Sixth Framework Programme

IST-5-033437

Chemomentum

A versatile execution management system for Next-Generation UNICORE Grids

Bernd Schuller, Roger Menday, Achim Streit

Distributed systems and Grid computing division, ZAM, Research Centre Juelich

b.schuller@fz-juelich.de

Forschungszentrum Jülich
in der Helmholtz-Gemeinschaft
50 Jahre / Zukunft

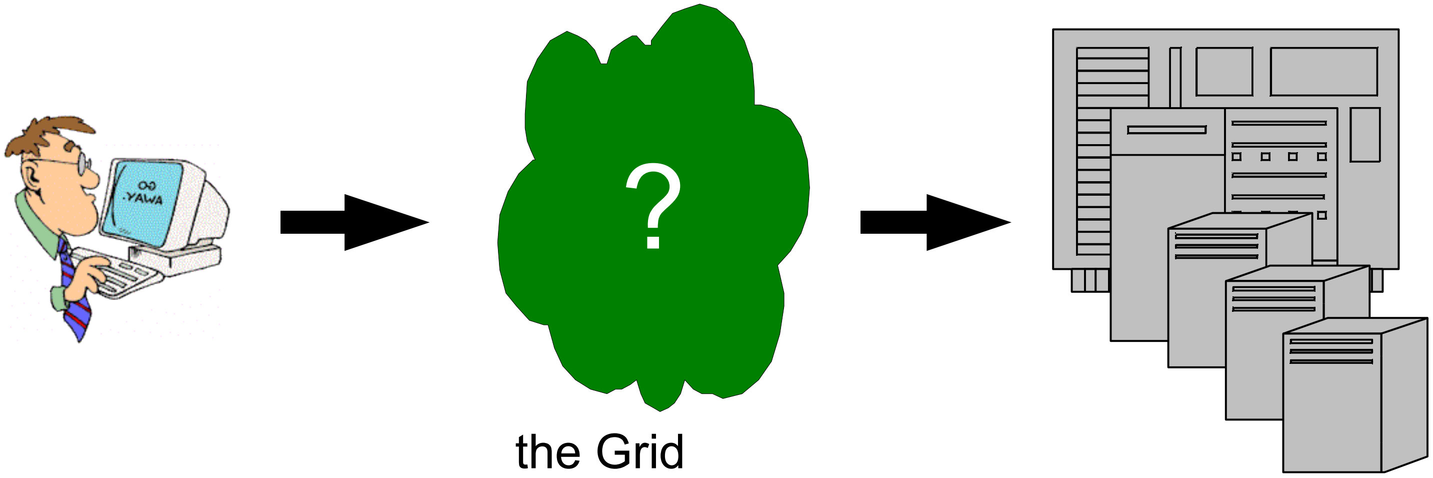
Outline

- Introduction: execution management in Grids
- Motivation: do we need an „X“ NJS ?
- XNJS design & implementation
- Usage examples and performance
  - Chemomentum scenarios
- Outlook & future work

Execution management in Grids

- obviously, we want to run jobs
- Execution management systems bridge the gap

from abstract middleware
to concrete target systems



Research areas

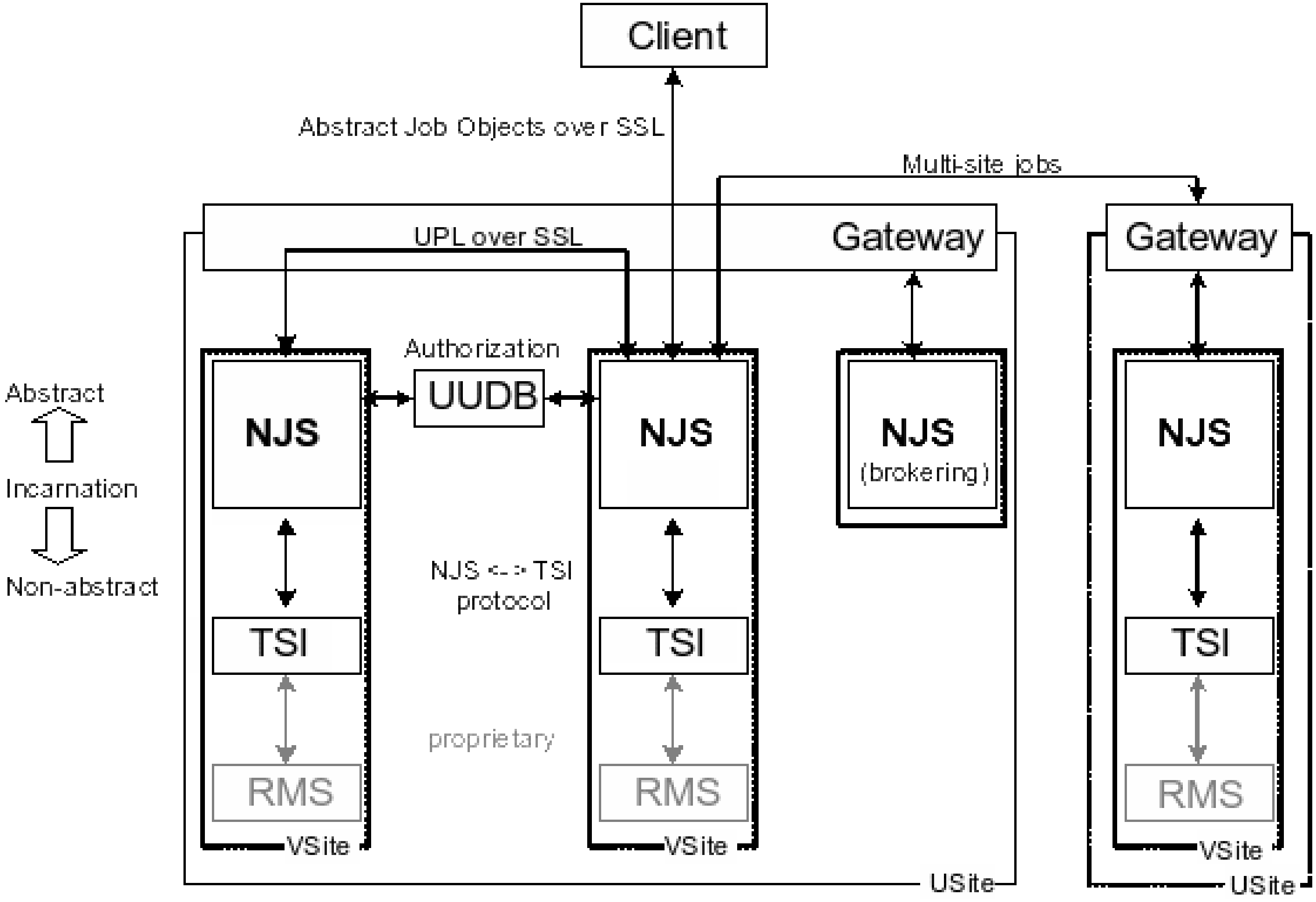
- interfaces:
  - legacy, OGSA-BES, UNICORE 6 atomic services
- languages:
  - AJO, JSDL, ...
- integration into Grid infrastructure:
  - legacy (e.g. UNICORE 4, GT2.x)
  - WS, WSRF (e.g. UNICORE 6, Globus 4)
  - WS-NonexistentStandards?

- what about the **software** that does the actual work?

Concrete execution management

- UNICORE 5 as an example
  - NJS

The UNICORE 5 architecture



NJS – the heart of UNICORE 5

- Job management
  - authorise users using the UNICORE user database (UUDB)
  - translate the incoming abstract jobs into concrete jobs for the target system
  - submit the concrete jobs to the TSI and monitor their status
  - manage the outcome
- Communication
  - with the client (through the gateway)
  - with the TSI
  - with other NJSs
- Add-on functionality
  - accounting, resource reservation, AFT, ...

Core requirements for EMSs

- manage jobs
  - typical activity:
    - data in, execute, data out
- manage user access to jobs
  - submit, stop, start, ...
- support UNICORE concepts
  - Uspace: temporary job dir
  - Applications: abstract access to executables
  - Abstract filespaces (HOME, ROOT, ...)
  - nice to have: UNICORE 5 TSI support

Yet another NJS?

- functionality is only half the story...

- Thesis:

  existing NJS is not up to the challenges of Grid systems „beyond“ UNICORE 5:

  – it does not meet *most* of the **non functional requirements**

  – it does not meet *some* of the functional ones

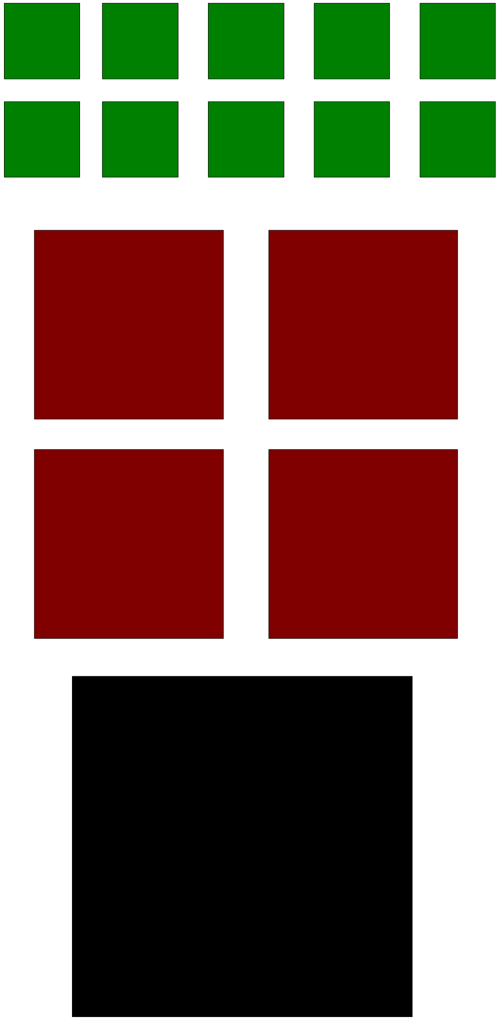
- present some „evidence“ in the following...

Let's limit the scope...

- Deal with „atomic“ activity, which typically consists of
  - data stage in
  - execute
  - data stage out
- no workflow

Challenge: Usage scenarios

- Grids come in different sizes ...
- Dimensions:
  - big systems or small systems?
  - many nodes or few nodes?
  - many users or few users?
  - small jobs or large jobs?
  - focus on response time or reliability?
  - ...
- can one size fit all?

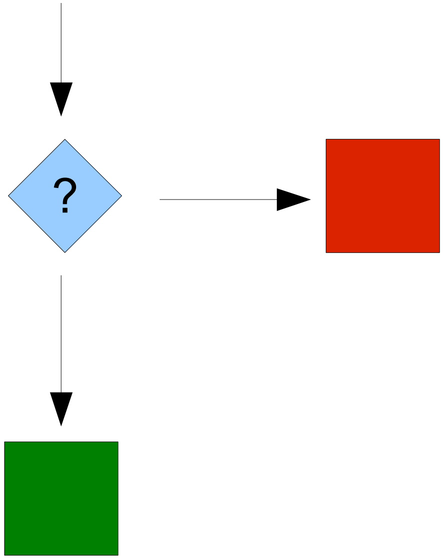


Goal: Reconfigurability

- be adaptable to varying deployment scenarios
- reconfiguration, not re-implementation

Challenge: Grid business rules

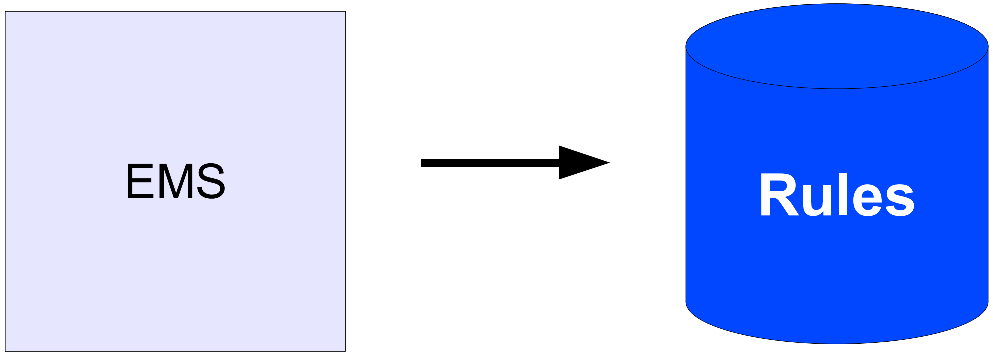
- Grid deployments vary in many operational aspects, or „business rules“



- Examples
  - how is accounting done?
  - what and where do we log, or write tracing information?
  - how do we deal with communication, e.g. notifications?
  - how do we recover from errors?
  - how is „successful“ job completion defined?

Goal: Explicit business rules

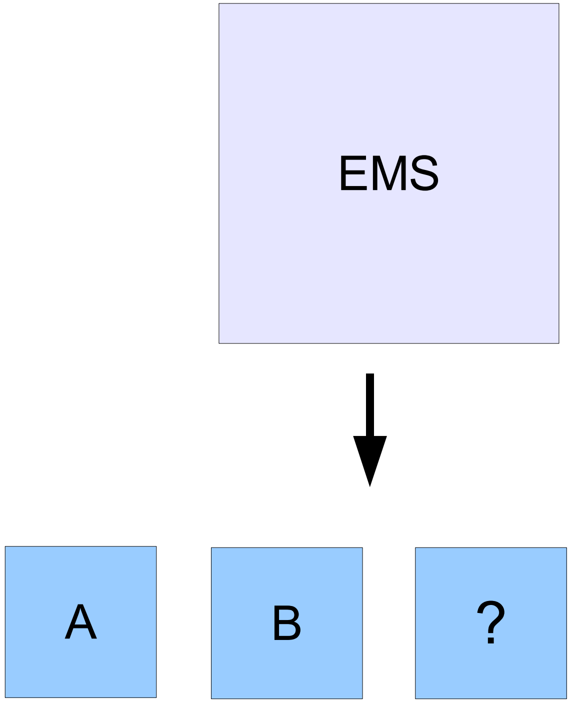
- make rules explicit (instead of „hiding“ them in the code)
- make rules modifiable



Challenge: Add-on functionality

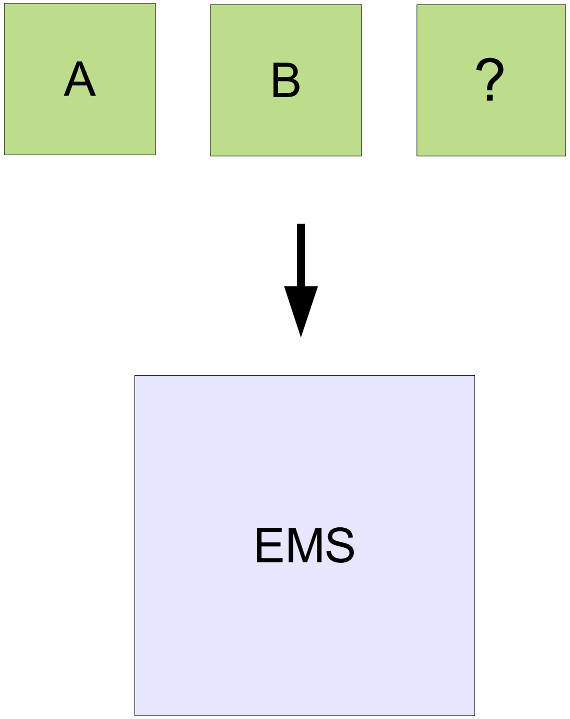
- Different deployments need to integrate with different third-party systems

- Examples:
  - LDAP, VOMS,...
  - Kerberos, Shiboleth, ACEGI, ...
  - resource usage, accounting systems
  - information services (e.g. GT MDS)
  - notification systems (wsn, mail, sms, jabber..)



Important special case: front-end

- trends change
  - AJO/UPL
  - UNICORE atomic services
  - OGSA-BES, ESI
  - ... ?



- front-end interfaces must be exchangeable

- other possiblity: embed EMS into a bigger app

Goal: Extensibility

- maximum extensibility
- „design for change“

Challenge: Flexible processing

- Requirements **will** change. The engine may need to learn „new tricks“.

- Examples:
  - add encryption/decryption of data
  - add a new filetransfer protocol
  - add new types of activity:
    might be JSDL today, but what about tomorrow?

Goal: extensible processing rules

- design system for extensible „processing rules“
- Allow...
  - adding new activity „types“
  - adding new processing steps for a given activity
  - adding new ways of performing the same processing step
- ... by re-configuration, not re-implementation of existing code

Challenge: Scalability

- Handle large numbers of jobs and/or users reliably
  - at least with well-defined characteristics, for example performance degrades, but system stays online

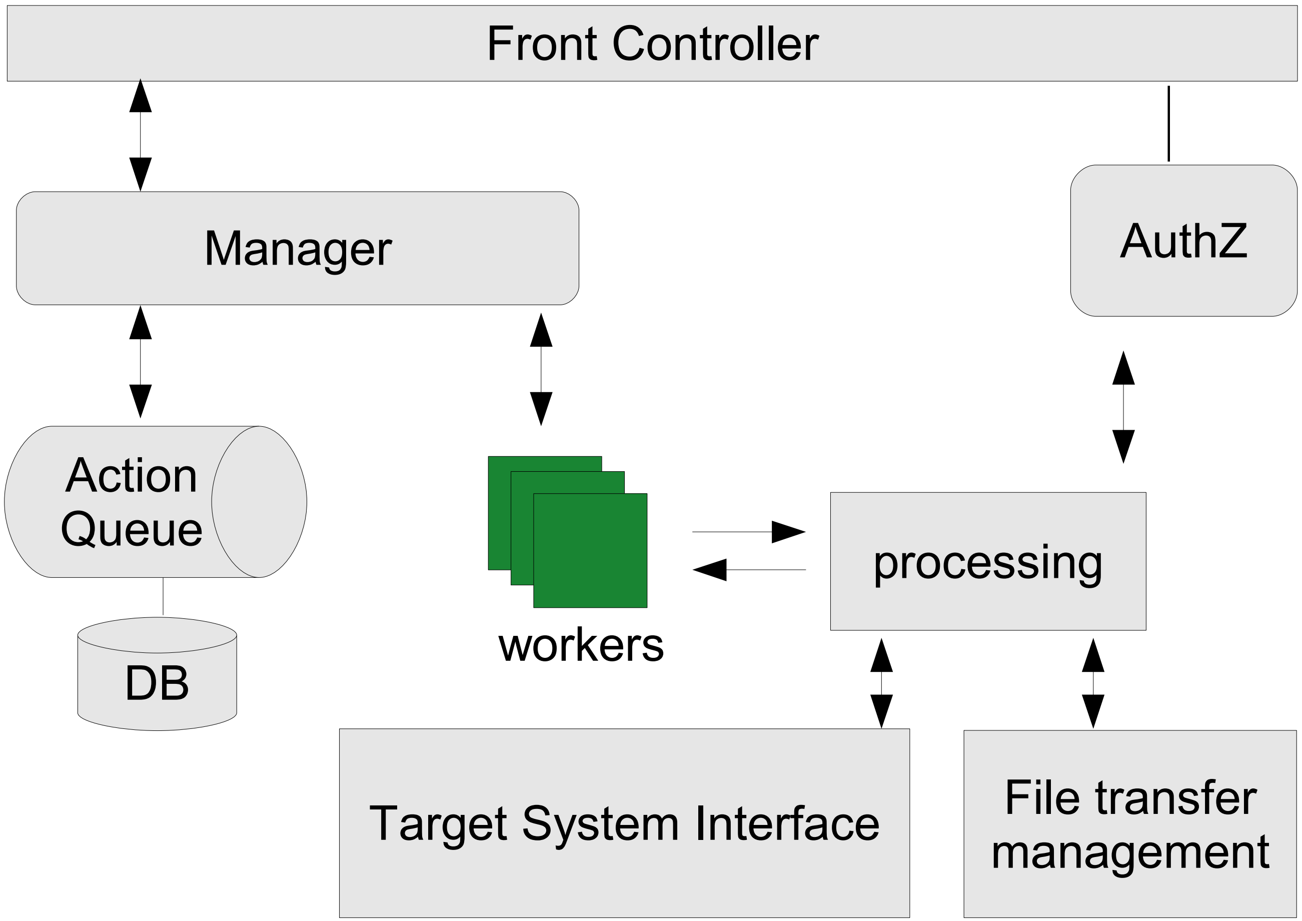
Goal: Build a scalable system

- Design
  - – make sure that the system does not go down easily
  - – design for clustering and loadbalancing
- Implementation
  - – manage internal resources (memory, threads) carefully
  - – avoid large amounts of in-memory storage

XNJS: design and implementation

- not enough time to cover everything, so focus on
  - overall architecture
    - how modularity and extensibility are achieved
  - core engine (action processing)
    - flexible processing
    - how scalability is achieved
  - example: JSDL processing

XNJS: overall architecture



Key:

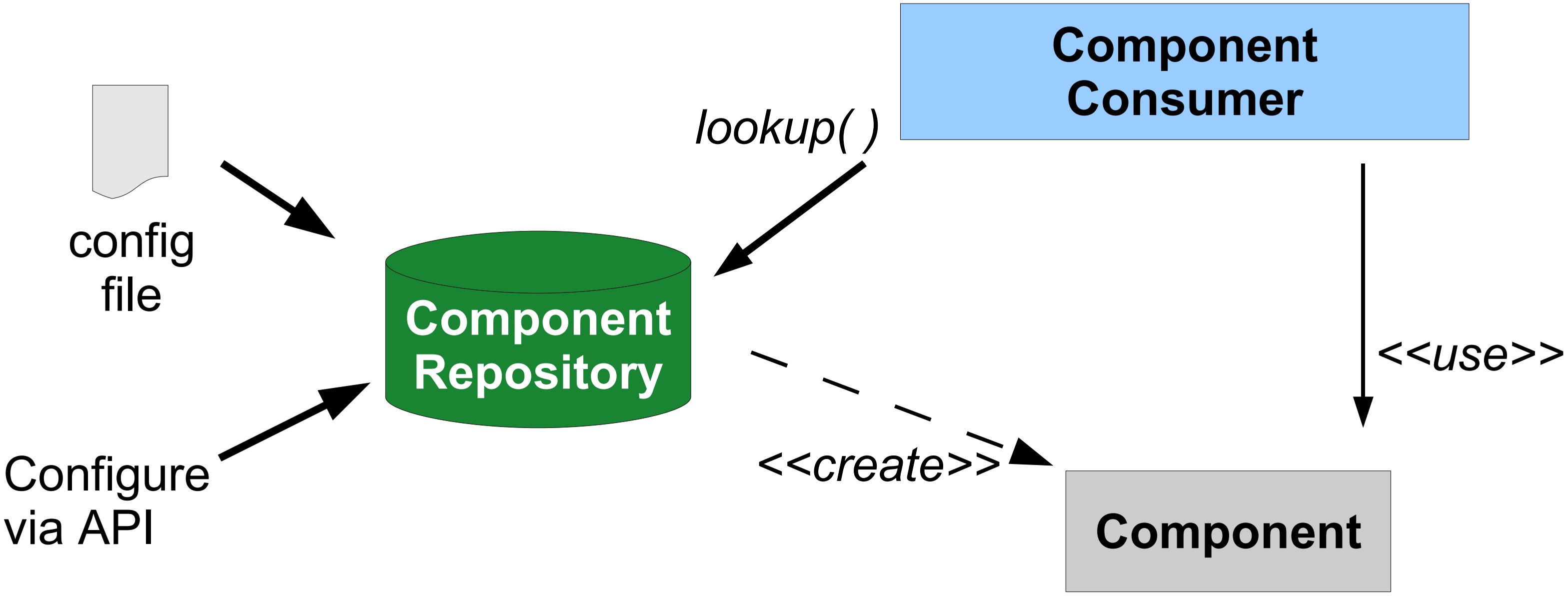
⬜	pluggable
🟩	fixed

Scalability measures

- very low memory footprint
  - use database for storing actions
  - only book keeping done in-memory
  - scales to very high numbers of actions
- many worker threads possible
- component design makes clustering possible
  - example: clustered Manager implementation

Modularisation concept

- Separate interfaces and implementation classes
- Use a component repository
  - Components lookup other components *by interface*
- Concrete system configuration defined in a config file

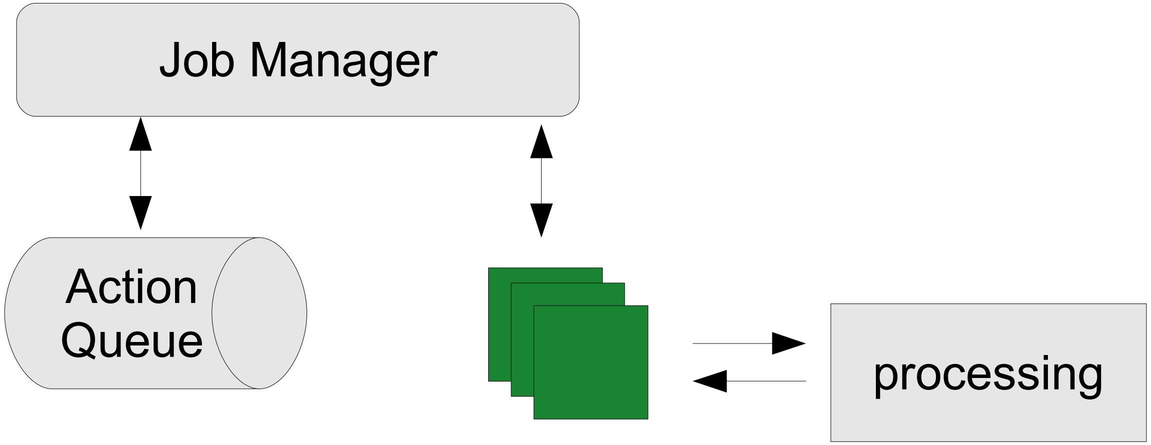


Modularisation concept

- Possible component repositories one can use
  - Spring Framework
    - powerful (many Java EE APIs, AOP, ...)
    - integrates very well with other systems
    - quite big
  - PicoContainer
    - small and light

- current XNJS implementation: PicoContainer
  - simple to replace ☺

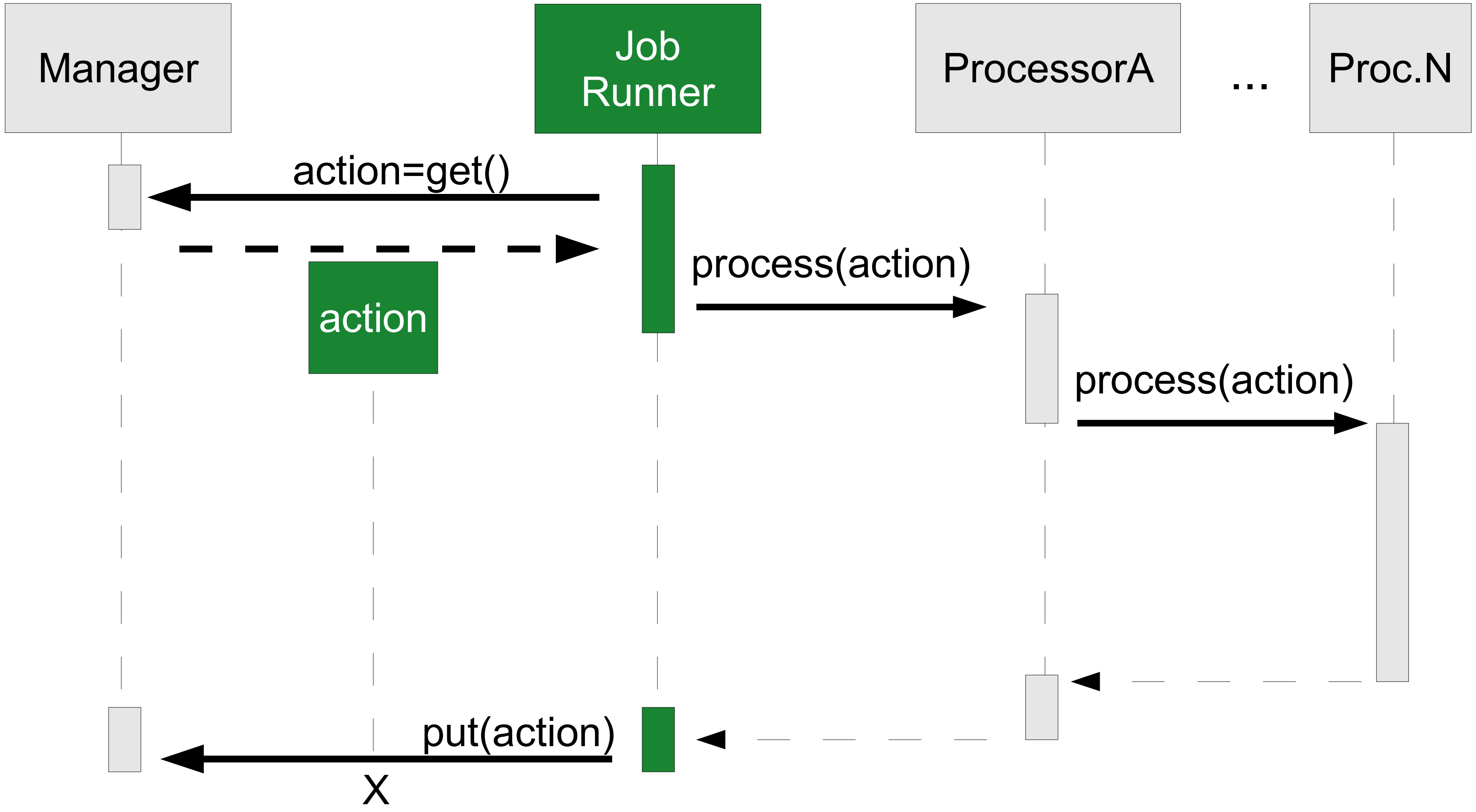
Action processing



Actions

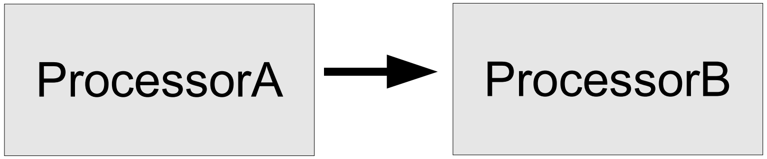
- Actions are the things that the XNJS processes
  - major pieces of information
    - activity description (any XML)
    - status
    - unique ID
    - Client (user and security information)
- new action types can be added easily
    - add code to process the action
    - re-configure the XNJS
- Example: Action of type „JSDL“

Processing basics



Flexible processing

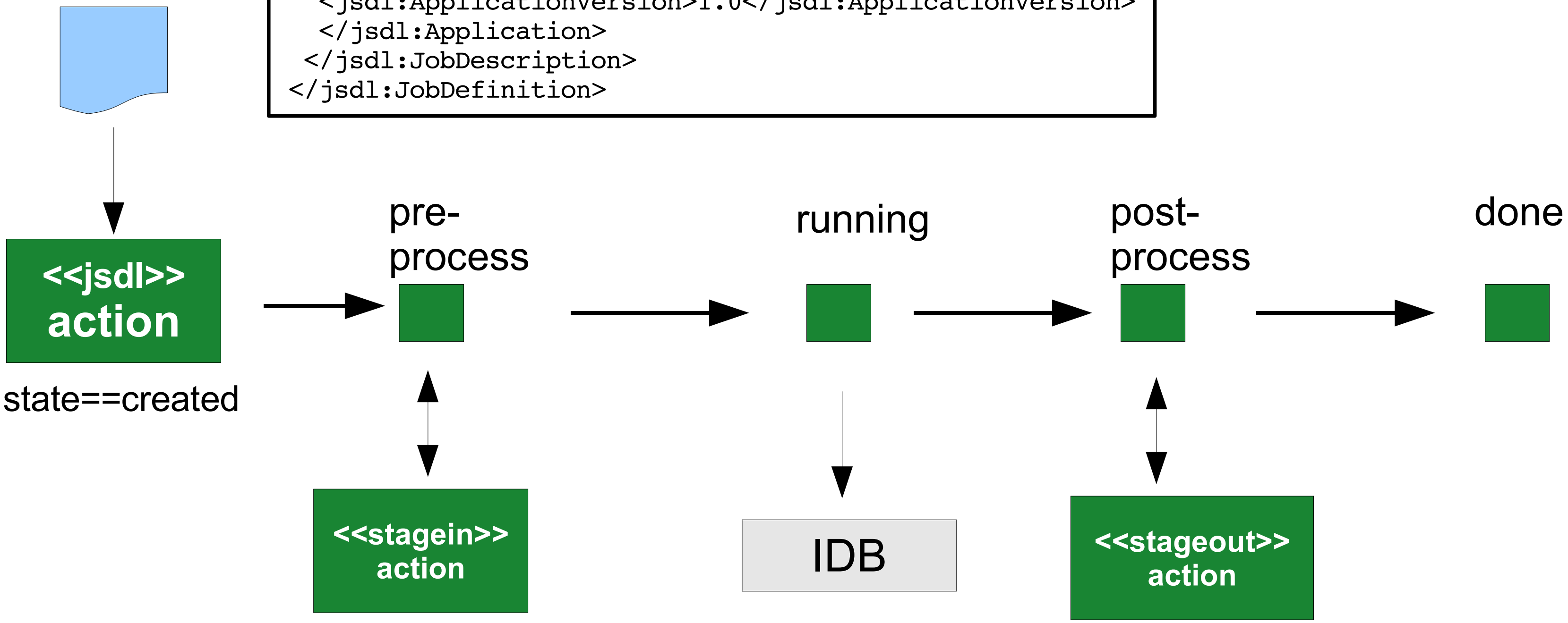
- processing chains are configurable per action type



- new action types can be added without changing the XNJS core
  - need to add new Processor implementations
  - edit config file
  - in principle even at runtime

example: JSDL processing

```
<jsdl:JobDefinition xmlns:jsdl="...">
 <jsdl:JobDescription>
  <jsdl:Application>
  <jsdl:ApplicationName>Date</jsdl:ApplicationName>
  <jsdl:ApplicationVersion>1.0</jsdl:ApplicationVersion>
  </jsdl:Application>
 </jsdl:JobDescription>
</jsdl:JobDefinition>
```



pluggable protocols:
-cp, http
-UNICORE 6

map Applications to executables
map JSDL resource requests to target system resources

Flexible processing

- Processors can be used for any activity within the XNJS
  - execution
  - filetransfer
  - logging, tracing, monitoring
  - notifications
  - third-party system integration
  - ...
- even workflow: workflow engine prototype for executing DAGs exists

Chemomentum in a nutshell...

- ... as seen from the „Grid“ point of view
- Take UNICORE 6 base services (job execution and storage)
- Build workflow processing on top
  - domain specific: „domain expert“ user
- Clients will be portals, web clients, standalone clients
- Main aims
  - scalable, well-performing (throughput, response times)
  - admin friendly, easy to install new nodes

XNJS usage scenario

- Users submit workflows to the Chemomentum workflow system, which results in many small jobs being submitted to the underlying Unicore 6 services

- Some numbers:
  - 10 users submitting 1 workflow each
  - 20 servers
  - 1 workflow = 1000 jobs
  - 10000 jobs
  - **500 jobs per server**

Demo

- submit 500 jobs to a single XNJS instance
- job characteristics
  - simple „Date“
- XNJS settings:
  - 128 MB for the VM
  - 20 worker threads
  - embedded Java TSI
  - HSQLDB embedded database for persistence
- measure
  - time for submitting the jobs
  - overall time needed

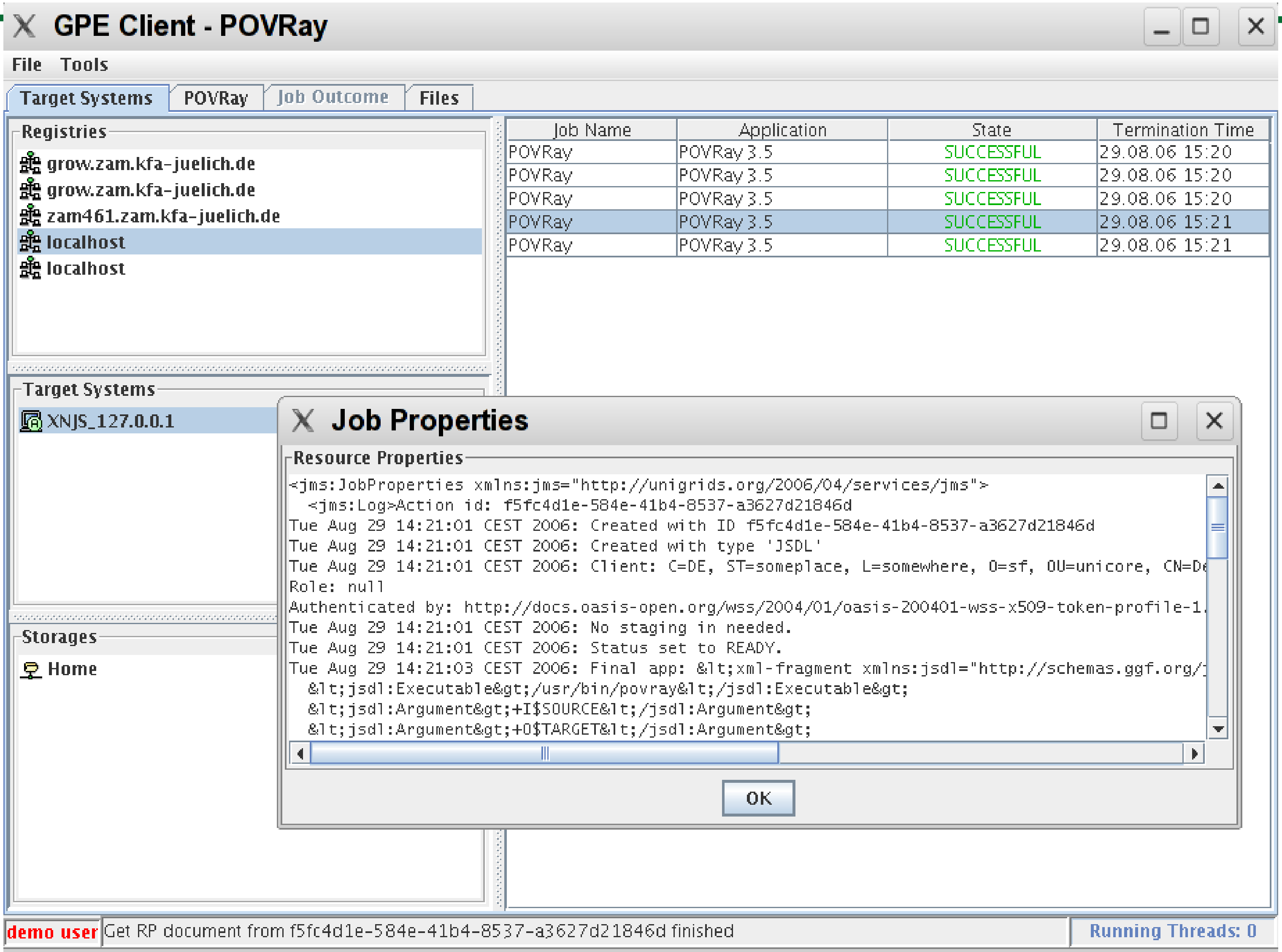
Some measurements...

Number of Jobs	100	500	1000	5000
Submission time [sec]	1	5	7	28
Submission rate [1/sec]	100	100	90	185
Overall time [sec]	8	35	71	331
Job rate [1/sec]	12	14	14	15

Tweaking possibilities

- very flexible engine, adaptable to the usage scenario
- can measure performance and optimize the „critical path“
- for example
  - use more workers (can add them at runtime)
  - tweak processing to decrease turnaround times:
    - e.g. use two identical processors per cycle
    - example: 4 processors, 500 jobs -> 55 jobs/sec
    - why? less database access, and less time spent in the queue

XNJS as UNICORE 6 backend



Registries:

- grow.zam.kfa-juelich.de
- grow.zam.kfa-juelich.de
- zam461.zam.kfa-juelich.de
- localhost
- localhost

Target Systems:

- XNJS_127.0.0.1

Storages:

- Home

Job Name	Application	State	Termination Time
POVRay	POVRay 3.5	SUCCESSFUL	29.08.06 15:20
POVRay	POVRay 3.5	SUCCESSFUL	29.08.06 15:20
POVRay	POVRay 3.5	SUCCESSFUL	29.08.06 15:20
POVRay	POVRay 3.5	SUCCESSFUL	29.08.06 15:21
POVRay	POVRay 3.5	SUCCESSFUL	29.08.06 15:21

Job Properties

Resource Properties

```
<jms:JobProperties xmlns:jms="http://unigrids.org/2006/04/services/jms">
  <jms:Log>Action id: f5fc4d1e-584e-41b4-8537-a3627d21846d
Tue Aug 29 14:21:01 CEST 2006: Created with ID f5fc4d1e-584e-41b4-8537-a3627d21846d
Tue Aug 29 14:21:01 CEST 2006: Created with type 'JSDL'
Tue Aug 29 14:21:01 CEST 2006: Client: C=DE, ST=someplace, L=somewhere, O=sf, OU=unicore, CN=De
Role: null
Authenticated by: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.
Tue Aug 29 14:21:01 CEST 2006: No staging in needed.
Tue Aug 29 14:21:01 CEST 2006: Status set to READY.
Tue Aug 29 14:21:03 CEST 2006: Final app: &lt;xml-fragment xmlns:jsdl="http://schemas.ggf.org/j
  &lt;jsdl:Executable&gt;/usr/bin/povray&lt;/jsdl:Executable&gt;
  &lt;jsdl:Argument&gt;+I$SOURCE&lt;/jsdl:Argument&gt;
  &lt;jsdl:Argument&gt;+O$TARGET&lt;/jsdl:Argument&gt;
```

demo user Get RP document from f5fc4d1e-584e-41b4-8537-a3627d21846d finished Running Threads: 0

XNJS as UNICORE 6 backend

- Web service frontend: UNICORE atomic services
- use XNJS instead of NJS as backend
- very promising, but...
- ... topic for a different talk!

Summary

- Achieved:
  - reconfigurable
  - extensible
  - flexible processing
  - scalable
  - embeddable
- Needs more work:
  - explicit business rules

Options for future work...

- support DRMAA TSI
- support for important OGSA specs
  - HPC profile
  - AuthZ
  - OGSA BES (more a front end issue)
- investigate options for realising „explicit business rules“

Conclusions

- Presented XNJS execution management system
- UNICORE concepts: Uspace, Applications, TSI
- Simple, high-performance core
- Modular, flexible, extensible, and highly scalable
- „native“ JSDL support
  - execution, data staging with pluggable protocols

- sound basis for future work
- **download it and try it**
  („experimental“ part of UNICORE 6 alpha)

Questions?