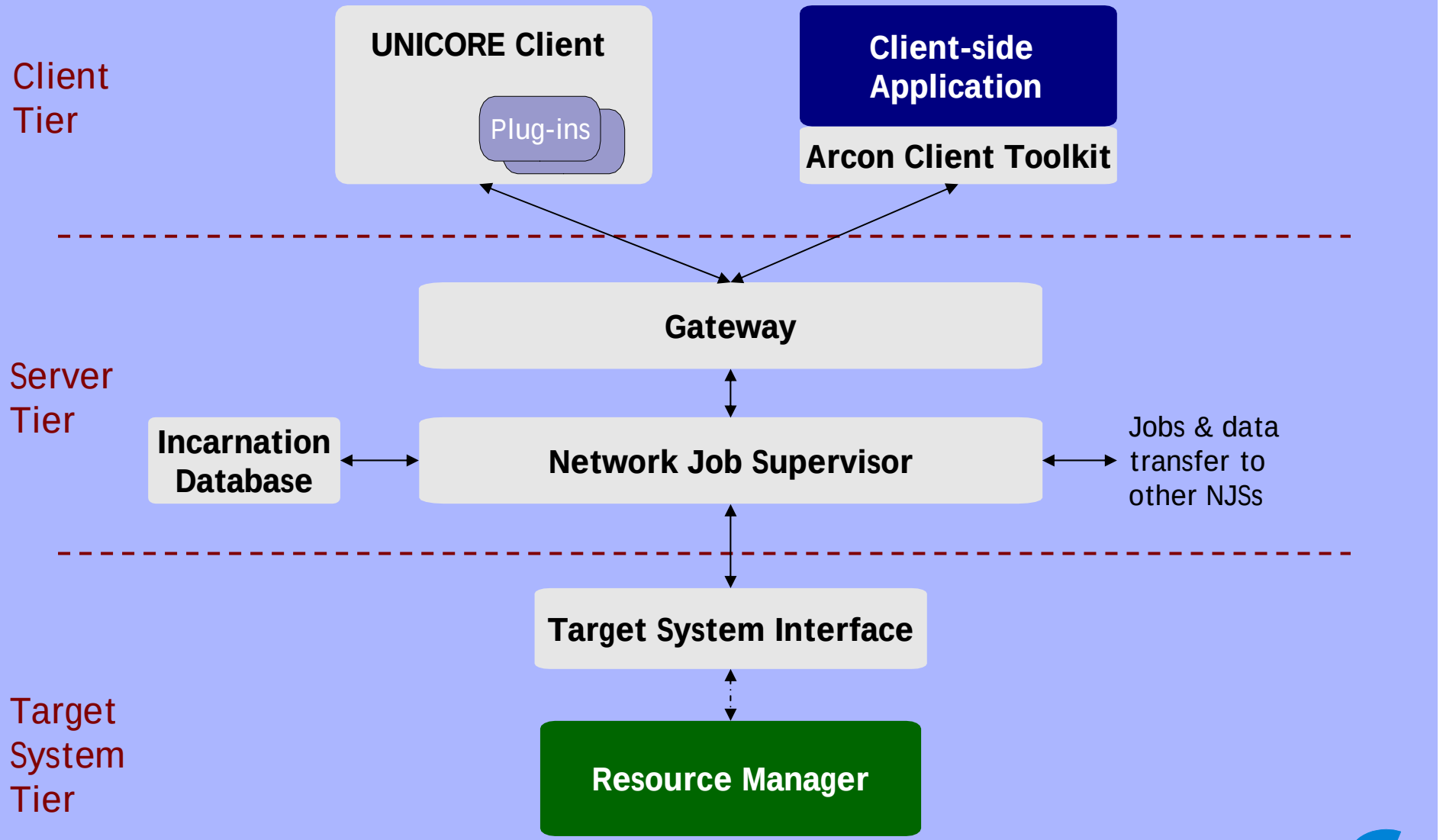
Unicore: Extended Application Support

Philipp Wieder
L. Kirtchakova, M. Romberg, B. Schuller
Research Centre Jülich
2nd International Summer School on Grid Computing
Vico Equense, Italy, July 28, 2004

Contents

- Motivation: Why extend Unicore?
  - And how and where?
- Real world examples
  - Interoperability
  - Domain-specific user support: “Meta-workflow”
  - Command line interface
- Resumé

Architecture revisited



Why extend Unicore?

Unicore = Uniform Interface to Computing Resources

Enhance it to e.g.

- add resources to Unicore (applications, computing systems, …)
- provide application-/domain-specific user support
- integrate organisation-specific security solutions
- implement new core functionality
- ...

How extend Unicore?

- Client customisation through plug-ins:
  - Integration of software & applications
  - Enhancement of client's functionality
- Server customization through specific TSIs:
  - Integration of new target systems (resources)
  - Support for client-side enhancements (e.g. interactive access)
- Client & Server extensions: Ask your developer!
  - This may change with Open Source Unicore

Hypothetical example

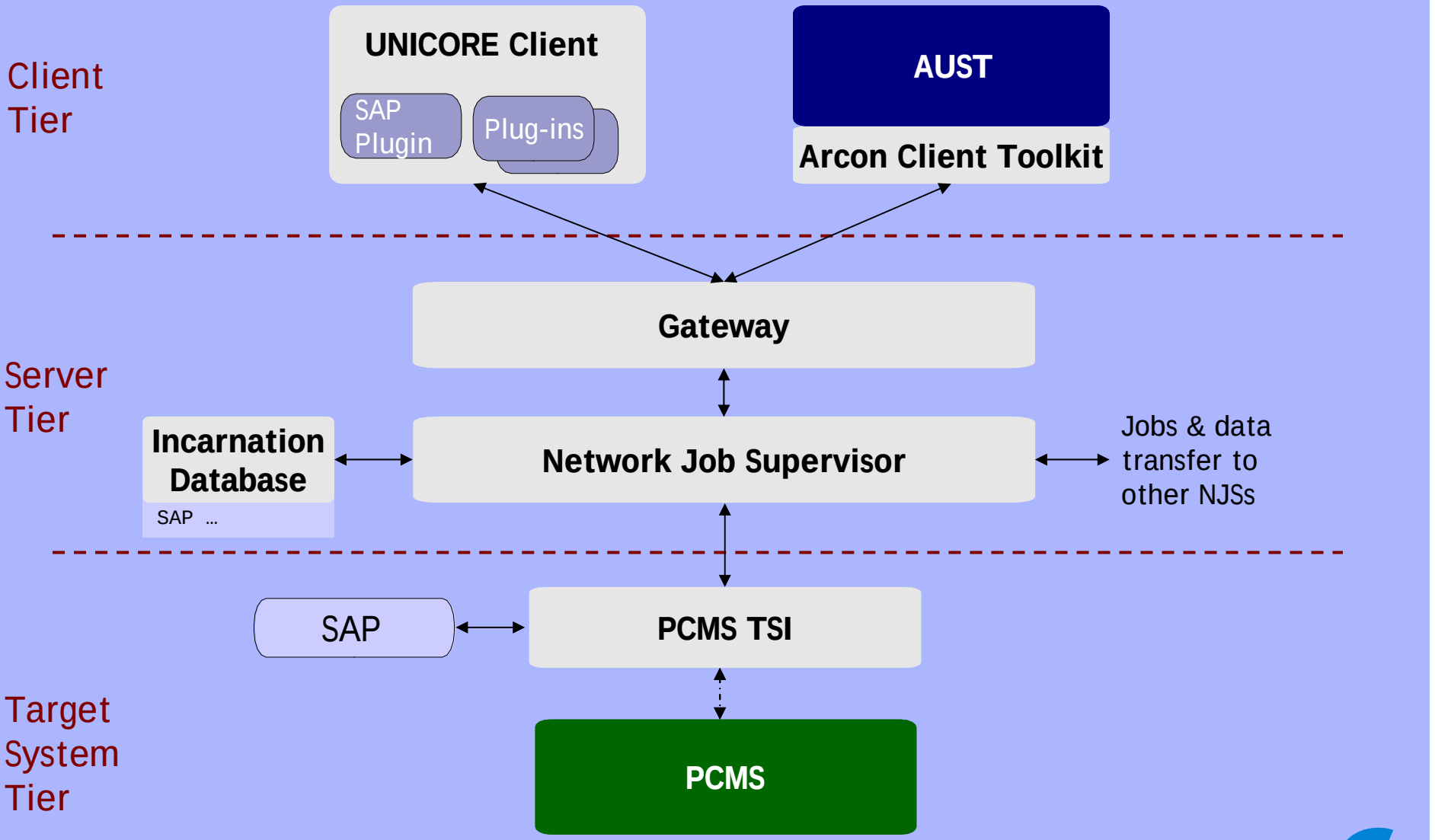
Integration of a development cluster

into a UNICORE Grid:

- Proprietary cluster management software (PCMS)
- Graphical user support for some application SAP
- Automated job submission and result evaluation for system testing (AUST)

... see next slide

Where extend UNICORE?



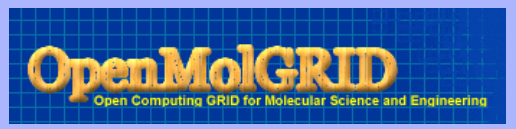
Real world scenarios

Scenario I: Interoperability

- Integration of Globus resources into Unicore
  - Unicore jobs on Globus resources
  - Integrated security solution
  - Cross-Grid resource brokerage

Scenario II: “Meta-Workflow”

- Simplify user access to a complex and dynamic Grid:
  - Improve support for complex workflows
  - Select resources automatically
  - Distribute computations to multiple Grid nodes



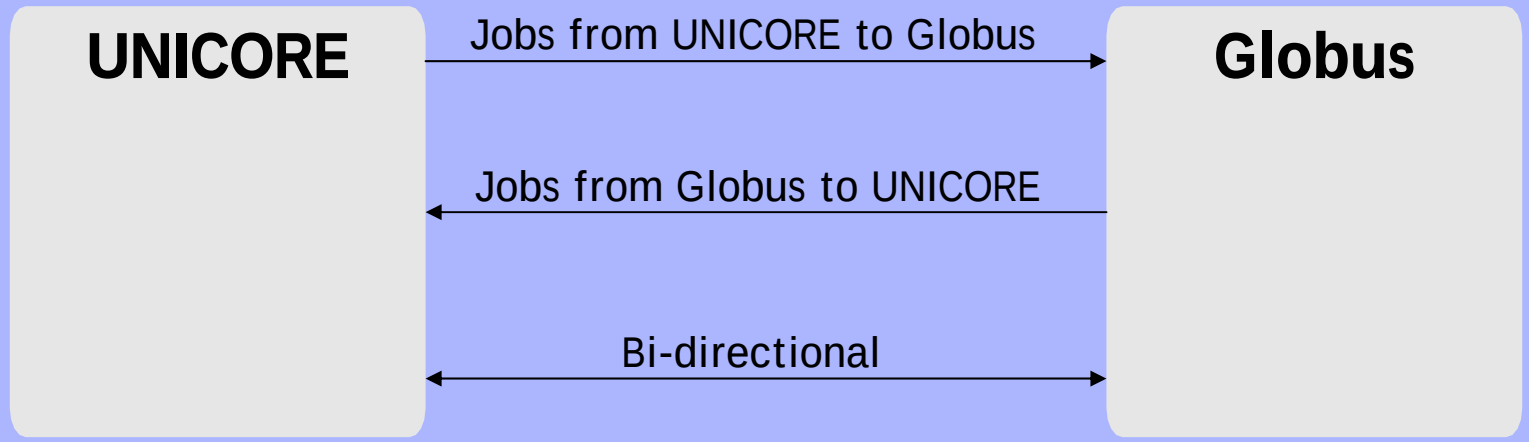
Scenario I: GRIP

... to realise the interoperability of UNICORE and Globus and to work towards standards for interoperability in the Global Grid Forum:

- Development of an interoperability layer between the two Grid systems
- Interoperable applications
- Contributions made to the Global Grid Forum
- UNICORE towards Grid Services

Requirements & usage scenarios

- Inter-Grid submission of jobs
- Status monitoring of jobs
- Output retrieval & data transfer
- Heterogeneous multi-site jobs
- Single-sign on
- Cross-Grid information services



Evaluation of requirements

- Grid approach
  - UNICORE: User oriented workflow environment
  - Globus: Services, APIs & portal builder
  - ⇨ UNICORE as a workflow portal for Globus
- Security
  - UNICORE: End-to-end security model
  - Globus: Requires transitive trust
  - ⇨ Don't violate UNICORE's security model
- Resource description
  - UNICORE: One model for discovery & request
  - Globus: Different models
  - ⇨ Description mapping for discovery & request

Design decisions

- Use UNICORE's sophisticated job & workflow model

  ⇨ Submit jobs from UNICORE to Globus

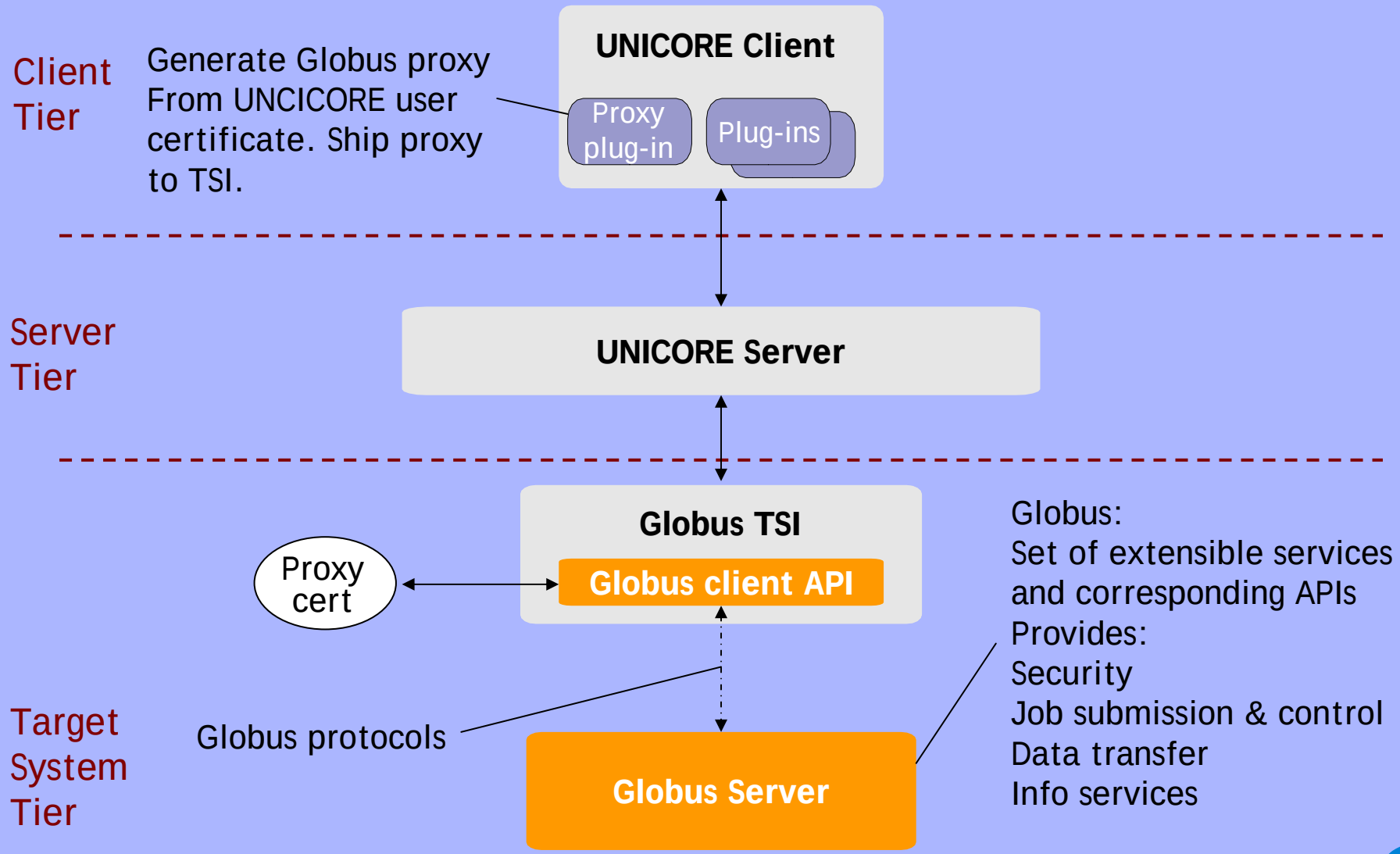
- Provide single sign-on without violating UNICORE's security model

  ⇨ Generate temporary Globus credentials (proxies) in the UNICORE client

- Postpone final integration of information services

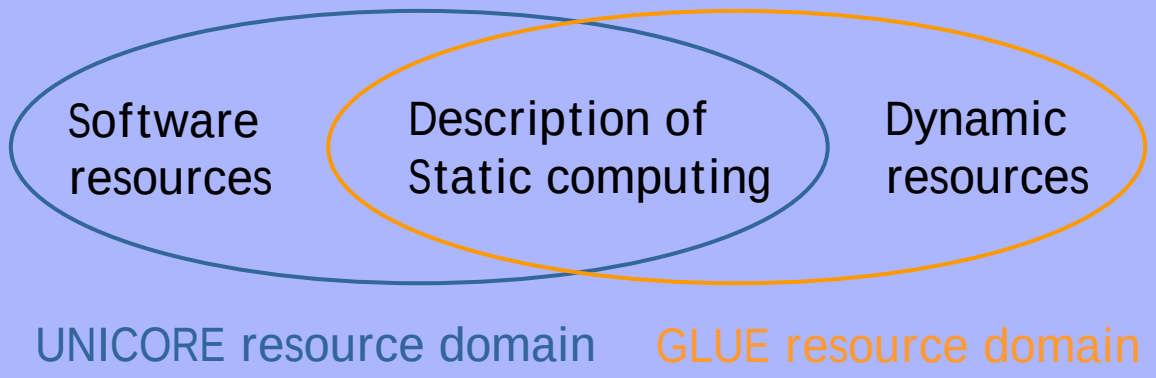
  ⇨ Use semantic techniques to broker across Grids

Interoperability architecture



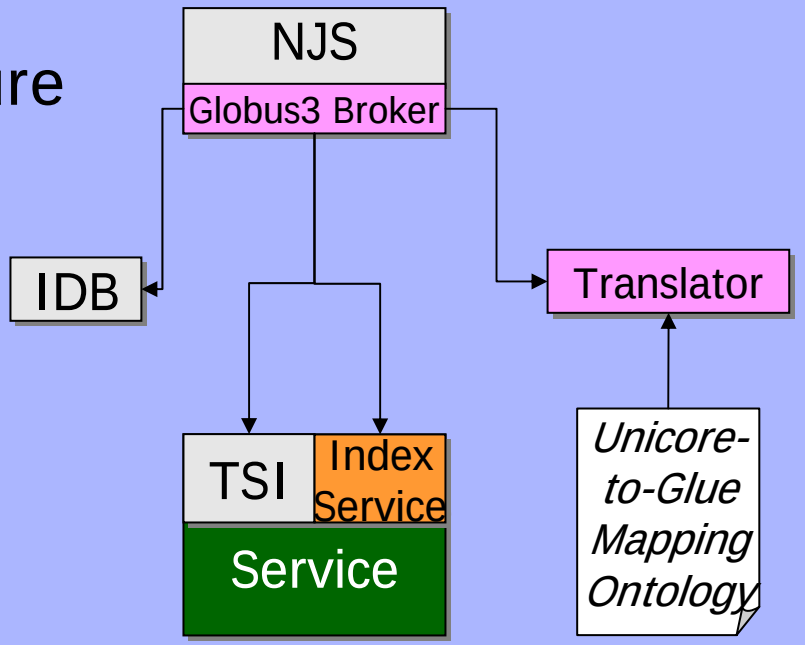
Resource models: Comparison

- UNICORE models user requests:
  - Maximum of a resource that is available (policy restrictions!)
  - Is a software available?
- Globus/GLUE provides machine meta-data model
  - Physical resources available
  - Dynamic capabilities of resources



Cross-Grid resource broker

- Ontology to translate resource descriptions automatically
- Extensible broker architecture
- Experimental brokering Web Service
- Standardised Grid resource description/ontology essential



Donal Fellows, University of Manchester

Lessons learned

Realization of uni-directional interoperability UNICORE -> Globus is technically solved, but:

- Not sensible to add an interoperability layer for every non-UNICORE Grid system
- Small intersection of UNICORE & Globus resource models
- Widely accepted standards are crucial

Broader vision of interoperability: Not just interoperability with Globus, not just “static” interoperability

Scenario II: OpenMolGRID

... to support the molecular design & engineering process of drug design through automating the following tasks

- data warehousing
- data mining
- molecular engineering

Unicore as the underlying Grid middleware is used to integrate the various applications & to add domain-specific workflow support

Scenario II: Why Meta-Workflow?

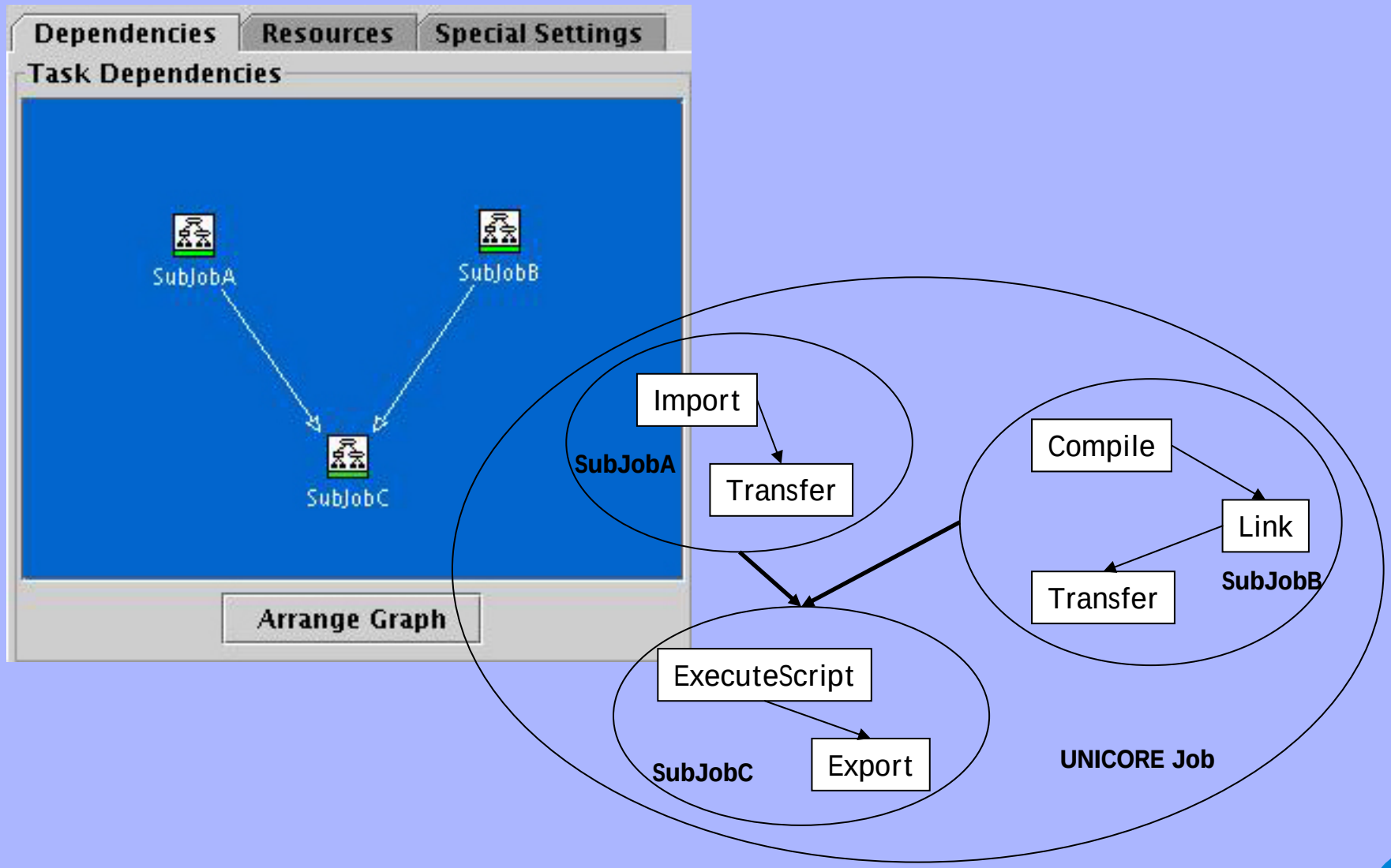
Requirements:

- **Automated** workflow support
- **Automated** resource management
- Application support

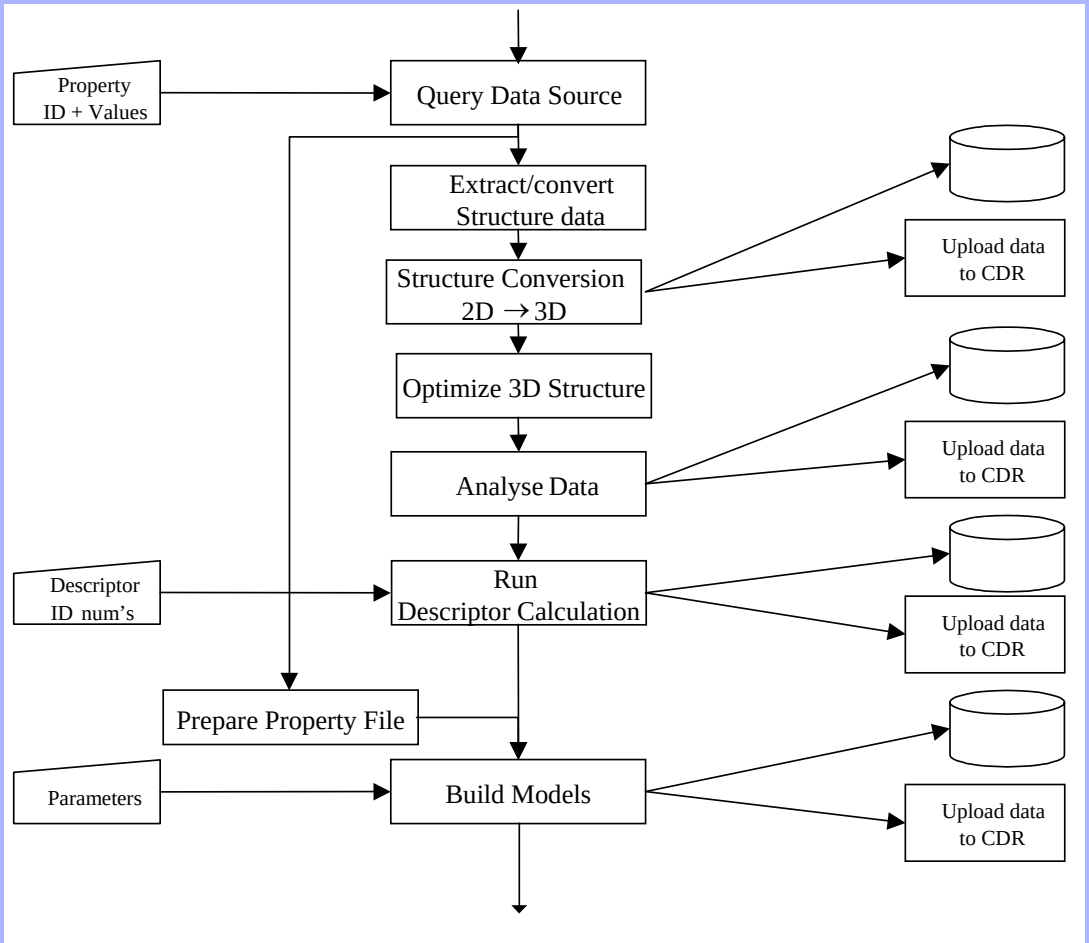
... based on core UNICORE services and functions

So, why has UNICORE to be extended?

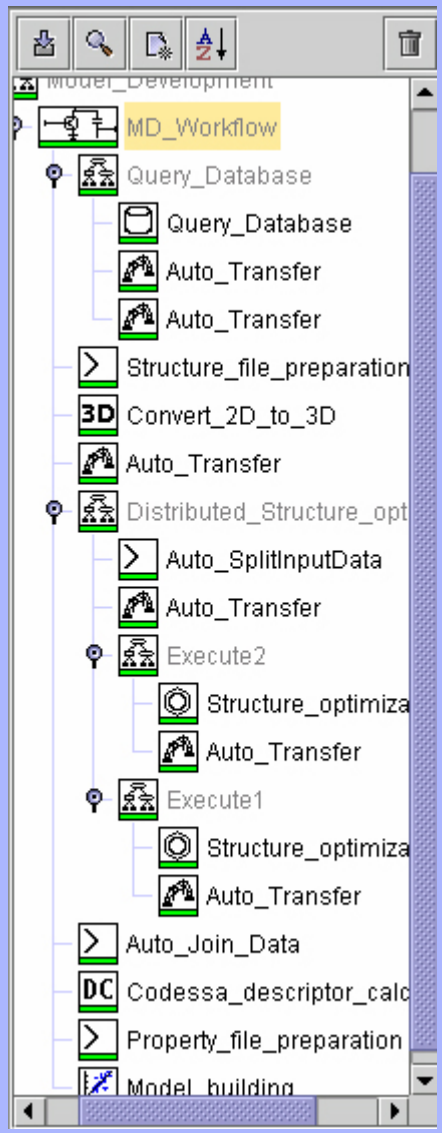
UNICORE workflow model



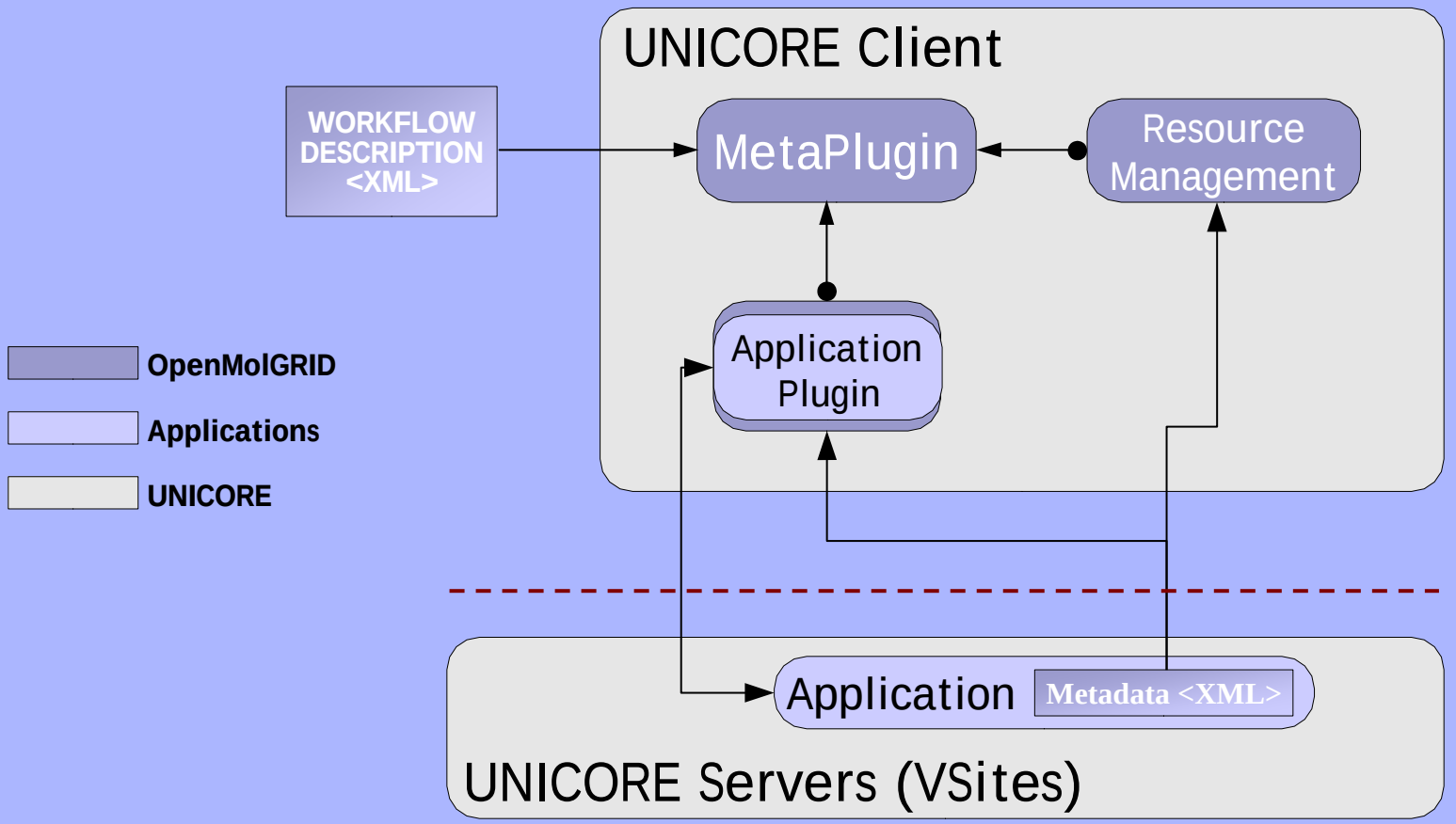
OpenMolGRID example workflow



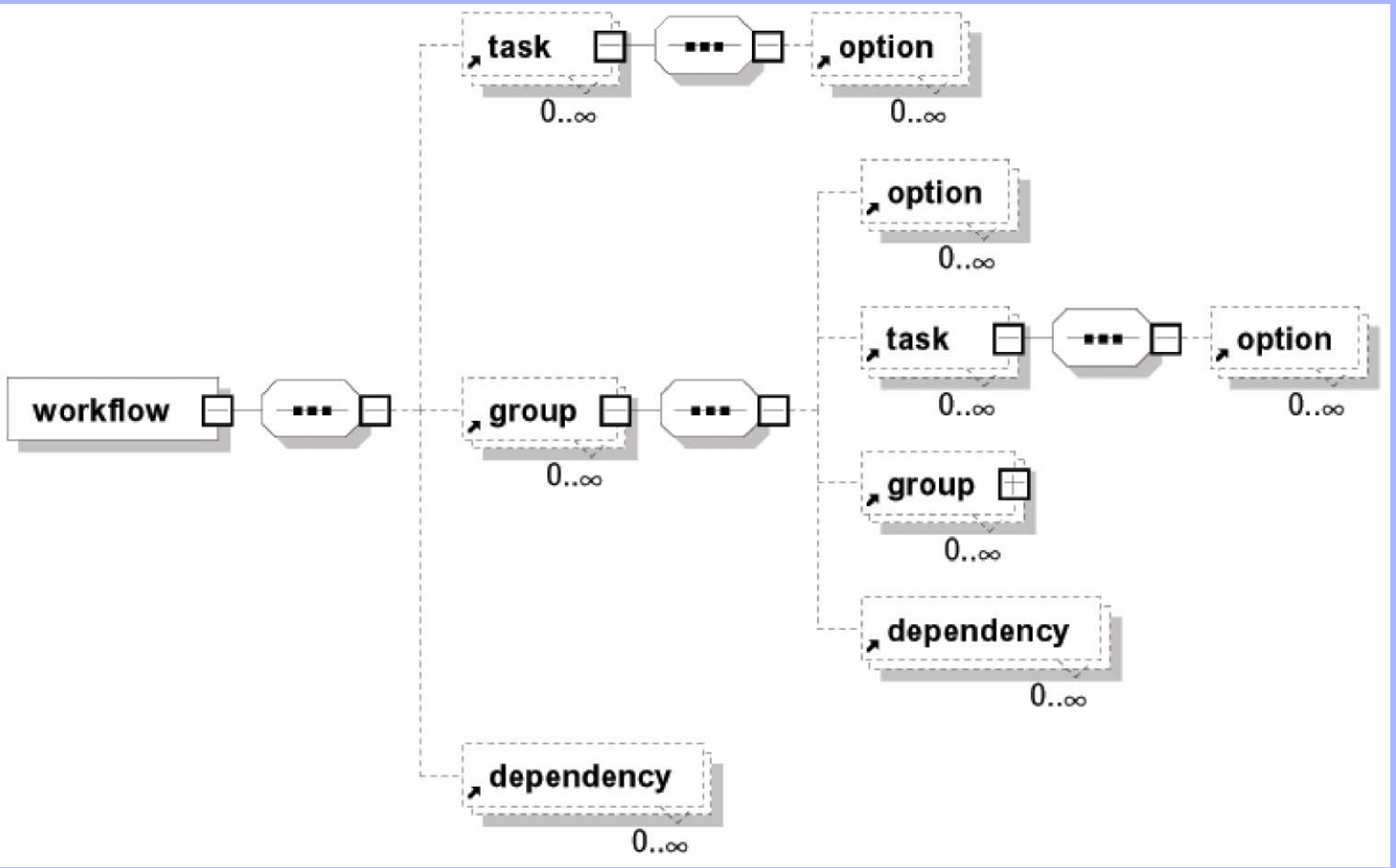
?



Workflow support: Realisation



Workflow description



Workflow XML schema

```
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
+ <xs:element name="workflow">
- <xs:element name="task">
- <xs:complexType>
+ <xs:sequence>
<xs:attribute name="name" use="required" />
<xs:attribute name="identifier" use="required" />
<xs:attribute name="id" use="required" />
- <xs:attribute name="export" use="required">
+ <xs:simpleType>
</xs:attribute>
- <xs:attribute name="split" use="required">
+ <xs:simpleType>
</xs:attribute>
</xs:complexType>
</xs:element>
+ <xs:element name="group">
- <xs:element name="dependency">
- <xs:complexType>
<xs:attribute name="pred" use="required" />
<xs:attribute name="succ" use="required" />
</xs:complexType>
</xs:element>
+ <xs:element name="option">
</xs:schema>
```


MetaPlugin: Overview

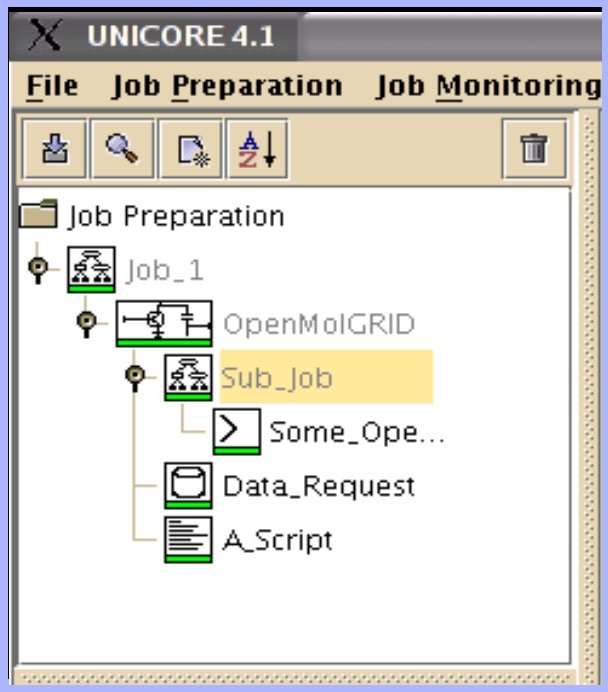
Workflow description

Server Resources (Applications, compute resources, storage, ...)

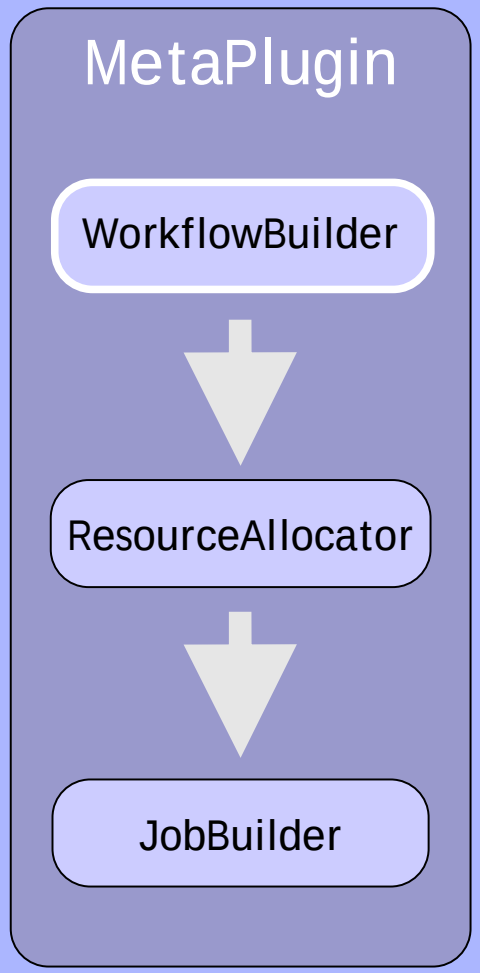
Client Resources (Plugins, resource management, ...)

User Preferences

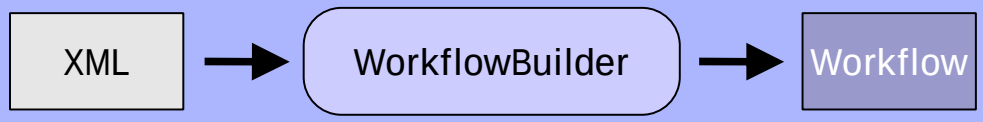
OpenMolGRID Workflow Support components



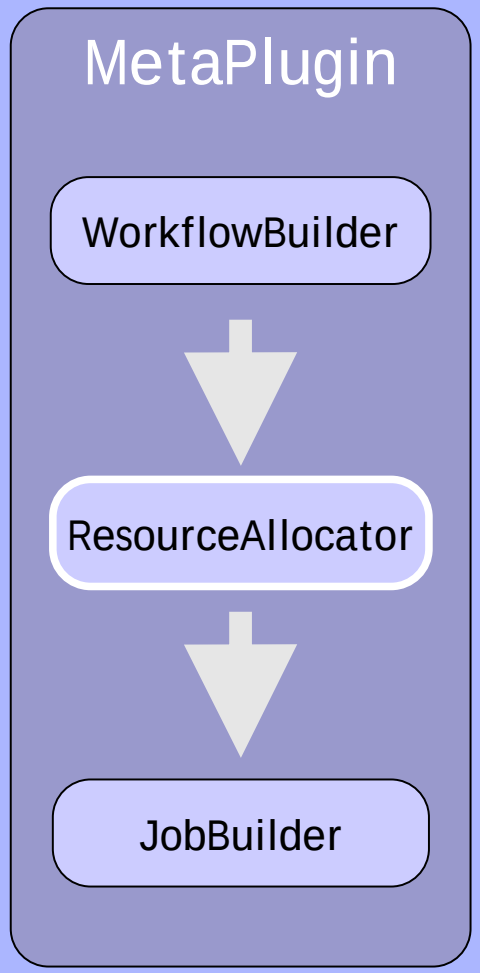
MetaPlugin: Workflow builder



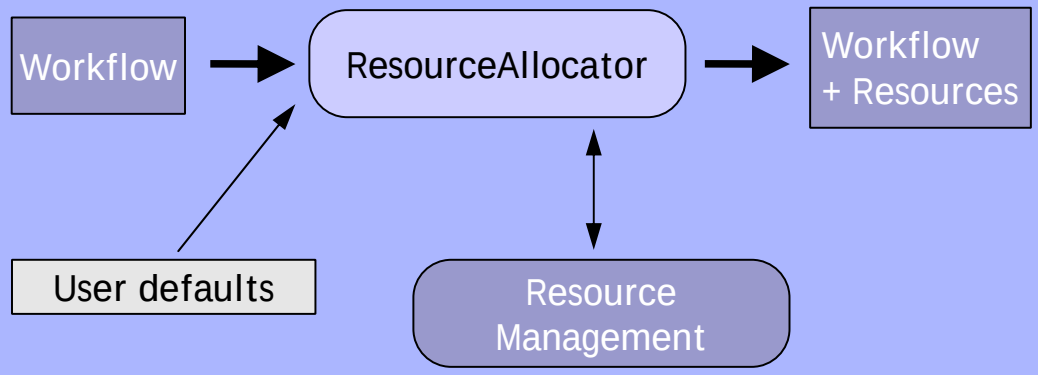
- WorkflowBuilder
  - Read workflow XML file
  - Build workflow object



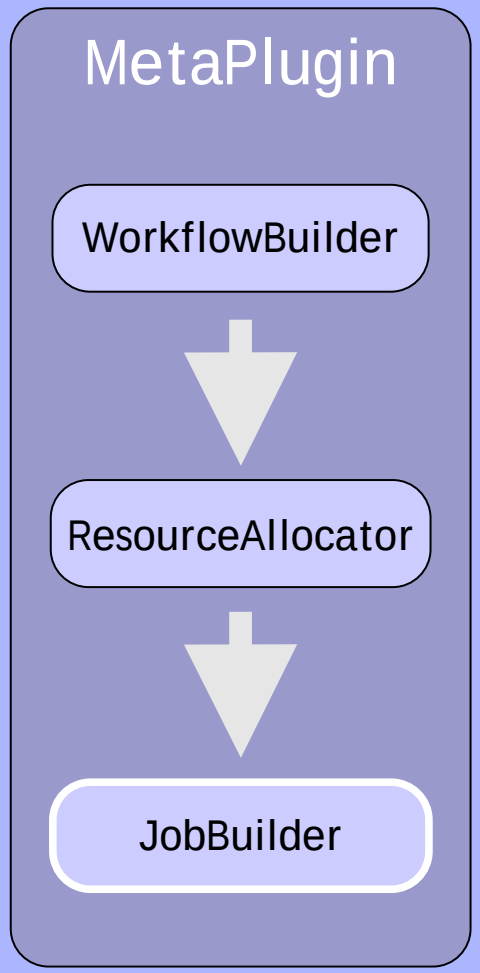
MetaPlugin: Resource Allocator



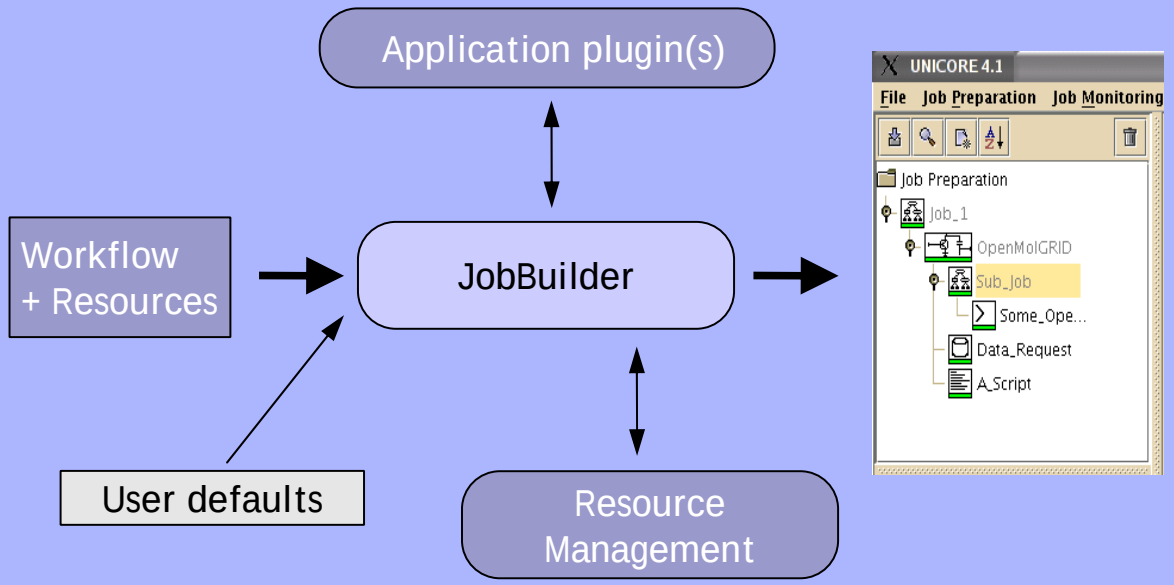
- ResourceAllocator
  - Find resources
  - Take user defaults into account
  - Attach them to the workflow



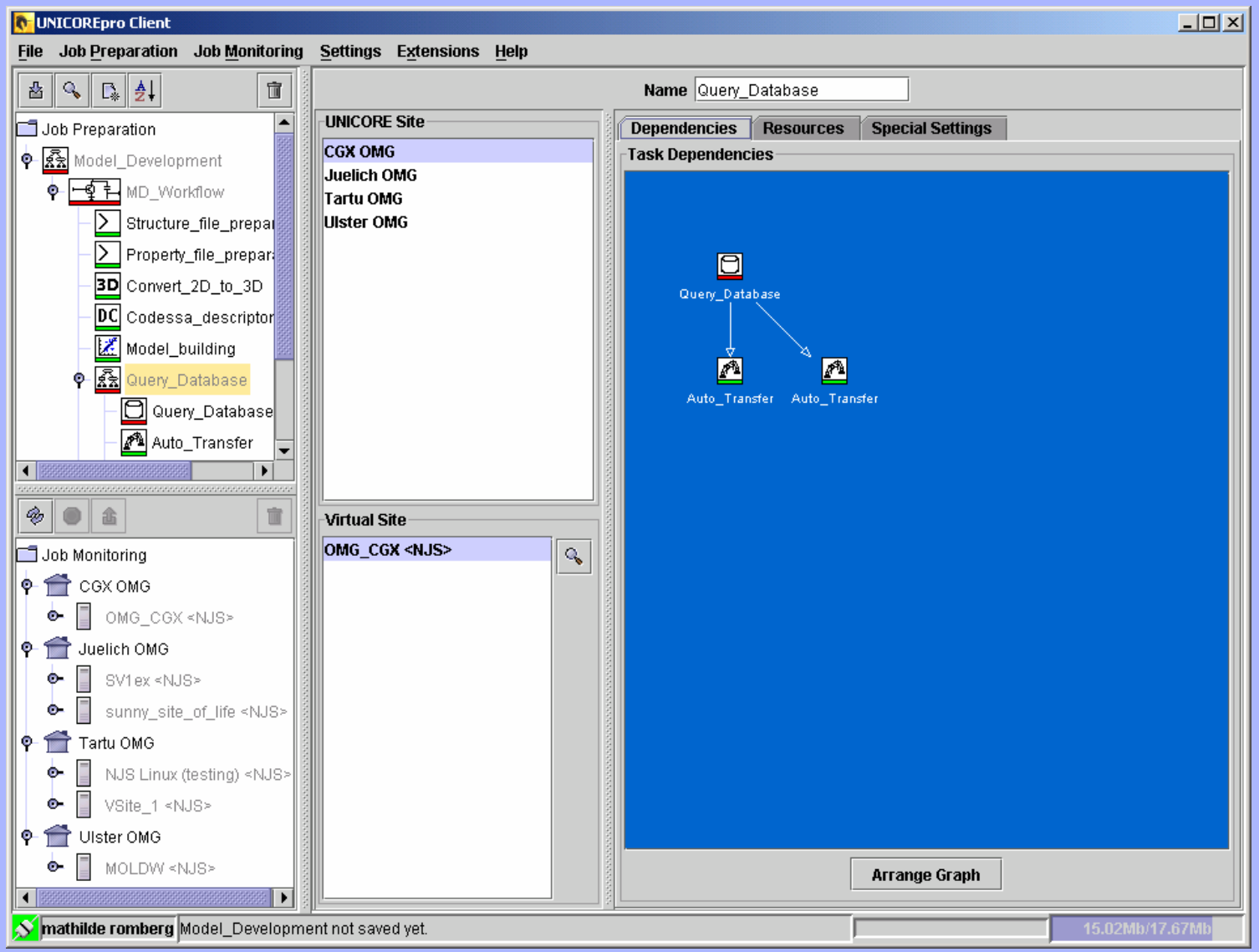
MetaPlugin: Job builder



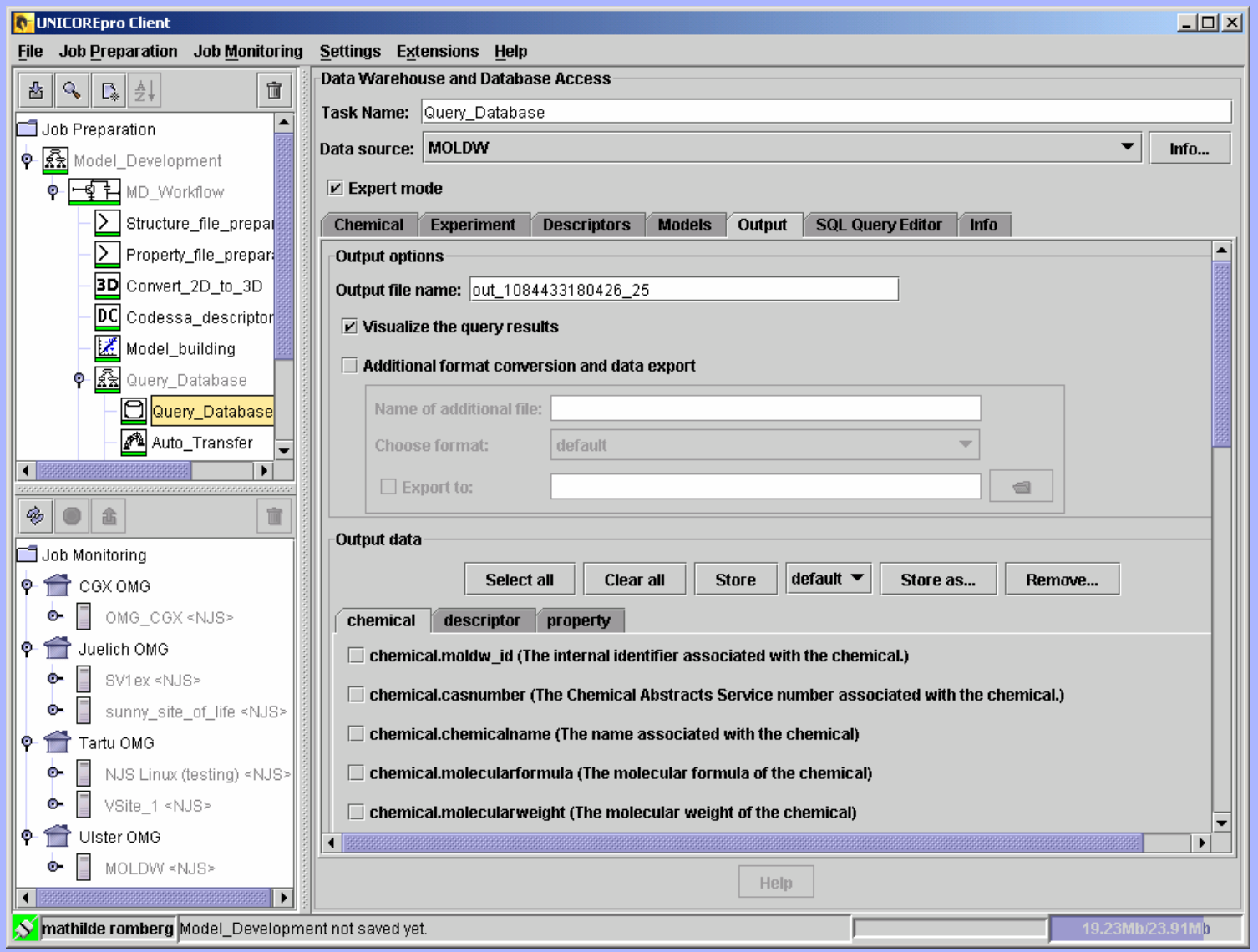
- JobBuilder
  - Map workflow onto Unicore job
  - Set filenames, options … (user interaction)
  - Perform data distribution



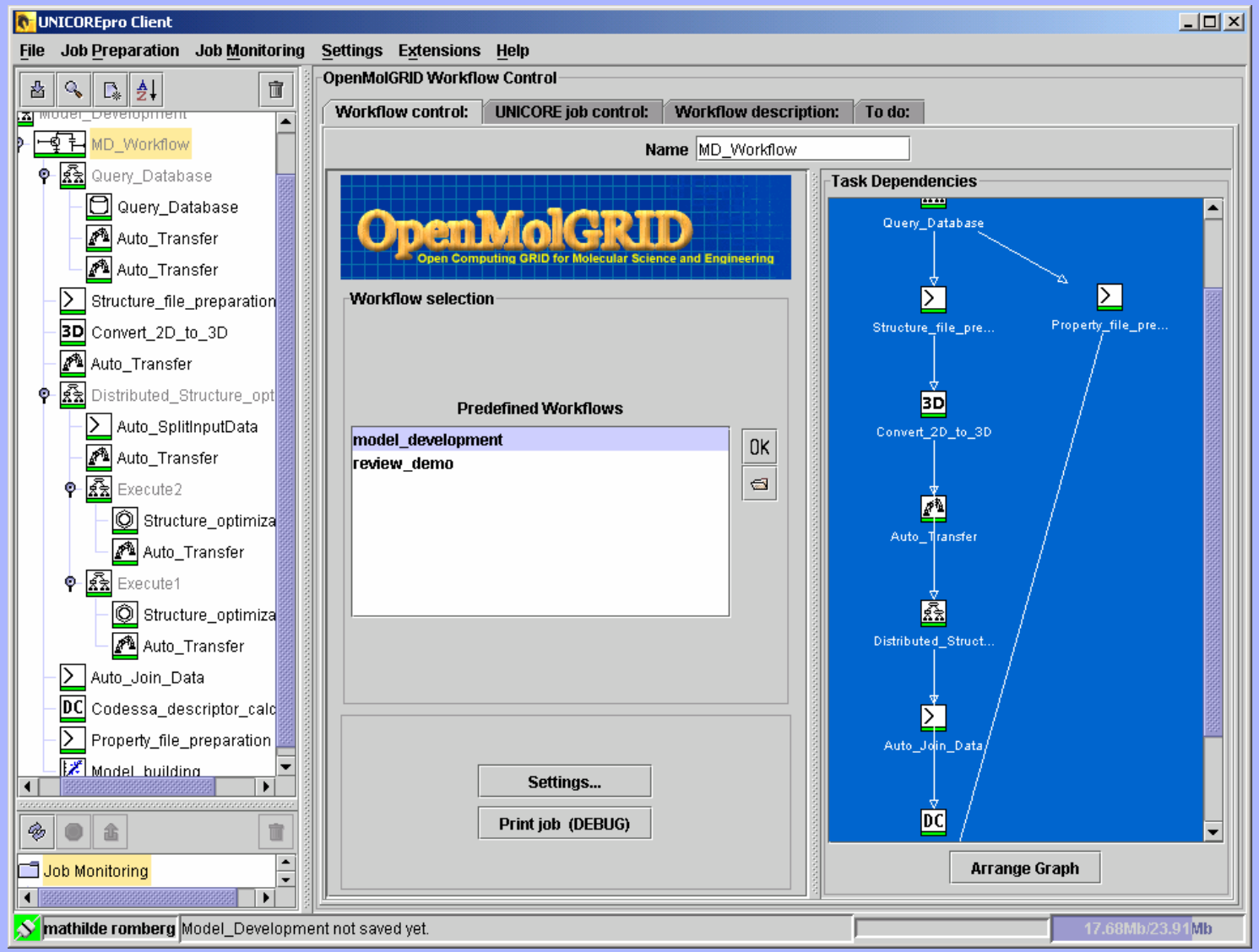
MetaPlugin: workflow adjustment



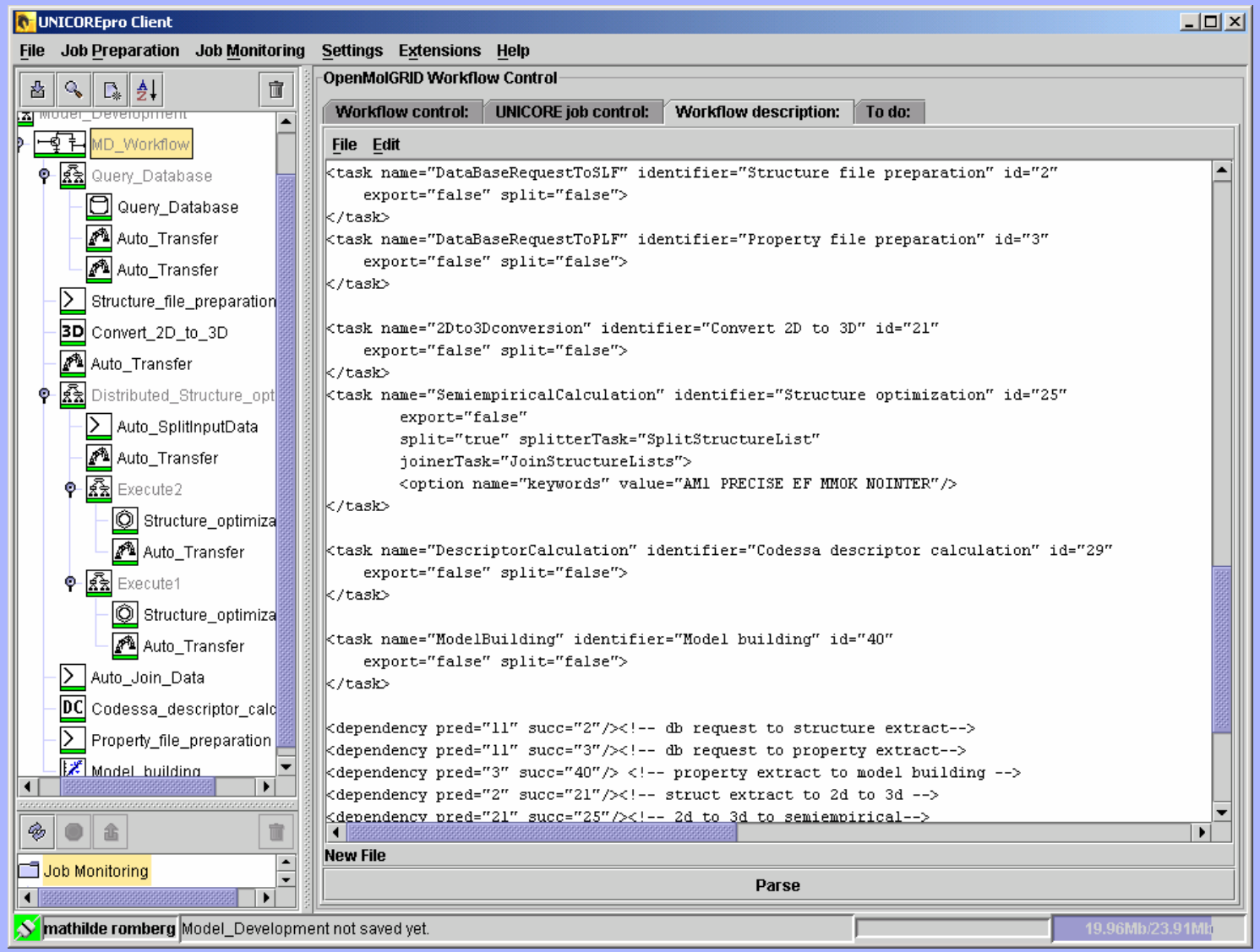
MetaPlugin: workflow adjustment (cont)



MetaPlugin: workflow control panel



MetaPlugin: Workflow description



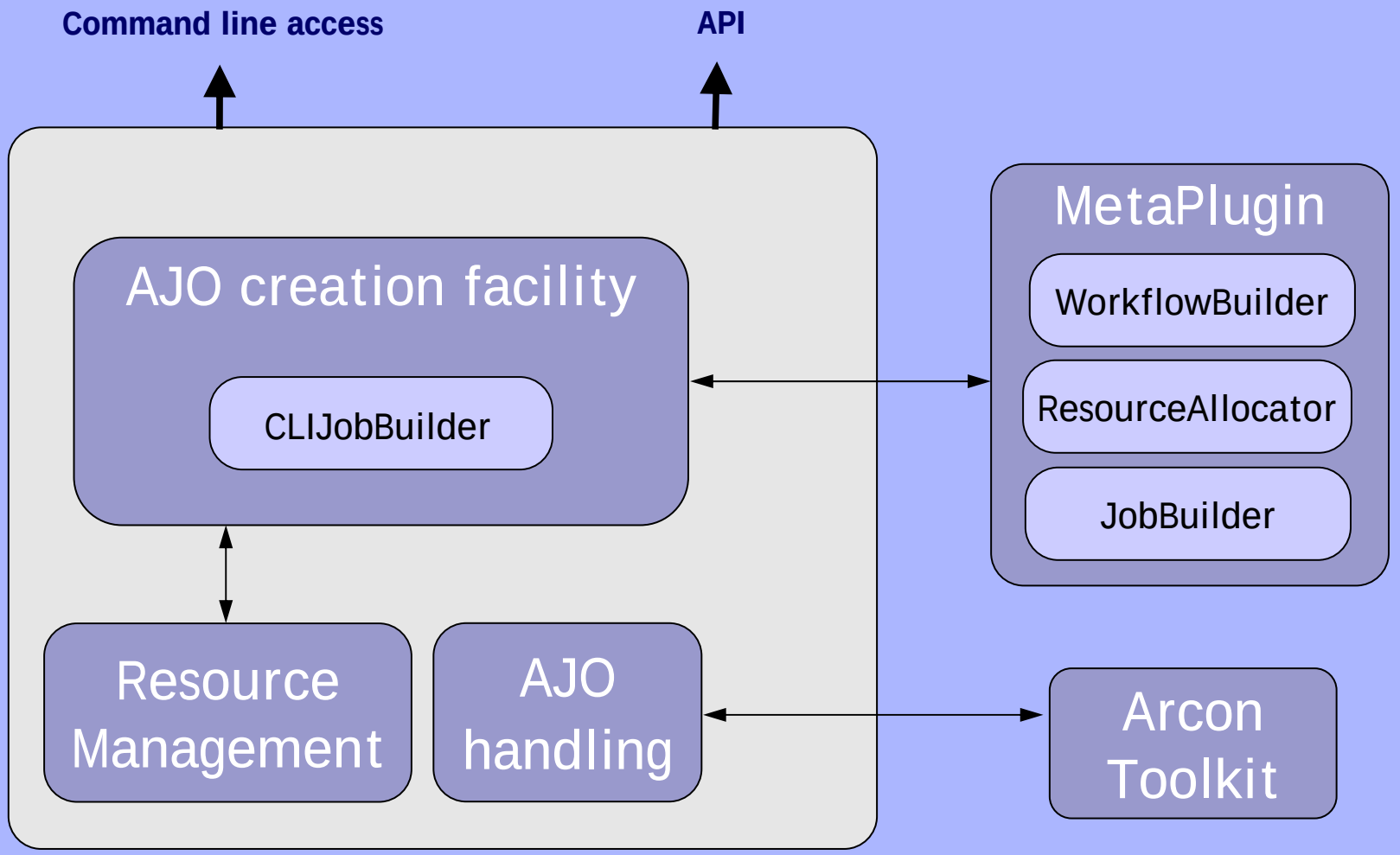
Application support

- Use case: “Non-GUI” workflow support within applications
- Requirements: Re-use of existing code
- Solution: Command line interface based on the MetaPlugin
- Two versions:
  - Command line interface: Rapid testing & widely applicable
  - Command line API: Integration into Java applications

Command Line Interface (CLI)

- Provide access to Unicore from within applications
- Build & submit jobs without user interaction (unlike GUI client)
- All commands use certificates for authentication
- All commands executed per job, queuing module supports batch processing
- CLI (& API) completely realised in Java

CLI: Architecture



CLI: Functions

- **`build_ajo`**: Builds AJO from XML workflow & saves it to file
- **`submit`**: Reads an AJO from file & submits it to Vsite (synchronous/asynchronous -> AJO_ID to identify job)
- **`get_status`**: Get status for job with AJO_ID
- **`get_outcome`**: Get result files for job with AJO_ID
- **`abort_job`**: Abort job identified by AJO_ID & remove it from NJS

Command Line API

- Collects required resources at start-up and caches them internally
- Prevents re-initializing of all resources for every job generation request
- Enables subsequent command execution without restarting CLI

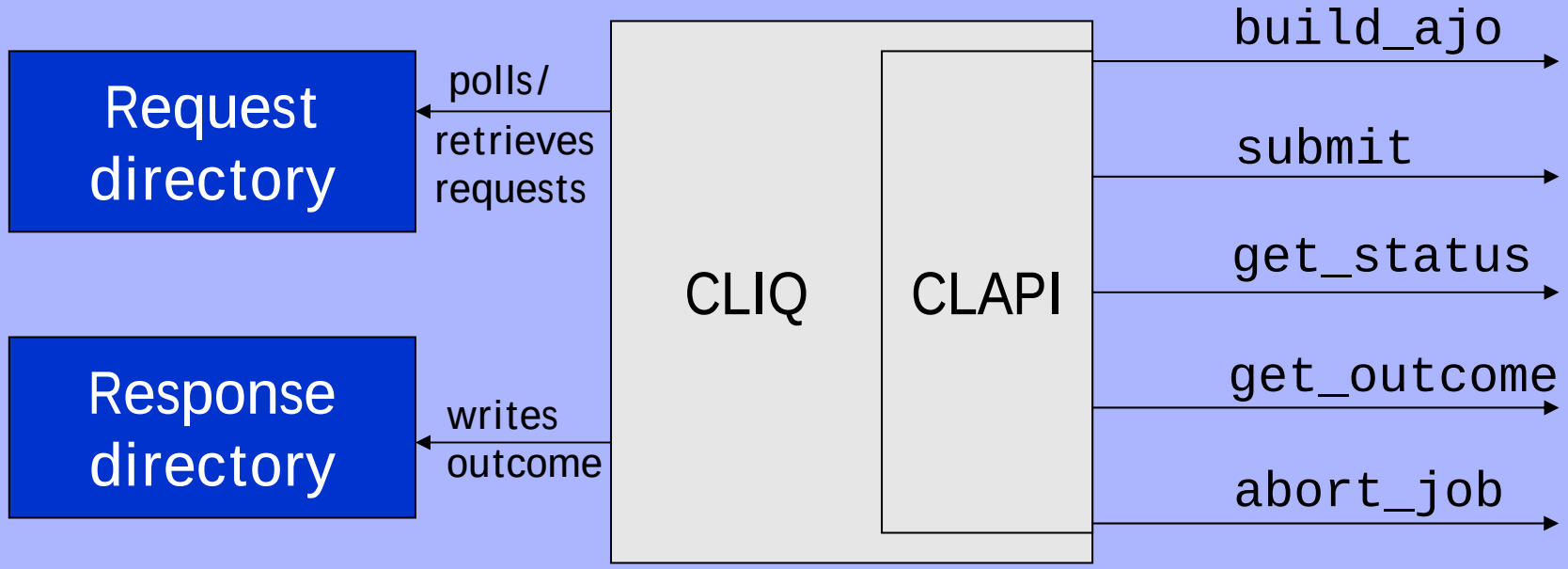
CLI: Example usage

CLI (from command line)	CLAPI (from Java application)

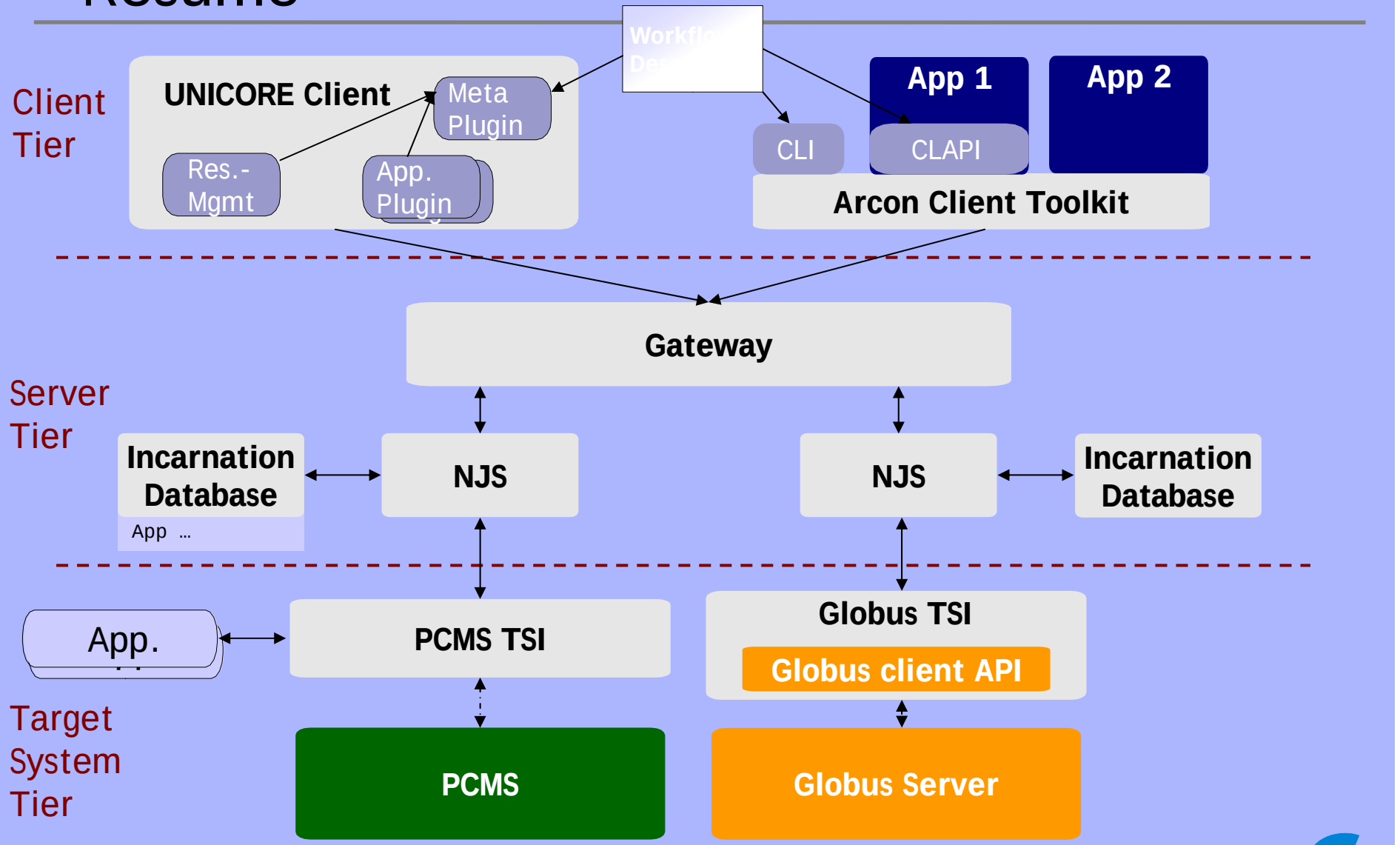
`$>cli build_ajo` ***`[options]`***
`$>cli submit` ***`[options]`***
	`cli = new CLAPI();`
	`cli.execute("build_ajo",`
	`[options]``);`
	`cli.execute("submit",`
	`[options]``);`

CLI Queue

- Automatic successive processing of multiple jobs



Resumé


Client Tier
UNICORE Client
Meta Plugin
Res.-Mgmt
App. Plugin
CLI
App 1
CLAPI
App 2
Arcon Client Toolkit
Gateway
Server Tier
Incarnation Database
App ...
NJS
NJS
Incarnation Database
App.
PCMS TSI
Globus TSI
Globus client API
Target System Tier
PCMS
Globus Server

Recommended reading

- Unicore Plus final report (good intro to Unicore): http://www.unicore.org/documents/UNICOREPlus-Final-Report.pdf
- GRIP: http://www.grid-interoperability.org
- OpenMolGRID: http://www.openmolgrid.org
- Sources, docs, etc. at https://unicore.sourceforge.net/