputFiles
getExecutables
chmodExecutables
setEnvironments
setRuntimeEnvironments
execute
sendOutputFiles
sendStatusFiles

- Sends job to resource using SCP

Implementation status

- Most basic functionality implemented
- One user (gene research)
- 83 resources, 122 cpu's
- Monitor

http://mig-imada.sdu.dk/monitor.html

The simple model - future

- Continue the implementation phase
- Add features (eg. Killjob)
- Have more test users and resources with different setups (large PBS clusters etc.)
  This will without doubt create new feature requests
- Documentation :-(