

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

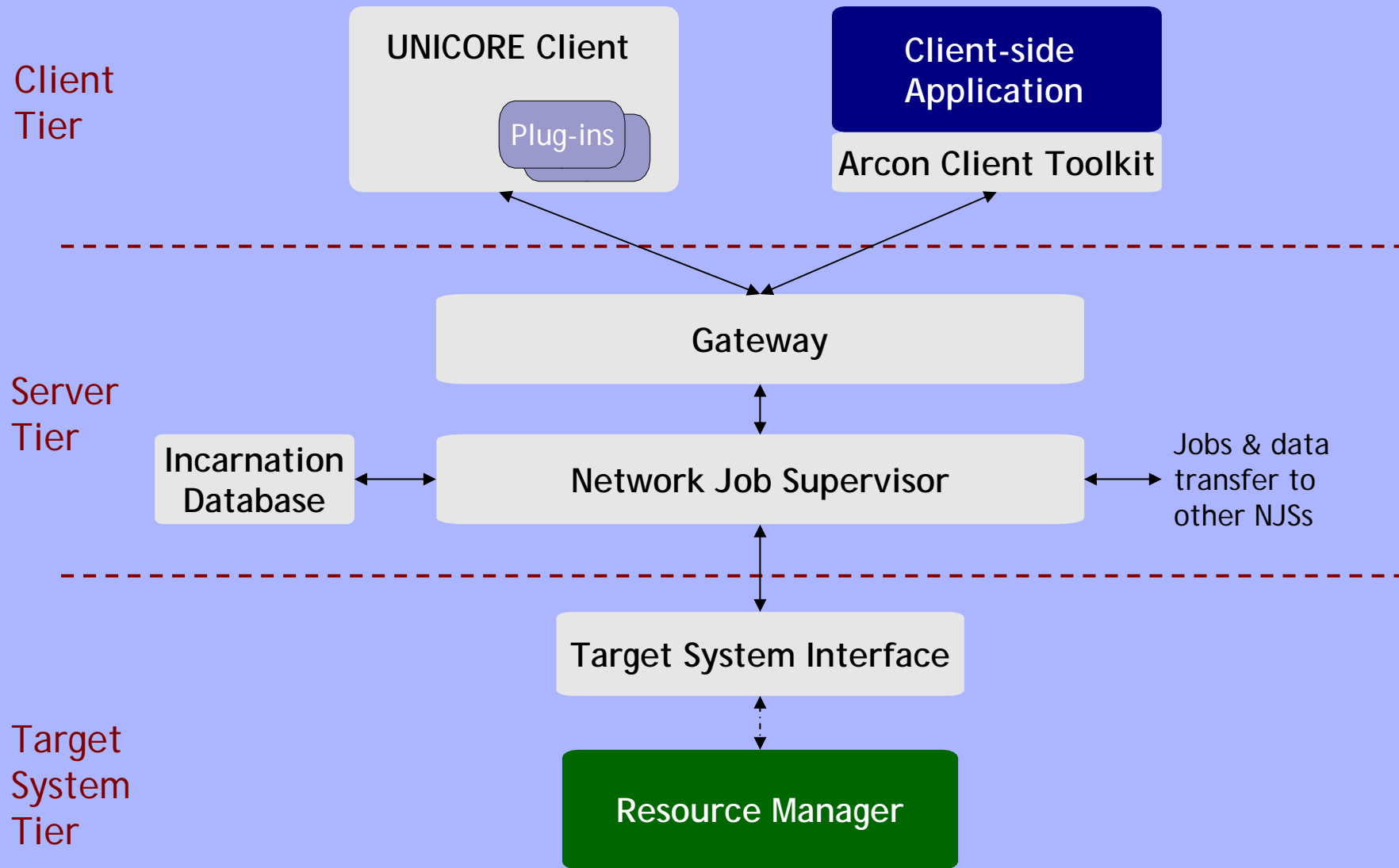


Contents

- 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

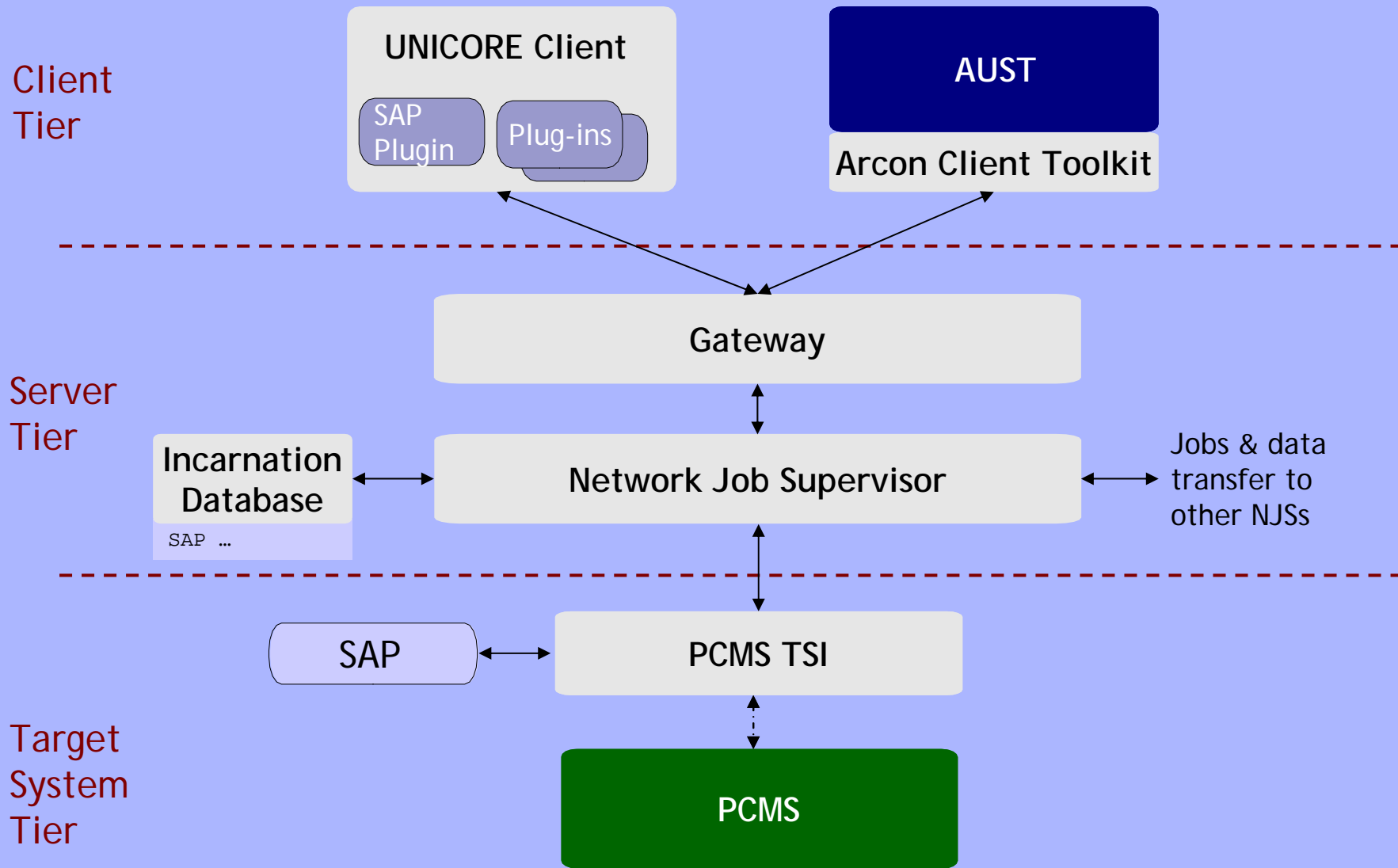
Integration of a development cluster
into a UNICORE Grid:

- 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

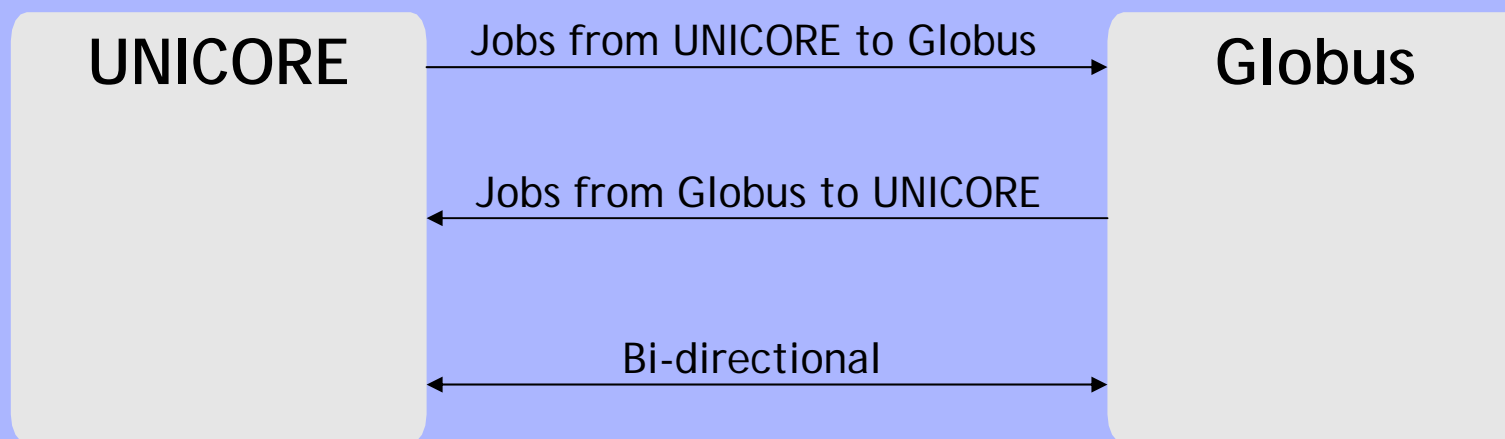
... to realise the interoperability of UNICORE and Globus and to work towards standards for interoperability in the Global Grid Forum:

- 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

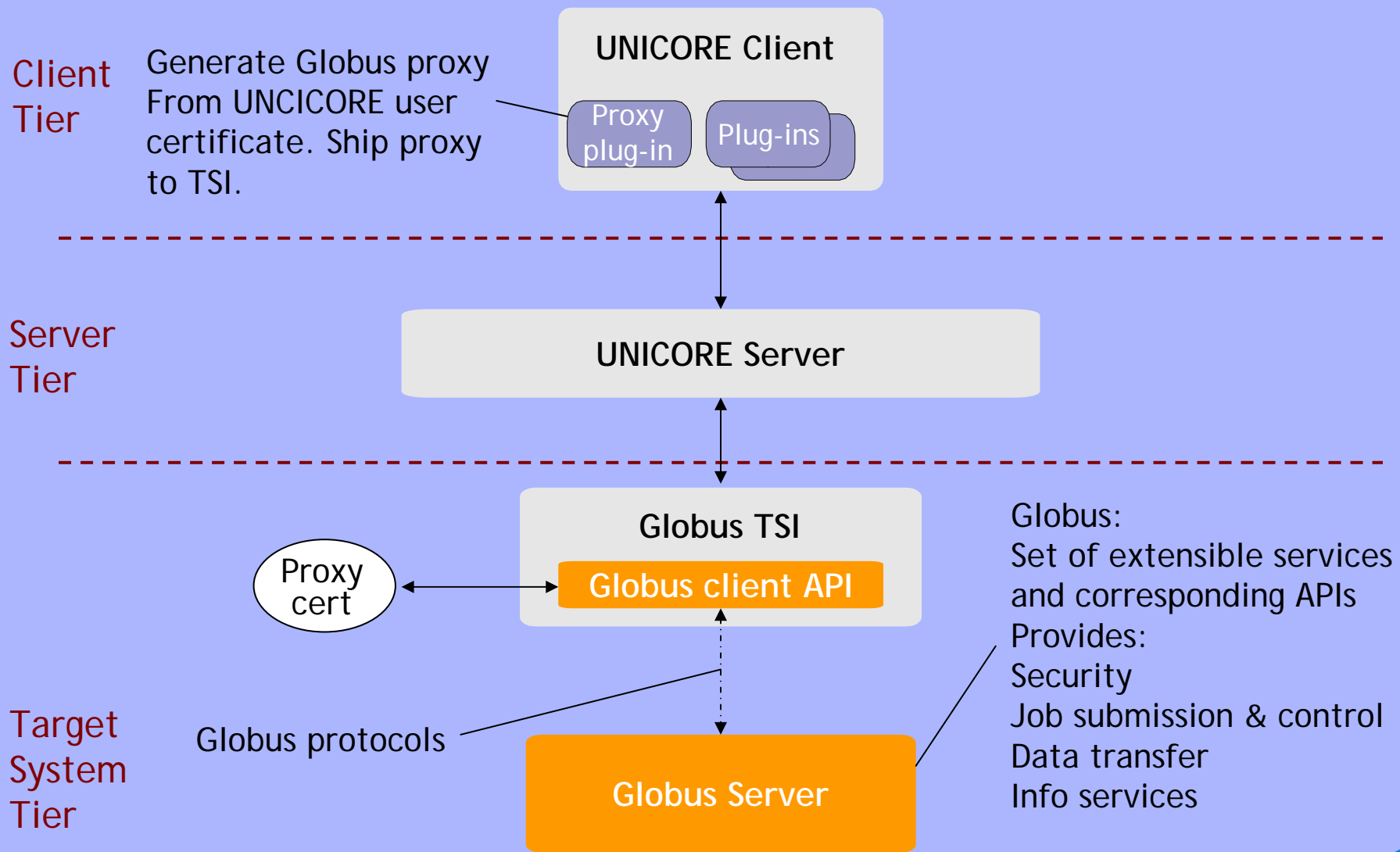


Design decisions

- 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

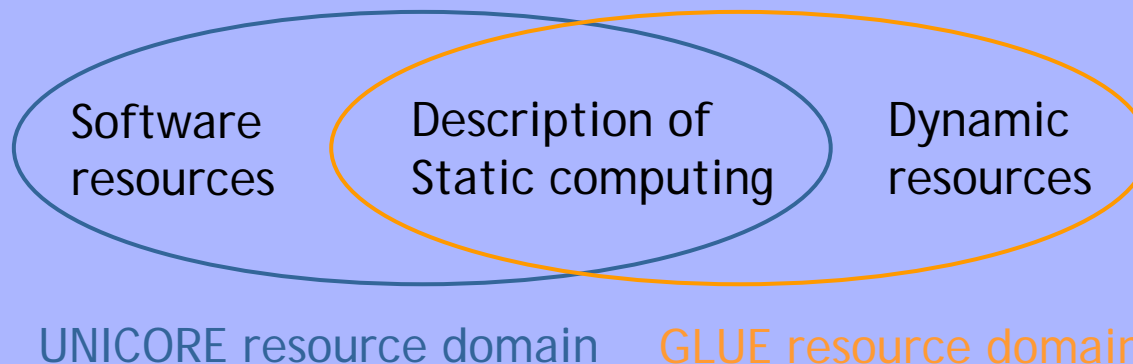


Interoperability architecture



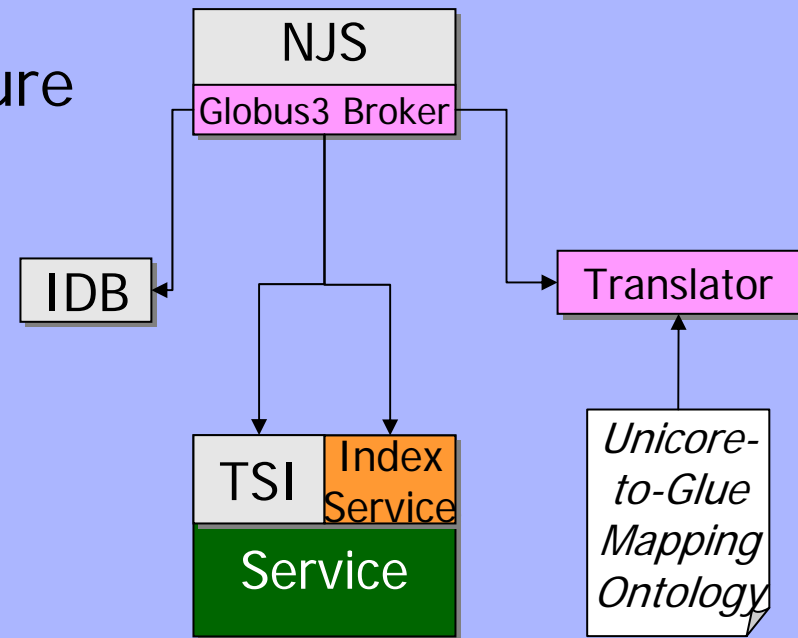
Resource models: Comparison

- 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

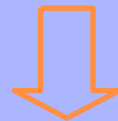


Lessons learned

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?

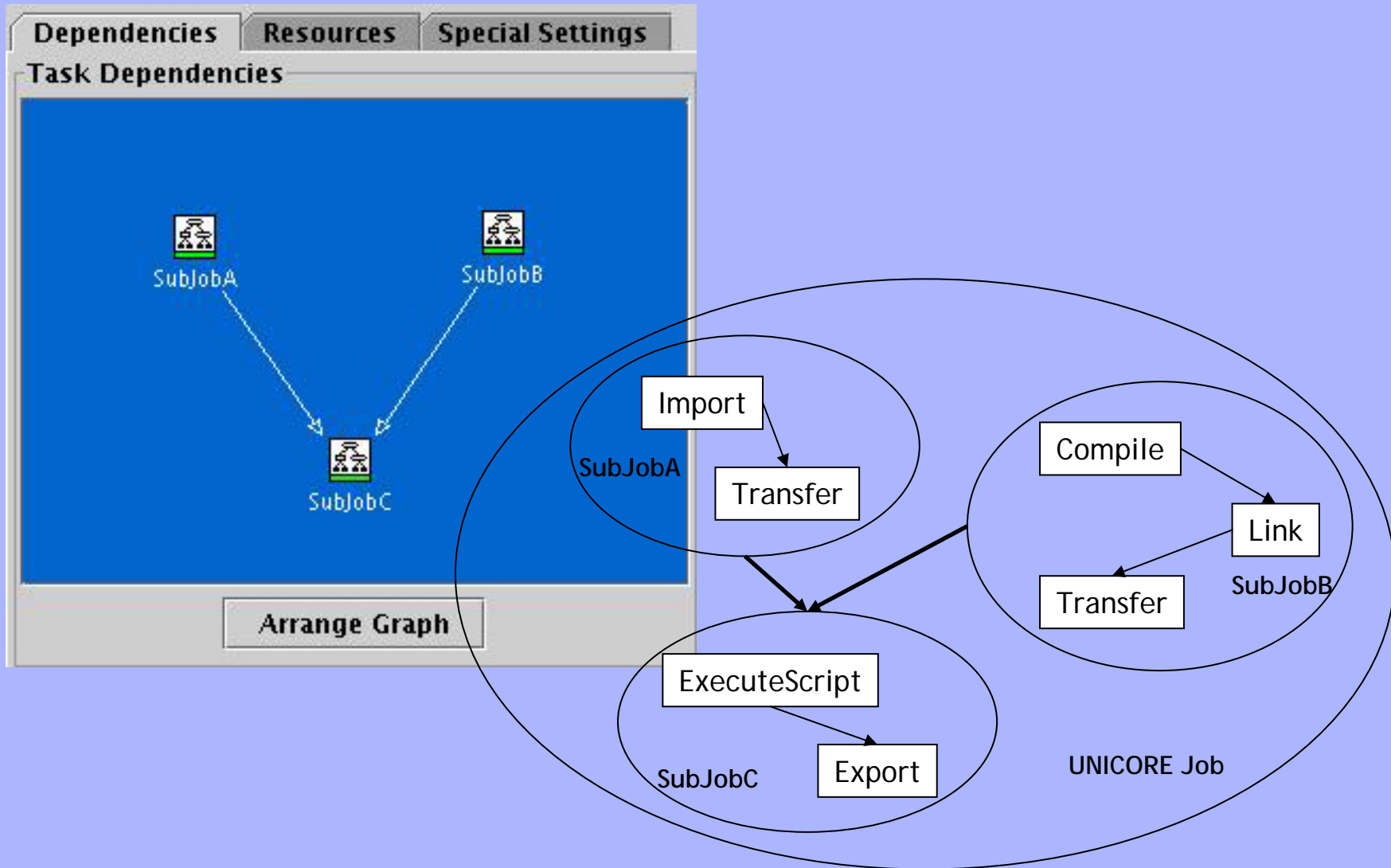
Requirements:

- **Automated** workflow support
 - **Automated** resource management
 - Application support
- ... based on core UNICORE services and functions

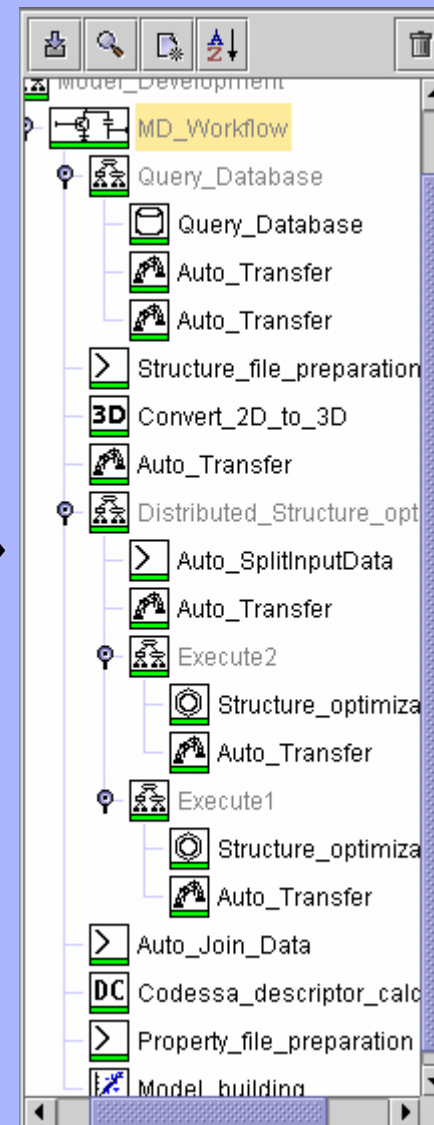
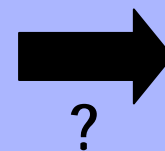
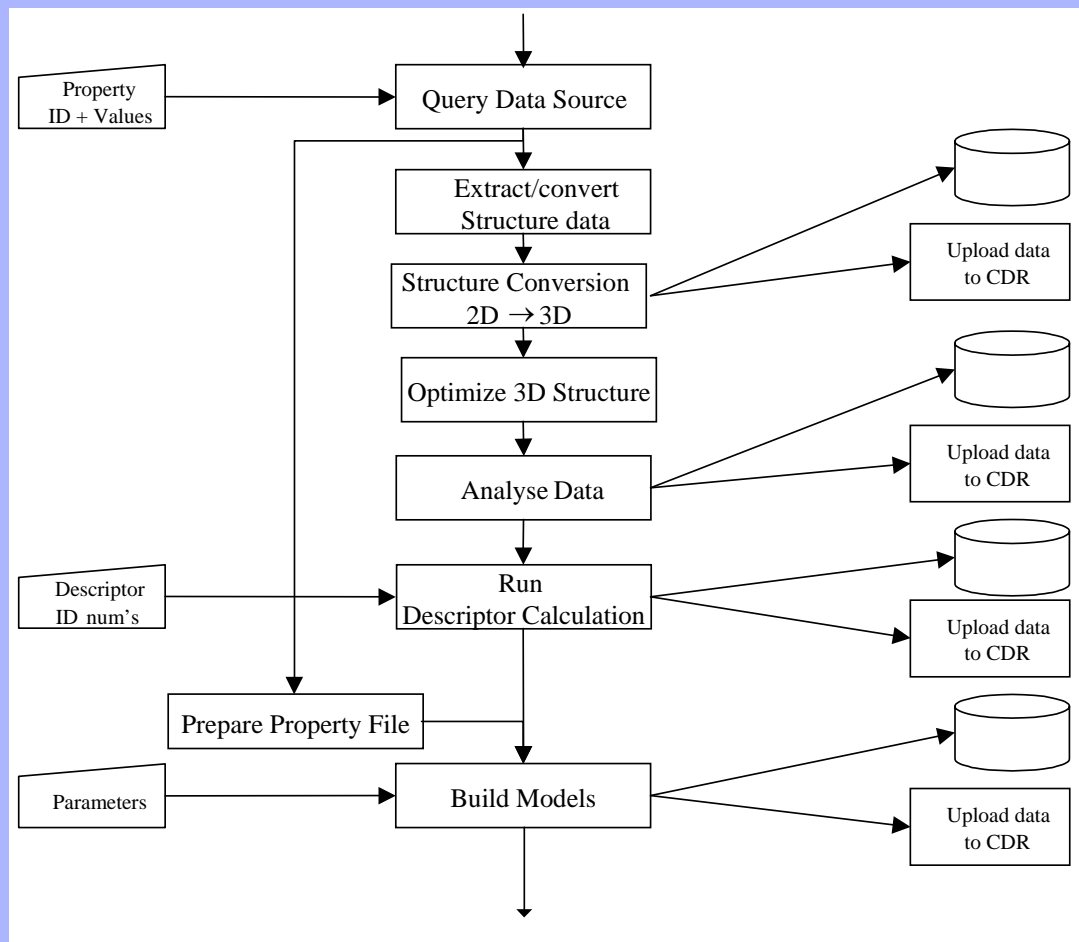
So, why has UNICORE to be extended?



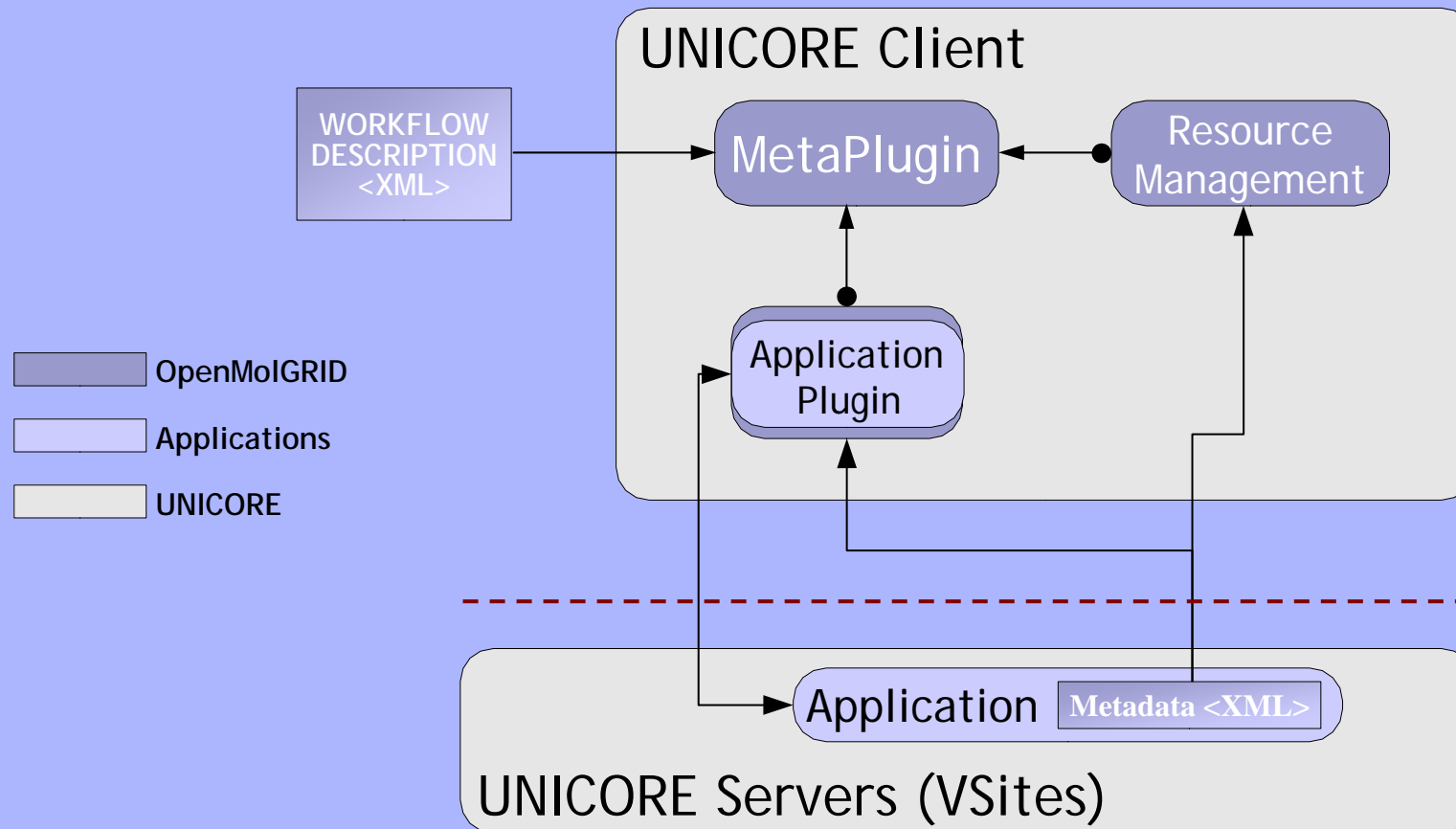
UNICORE workflow model



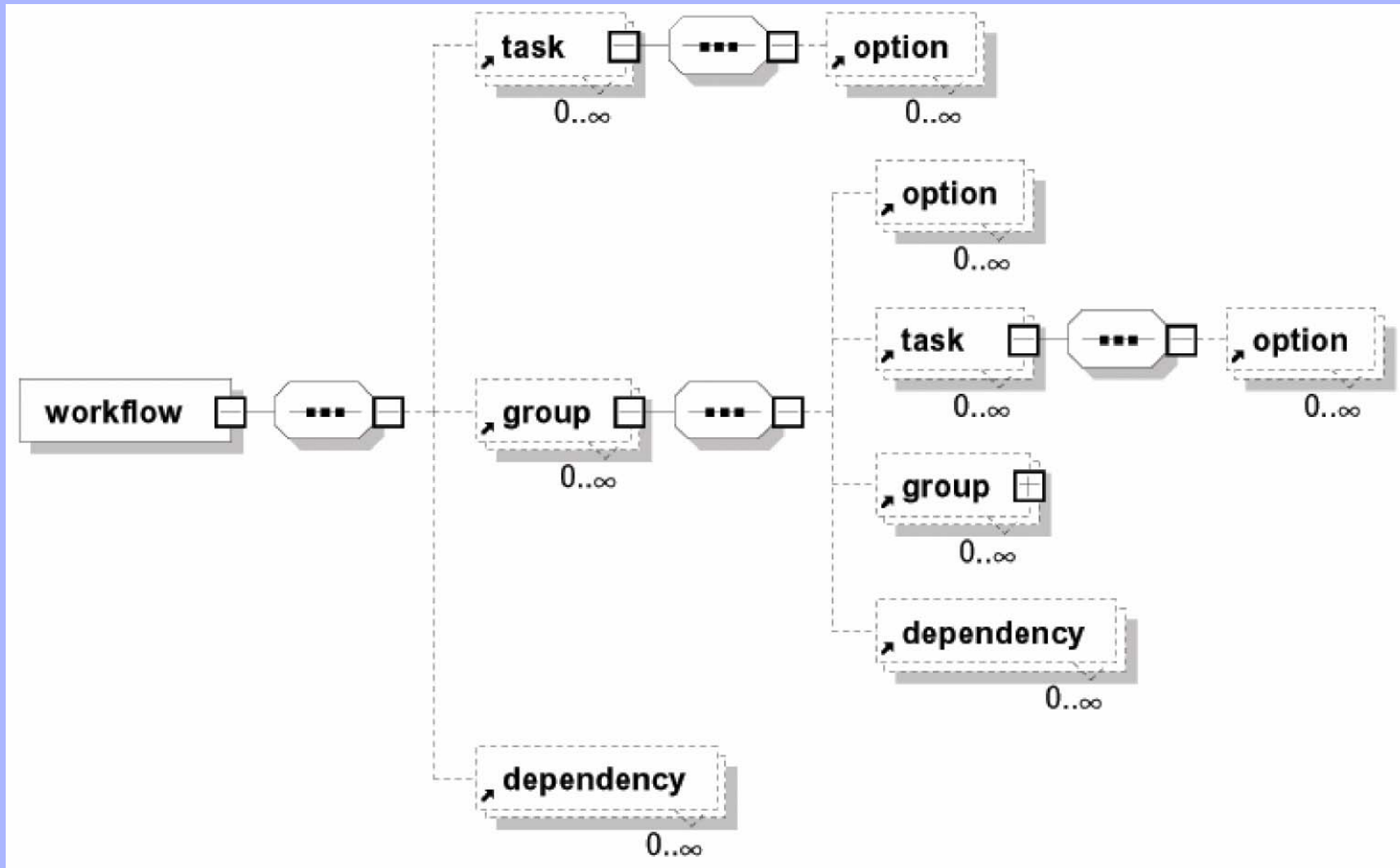
OpenMolGRID example workflow



Workflow support: Realisation



Workflow description

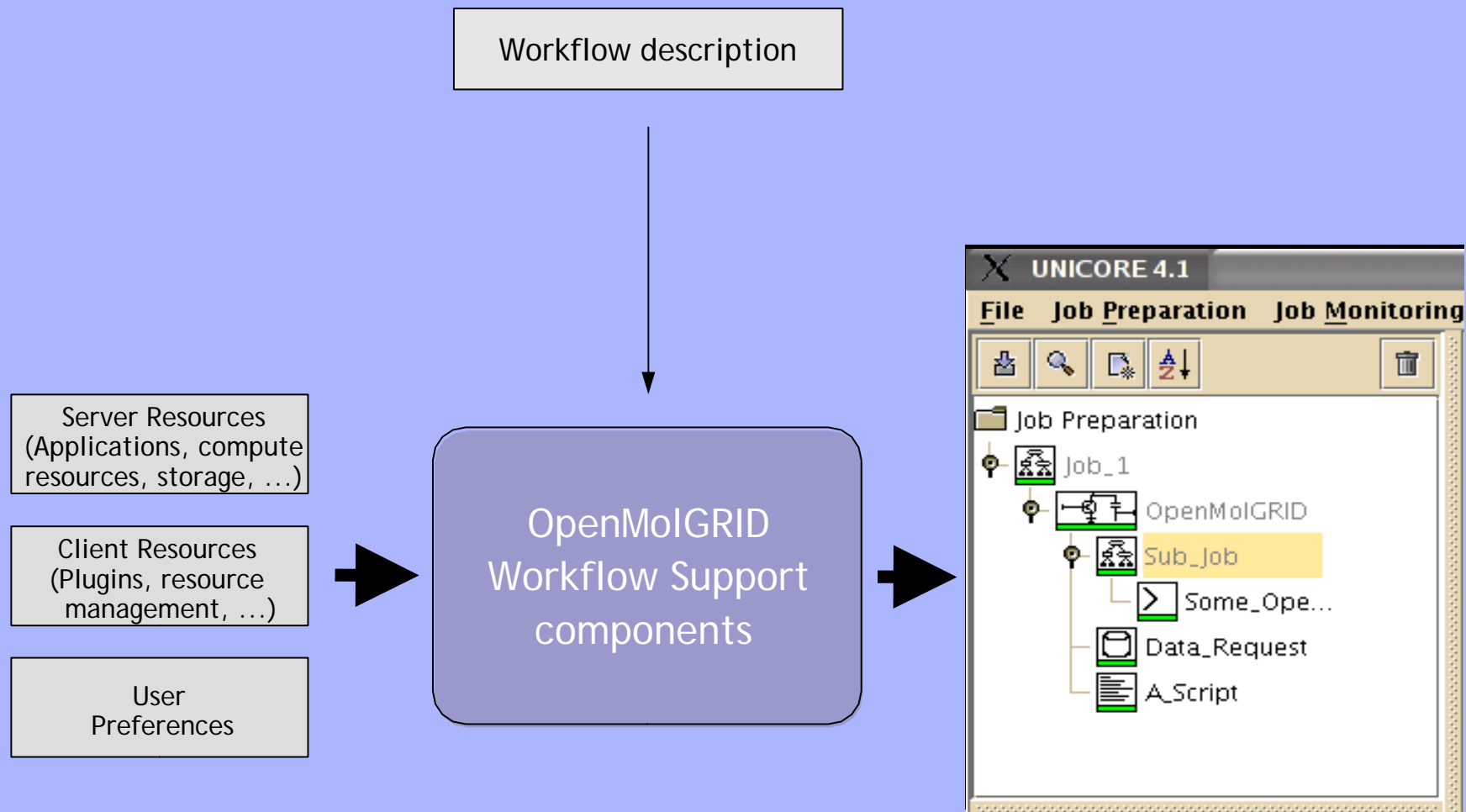


Workflow XML schema

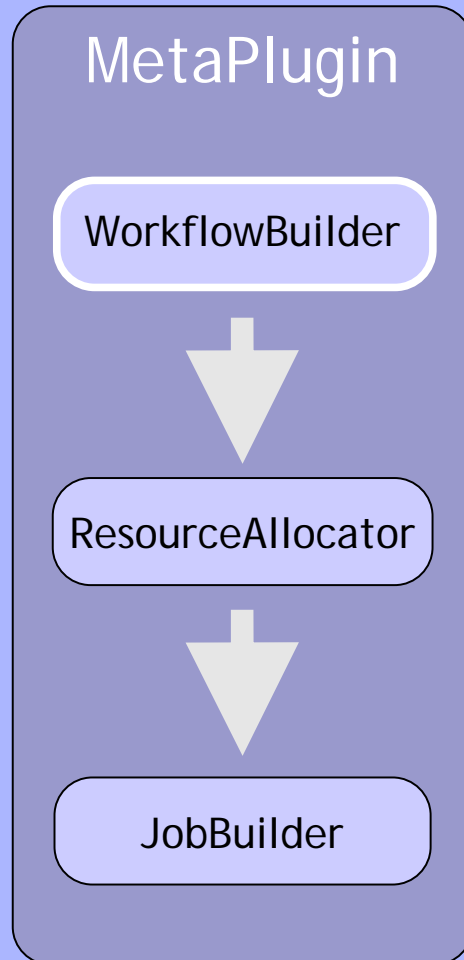
```
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
+ <xs:element name="workflow">
- <xs:element name="task">
- <xs:complexType>
+ <xs:sequence>
<xs:attribute name="name" use="required" />
<xs:attribute name="identifier" use="required" />
<xs:attribute name="id" use="required" />
- <xs:attribute 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>
```



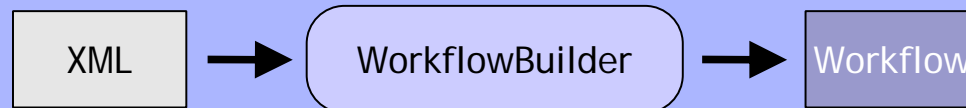
MetaPlugin: Overview



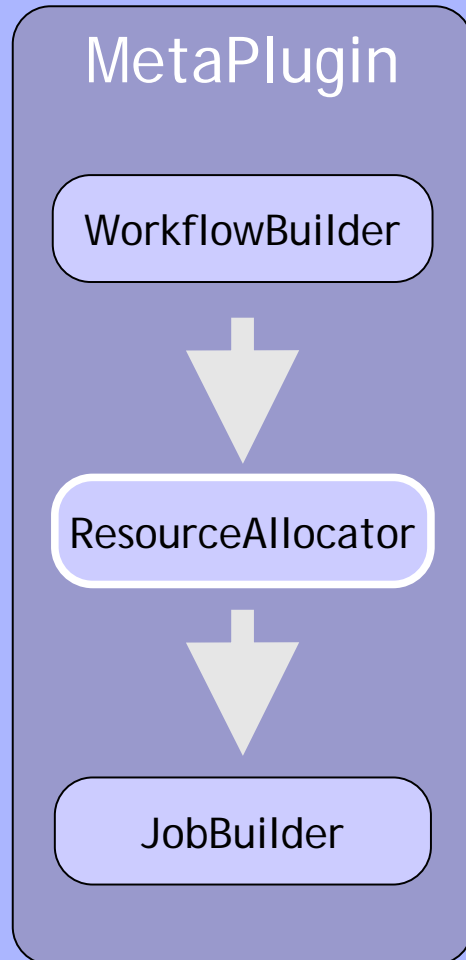
MetaPlugin: Workflow builder



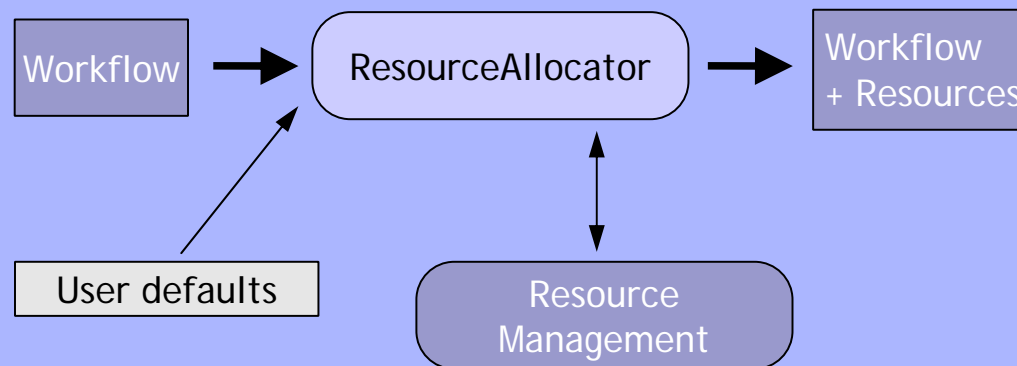
- WorkflowBuilder
 - Read workflow XML file
 - Build workflow object



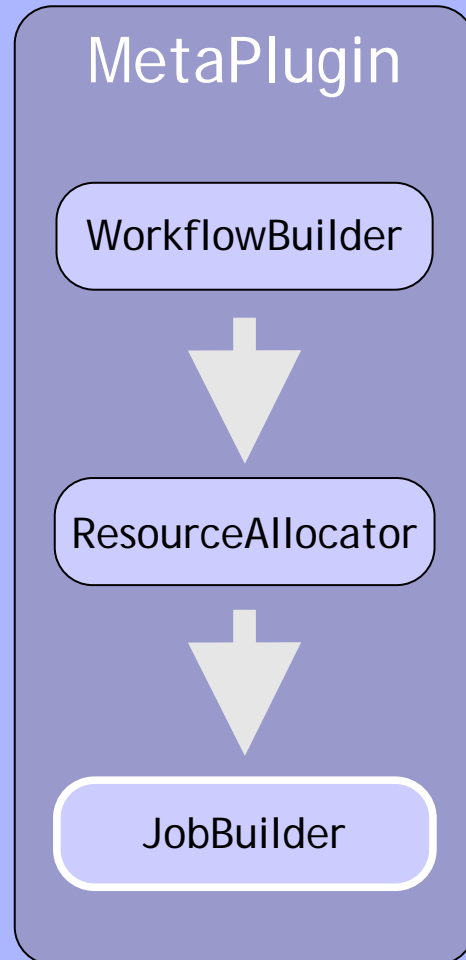
MetaPlugin: Resource Allocator



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

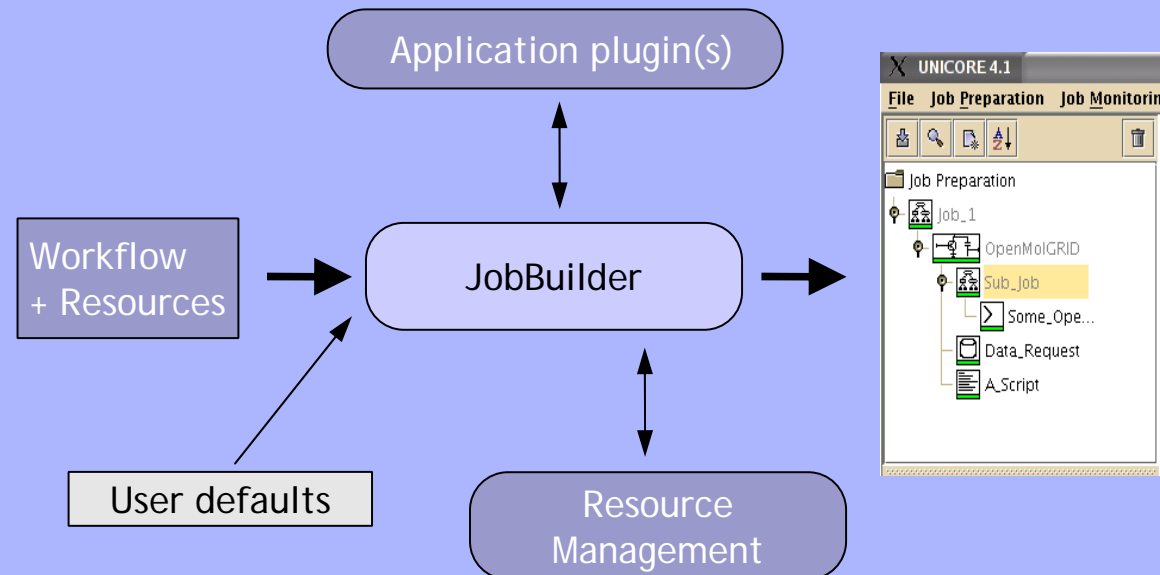


MetaPlugin: Job builder



➤ JobBuilder

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



MetaPlugin: workflow adjustment

The screenshot displays the UNICOREpro Client interface. The main window is titled "UNICOREpro Client" and has a menu bar with "File", "Job Preparation", "Job Monitoring", "Settings", "Extensions", and "Help".

Job Preparation Panel: A tree view on the left shows the workflow structure. The "Query_Database" task is selected and highlighted in yellow. Below it, "Auto_Transfer" tasks are visible.

Job Monitoring Panel: A tree view on the bottom left shows the status of various sites, including CGX OMG, Juelich OMG, Tartu OMG, and Ulster OMG.

Task Configuration Panel: The right side of the window is divided into several sections:

- Name:** A text box containing "Query_Database".
- Dependencies:** A tabbed interface with "Dependencies", "Resources", and "Special Settings". The "Dependencies" tab is active, showing a "Task Dependencies" graph. The graph illustrates a task named "Query_Database" with two arrows pointing to two "Auto_Transfer" tasks below it.
- UNICORE Site:** A list of sites: CGX OMG, Juelich OMG, Tartu OMG, and Ulster OMG.
- Virtual Site:** A list of virtual sites: OMG_CGX <NJS>.

Status Bar: At the bottom, it shows the user "mathilde romberg", the message "Model_Development not saved yet.", and a progress indicator "15.02Mb/17.67Mb".



MetaPlugin: workflow adjustment (cont)

The screenshot displays the UNICOREpro Client interface. The left sidebar shows a tree view of the workflow under 'Job Preparation' > 'Model_Development' > 'MD_Workflow'. The 'Query_Database' task is selected. The main window shows the configuration for this task. The 'Task Name' is 'Query_Database' and the 'Data source' is 'MOLDW'. The 'Expert mode' checkbox is checked. The 'Output options' section includes an 'Output file name' field with the value 'out_1084433180426_25', a checked 'Visualize the query results' checkbox, and an unchecked 'Additional format conversion and data export' checkbox. The 'Output data' section has buttons for 'Select all', 'Clear all', 'Store', 'default', 'Store as...', and 'Remove...'. Below these are tabs for 'chemical', 'descriptor', and 'property'. The 'chemical' tab is active, showing a list of output fields with checkboxes: 'chemical.moldw_id (The internal identifier associated with the chemical.)', 'chemical.casnumber (The Chemical Abstracts Service number associated with the chemical.)', 'chemical.chemicalname (The name associated with the chemical)', 'chemical.molecularformula (The molecular formula of the chemical)', and 'chemical.molecularweight (The molecular weight of the chemical)'. The status bar at the bottom shows the user 'mathilde romberg' and the message 'Model_Development not saved yet.'.



MetaPlugin: workflow control panel

The screenshot displays the UNICOREpro Client interface with the OpenMolGRID Workflow Control panel. The panel is titled "OpenMolGRID Workflow Control" and features a menu bar with "File", "Job Preparation", "Job Monitoring", "Settings", "Extensions", and "Help". The main area is divided into several sections:

- Workflow control:** Includes tabs for "UNICORE job control", "Workflow description:", and "To do:". A text field shows the "Name" as "MD_Workflow".
- Task Dependencies:** A graph showing the workflow structure. The tasks are: Query_Database (top), Structure_file_pre..., Property_file_pre..., Convert_2D_to_3D (with a "3D" icon), Auto_Transfer, Distributed_Struct..., Auto_Join_Data, and DC (bottom). Arrows indicate dependencies between these tasks.
- Workflow selection:** A section titled "Predefined Workflows" with a list containing "model_development" and "review_demo". It includes "OK" and "Cancel" buttons.
- Settings and Debug:** Buttons for "Settings...", "Print job (DEBUG)", and "Arrange Graph".

The status bar at the bottom shows the user "mathilde romberg", the file "Model_Development not saved yet.", and the memory usage "17.68Mb/23.91Mb".



MetaPlugin: Workflow description

The screenshot shows the UNICOREpro Client interface. On the left, a tree view displays a workflow named 'MD_Workflow' with various tasks and sub-tasks. The main window, titled 'OpenMolGRID Workflow Control', shows the 'Workflow description' tab with the following XML content:

```
<task name="DataBaseRequestToSLF" identifier="Structure file preparation" id="2"
  export="false" split="false">
</task>
<task name="DataBaseRequestToPLF" identifier="Property file preparation" id="3"
  export="false" split="false">
</task>
<task name="2Dto3Dconversion" identifier="Convert 2D to 3D" id="21"
  export="false" split="false">
</task>
<task name="SemiempiricalCalculation" identifier="Structure optimization" id="25"
  export="false"
  split="true" splitterTask="SplitStructureList"
  joinerTask="JoinStructureLists"
  <option name="keywords" value="AM1 PRECISE EF MMOK NOINTER"/>
</task>
<task name="DescriptorCalculation" identifier="Codessa descriptor calculation" id="29"
  export="false" split="false">
</task>
<task name="ModelBuilding" identifier="Model building" id="40"
  export="false" split="false">
</task>
<dependency pred="11" succ="2"/><!-- db request to structure extract-->
<dependency pred="11" succ="3"/><!-- db request to property extract-->
<dependency pred="3" succ="40"/> <!-- property extract to model building -->
<dependency pred="2" succ="21"/><!-- struct extract to 2d to 3d -->
<dependency pred="21" succ="25"/><!-- 2d to 3d to semiempirical-->
```



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

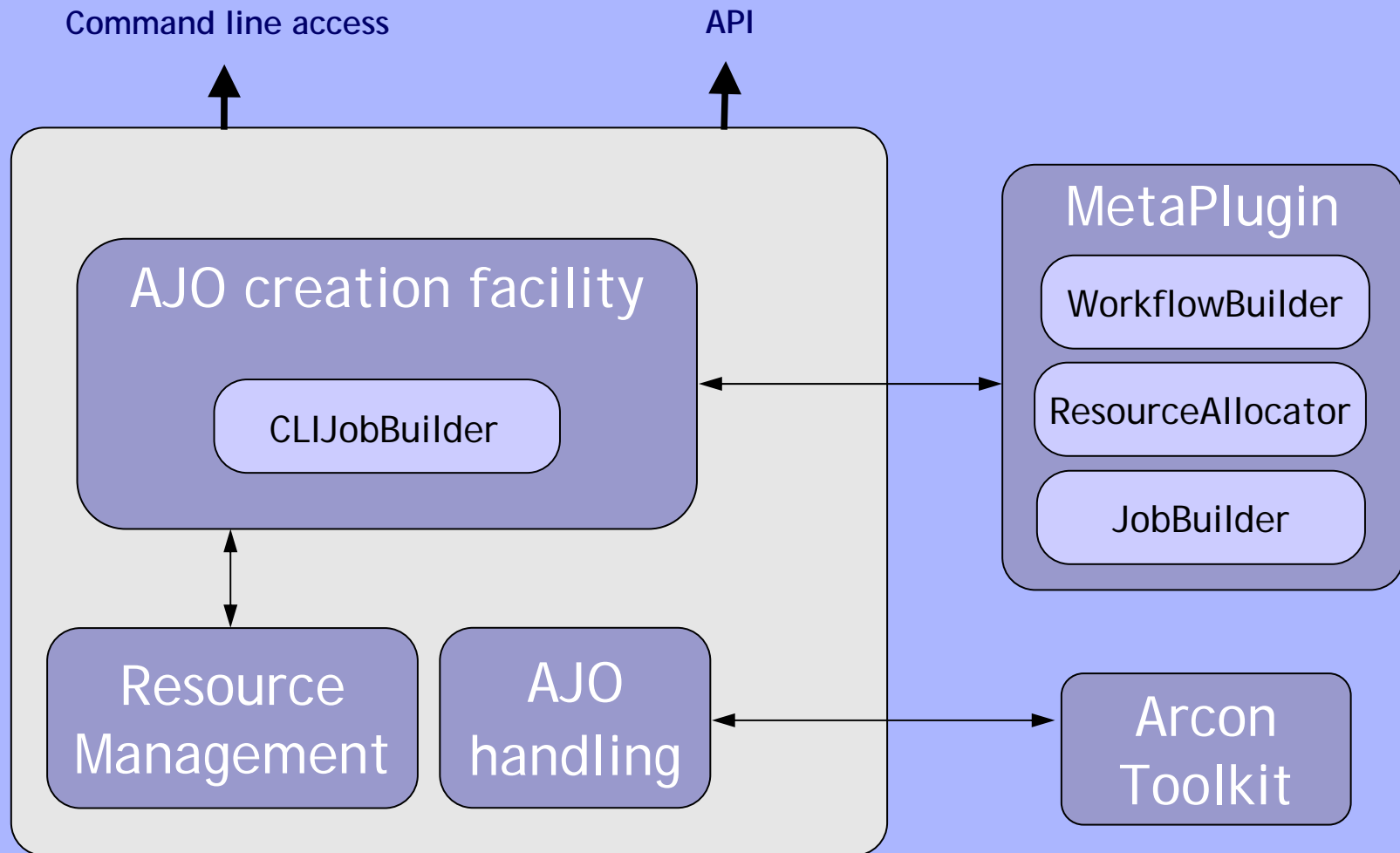


Command Line Interface (CLI)

- 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



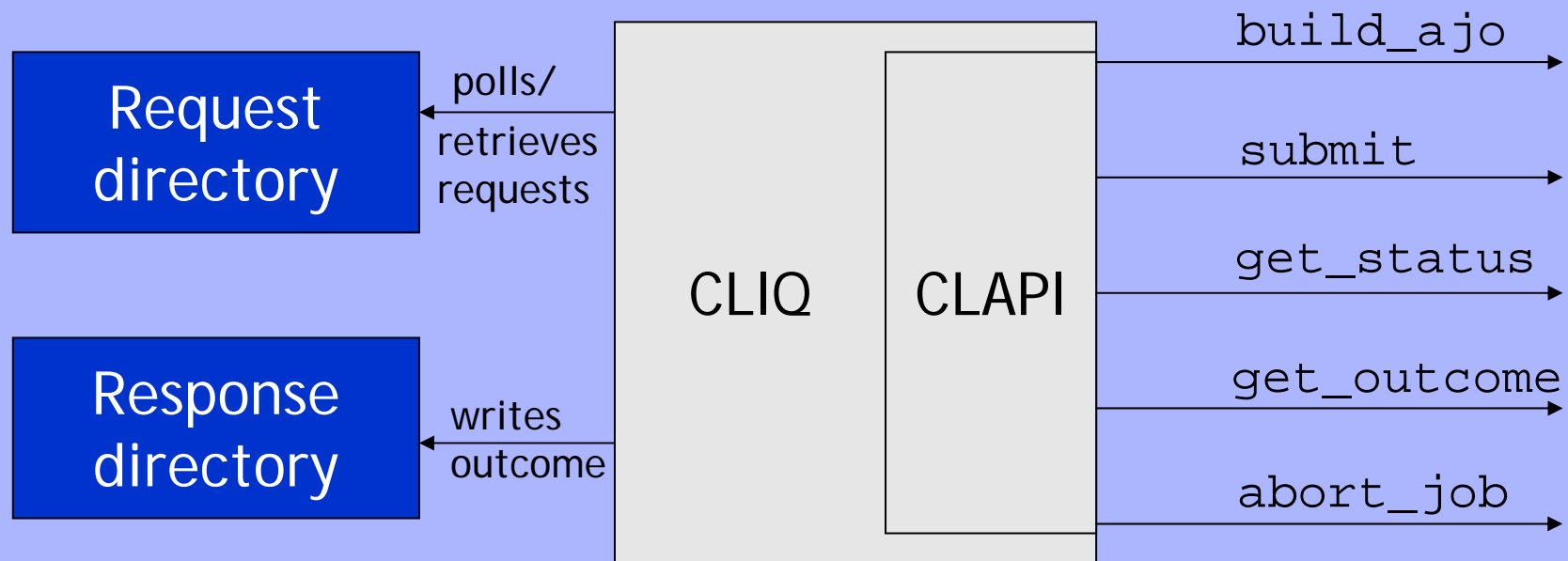
CLI: Example usage

CLI (from command line)	CLAPI (from Java application)
<pre>\$>cli build_ajo [options] \$>cli submit [options]</pre>	<pre>cli = new CLAPI(); cli.execute("build_ajo", [options]); cli.execute("submit", [options]);</pre>

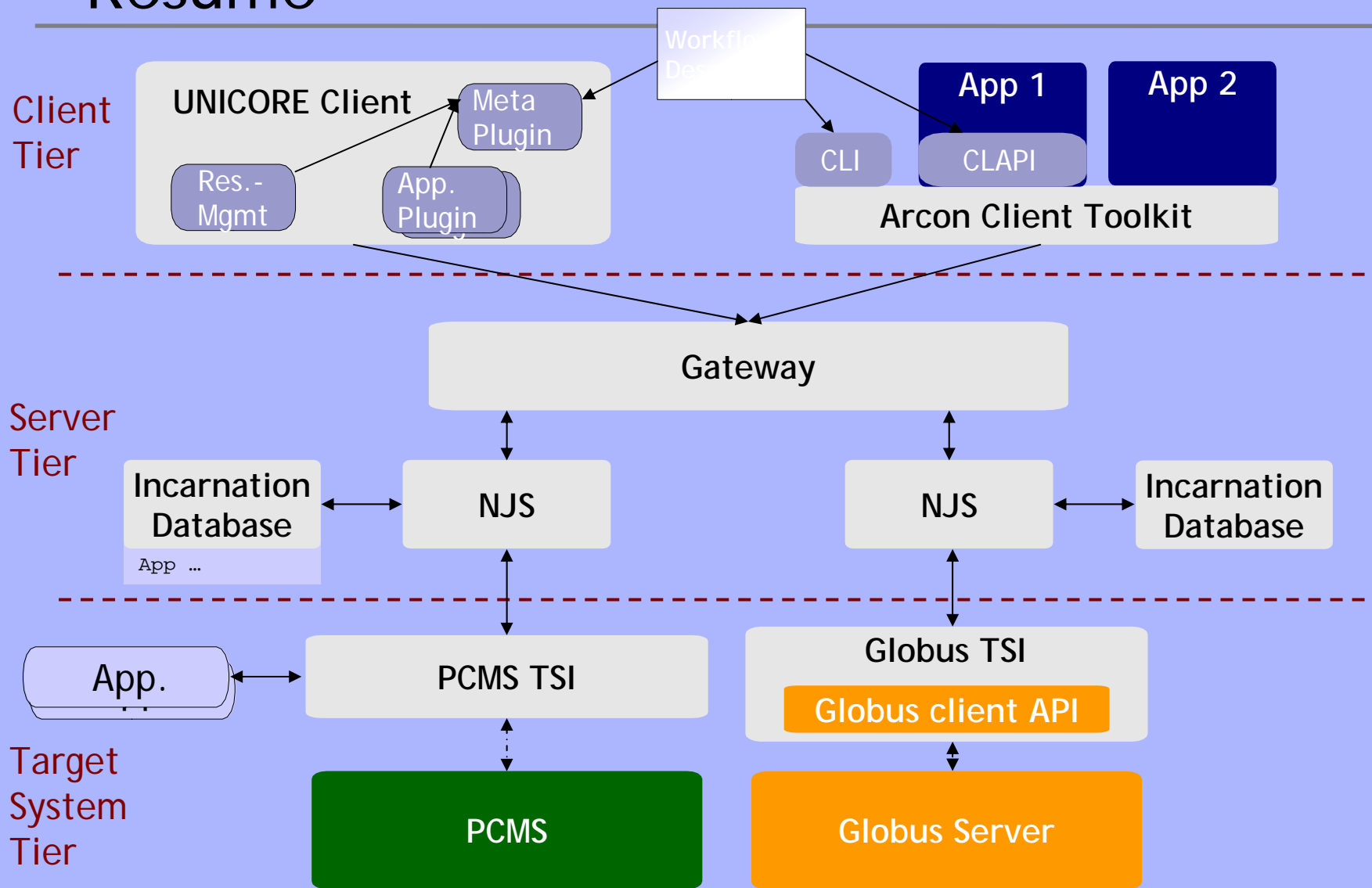


CLI Queue

- Automatic successive processing of multiple jobs



Resumé



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/>

