CoreGRID

Managed by

A Practical Approach for a Workflow Management System

Simone Pellegrini, Francesco Giacomini, Antonia Ghiselli

INFN Cnaf

Viale B. Pichat, 6/2 40127 Bologna

{simone.pellegrini | francesco.giacomini | antonia.ghiselli}@cnaf.infn.it

Outline

- Workflow Management Systems overview
- A *practical* approach for *real* workflows
- Implementation issues
- A case study: JDL to GWorkflowDL conversion

Outline

- Workflow Management Systems overview
- A *practical* approach for *real* workflows
- Implementation issues
- A case study: JDL to GWorkflowDL conversion

Workflow Management Systems Overview

- A lot of interest around WfMSs exist
  - Thanks to workflows, the processes' business logic can be easily expressed using graphs
    - Appealing for users with limited programming skills
- However, the lack of a recognized standard causes incompatibility among WfMSs:
  - *several* languages for workflow description exist:
    - Based on different *modeling formalisms* (DAGs, Petri-Nets, Pi-Calculus...);

Problems in Workflow Management

- The choice of a WfMS usually binds users to a specific workflow language
  - Change of the WfMS has a *high* cost:
    - *Legacy* workflows must be rewritten.
- *Interoperability* among WfMSs is still an open issue
- Difficulty to manage *large*, *complex* and *real life* scientific processes

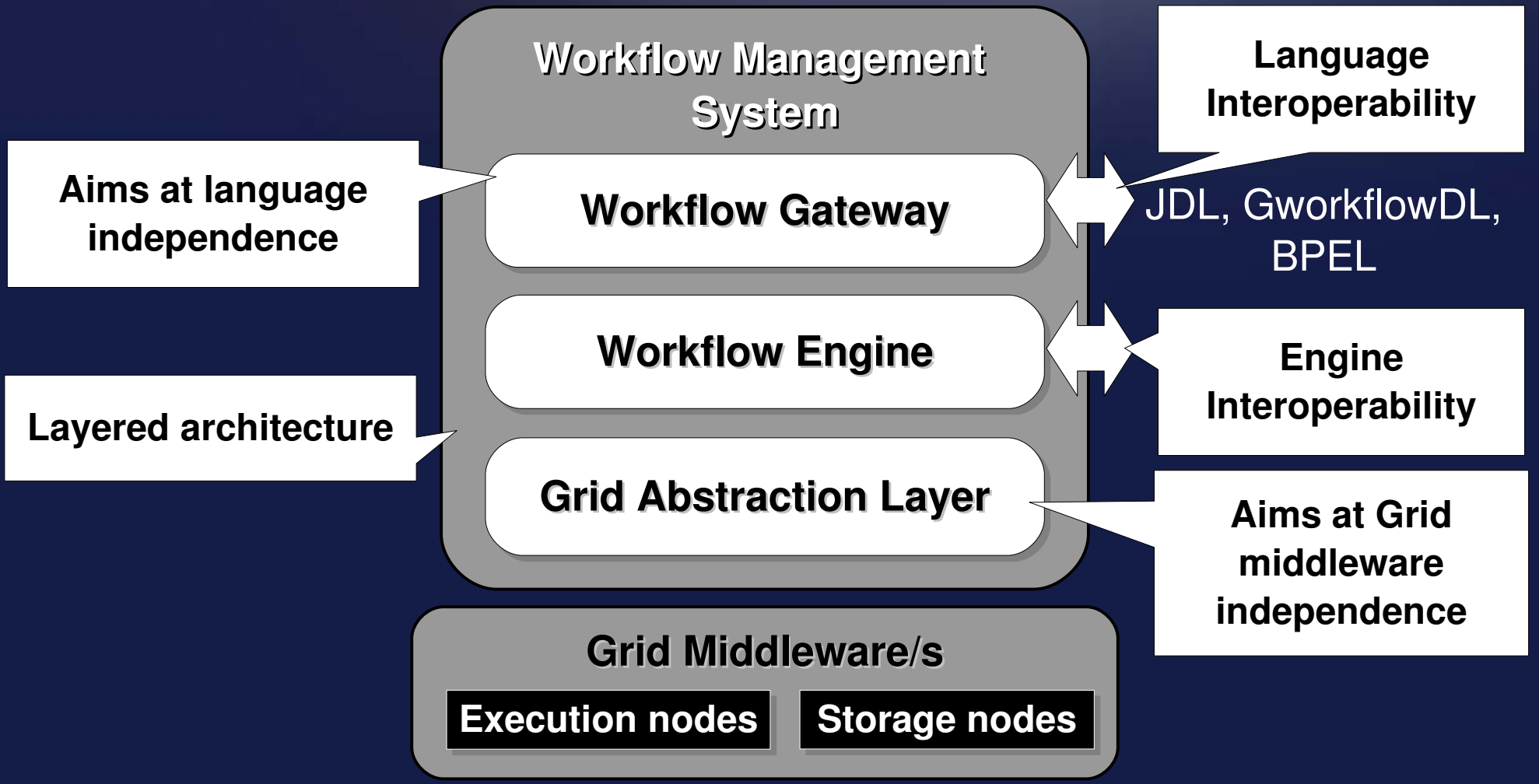
Outline

- Workflow Management Systems overview
- A *practical* approach for *real* workflows
- Implementation issues
- A case study: JDL to GWorkflowDL conversion

A practical approach for real workflows

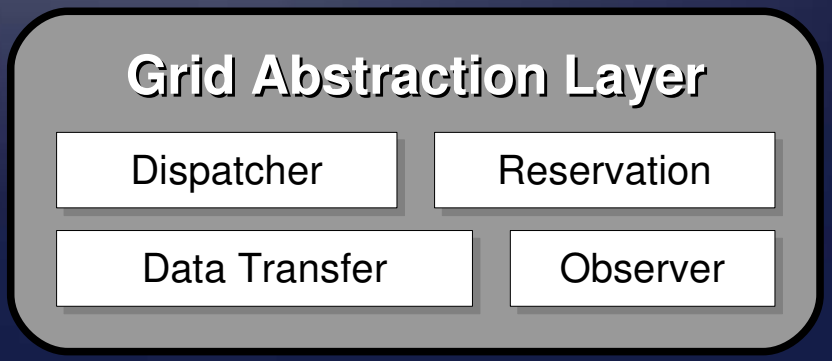
- Introducing a WfMS:
  - Petri-Nets based
    - Formal semantics
    - Turing-complete (deals with Workflow Patterns)
    - Build time analysis tools (*reachability*, *boundedness...*)
  - *Independent* from the underlying *Grid middleware*
  - Multi-language:
    - GWorkflowDL used as internal representation
  - Deals with *interoperability*

The WfMS Architecture



Grid Abstraction Layer

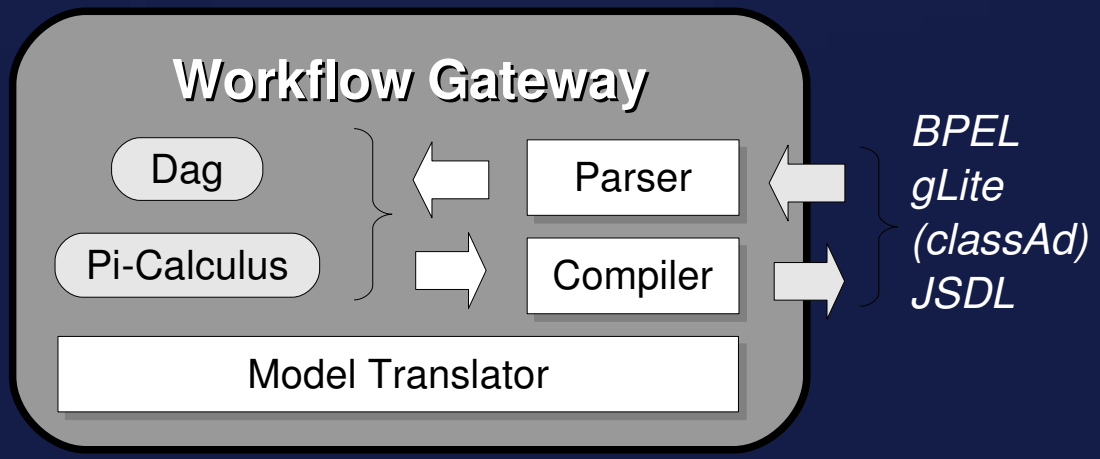
- *Abstracts* the basic functions of a Grid providing *portability* of the Workflow engine over different Grid middlewares



  - Dispatcher: Job submission/cancellation
  - Data Transfer: *Move data* between Grid nodes
  - Observer: *Monitor* submitted job status
  - Reservation: Resource reservation

Workflow Gateway

- Provides language and model converters in order to achieve compatibility with *legacy* workflows and *legacy* WfMSs
  - Parser: Extracts the *model* from a workflow description



  - Compiler: Produces a workflow representation using a target language (*JDL, GWorkflowDL, BPEL, …*)

Language Interoperability

- The *Gateway* solves some interoperability issues in workflow management:
  - Part of a workflow (or a sub-workflow) can be translated and successively delegated to a third-party WfMS
  - Legacy workflows can be executed on our WfMS without being rewritten
    - Every process can be expressed in terms of a Petri Net (Turing-complete)

The Workflow Engine

- Our goal is to keep the engine as simple as possible and concentrate on interoperability issues
- Main characteristics
  - Petri-Net base
  - Micro-Kernel architecture:
    - aims at modularity and extendibility
  - Distributed

Outline

- Workflow Management Systems overview
- A *practical* approach for *real* workflows
- Implementation issues
- A case study: JDL to GWorkflowDL conversion

EGEE/gLite as a Grid Middleware

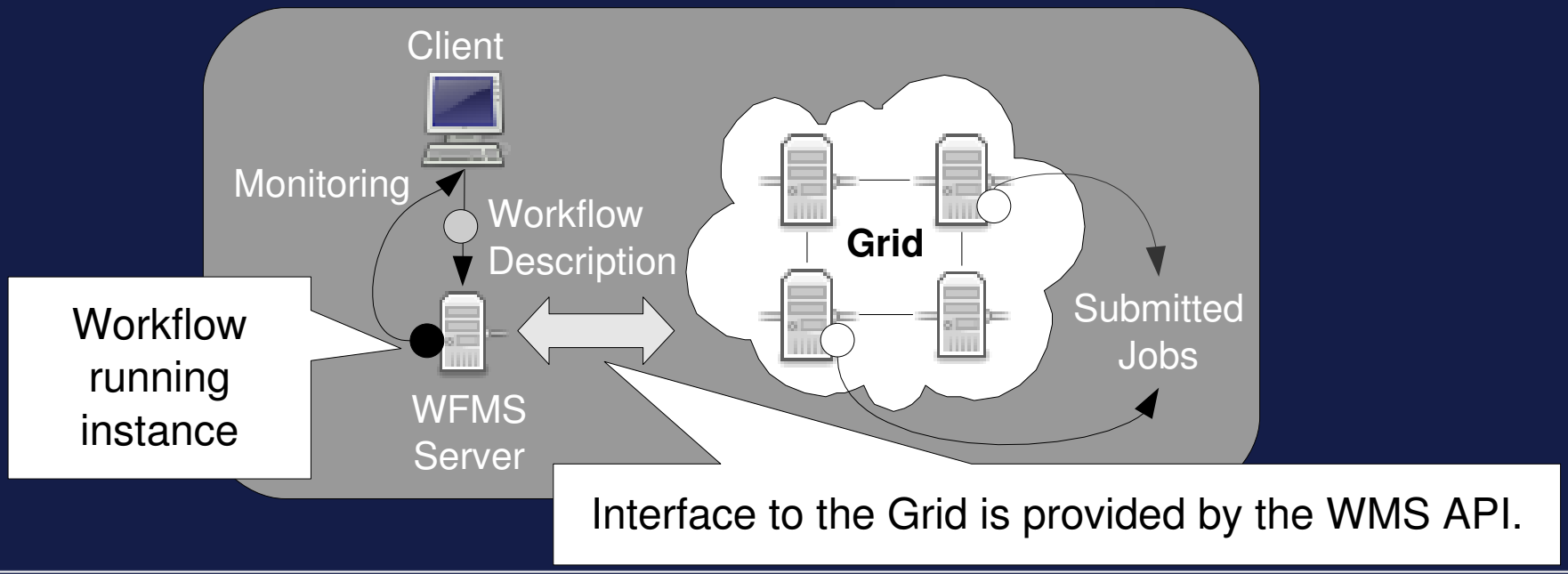
- Choice of EGEE/gLite middleware because of:
  - Services maturity
  - Reliability
  - Large adoption
- Job management is done by the *Workload Management System* (WMS)
- Job monitoring is done by the Logging and Bookkeeping Service (LB)

Project Goal: A WfMS over the WMS

- The practical outcome of our work is to build a WfMS relying on the WMS + LB
- Both WMS and LB provide a Web Service interface
  - simplify interaction with Grid services

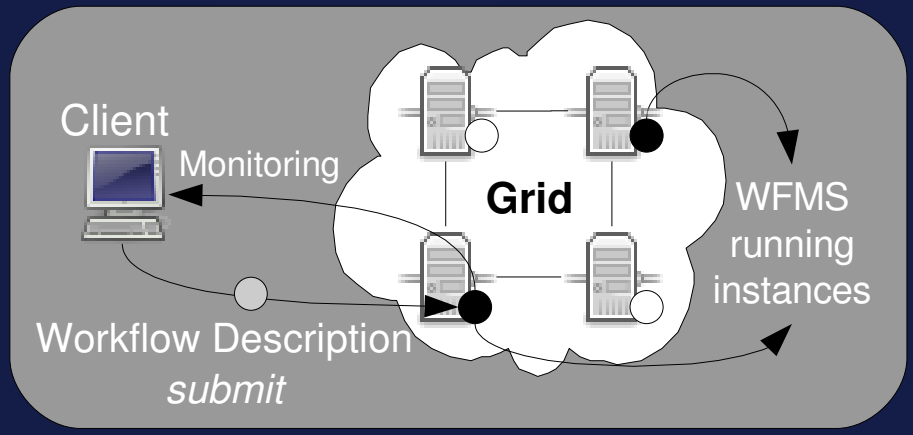
WfMS Deploy (1/2)

- WfMS running on a *dedicated server:*
  - Client sends the workflow description to the server;
  - WfMS server manages workflows execution;



WfMS Deploy (2/2)

- WfMS running as a *Grid job*:
  - Client submits a workflow to the Grid via the WMS and monitors it via the LB
  - The WfMS ends up running on a Grid node
  - The WfMS instance submits workflow tasks to the Grid using the WMS and monitors them via the LB



- Taking full advantage from the Grid resources and facilities (e.g. checkpointing)

Outline

- Workflow Management Systems overview
- A *practical* approach for *real* workflows
- Implementation issues
- A case study: JDL to GWorkflowDL conversion

The Job Description Language (JDL)

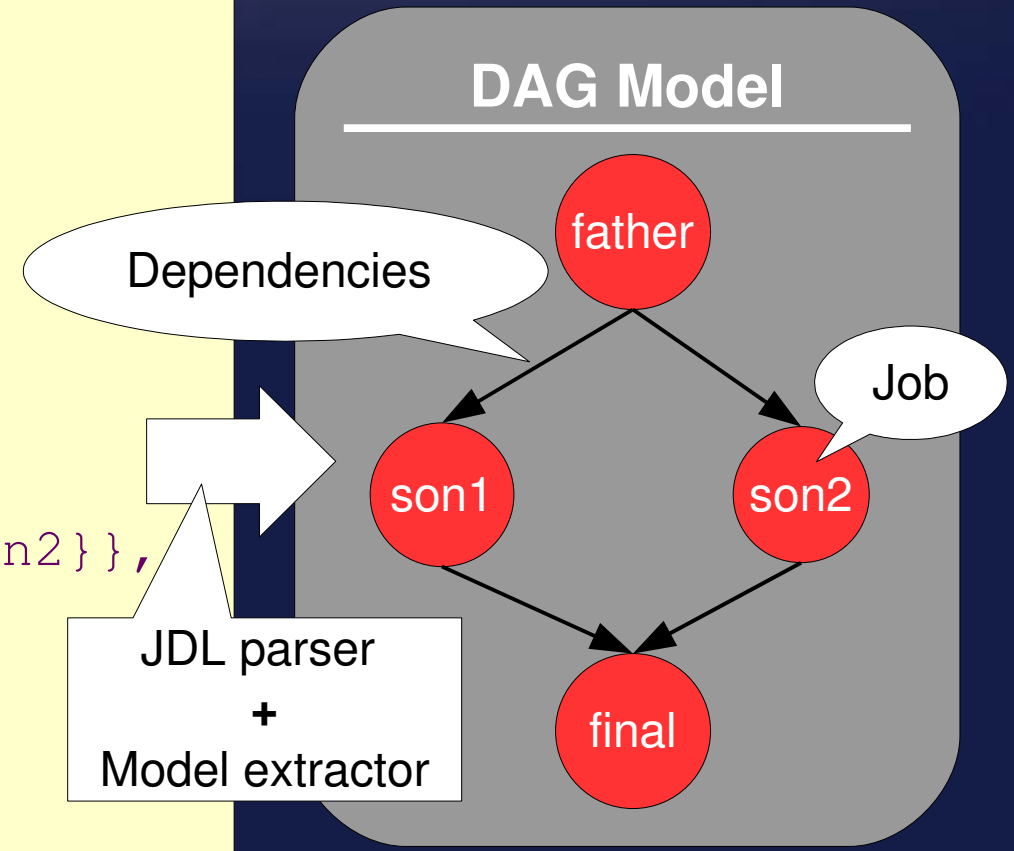
- The Job Description Language describes a job to be execution on the Grid
- The JDL adopted within the gLite middleware is based upon Condor's Classified Advertisements (ClassAds)
  - record-like structure composed of a finite number of attributes separated by semi-colon (;)

```
Attribute = Value;
```

A JDL DAG Workflow

- JDL allows workflow (DAG based) definition:

```
[
   Type = "dag";
   [ ... ]
   nodes = [
      father = [ ... ];
      son1   = [ ... ];
      son2   = [ ... ];
      final  = [ ... ];
      dependencies =  {
         {father, {son1, son2}},
         {son1, final},
         {son2, final}
      };
   ];
]
```



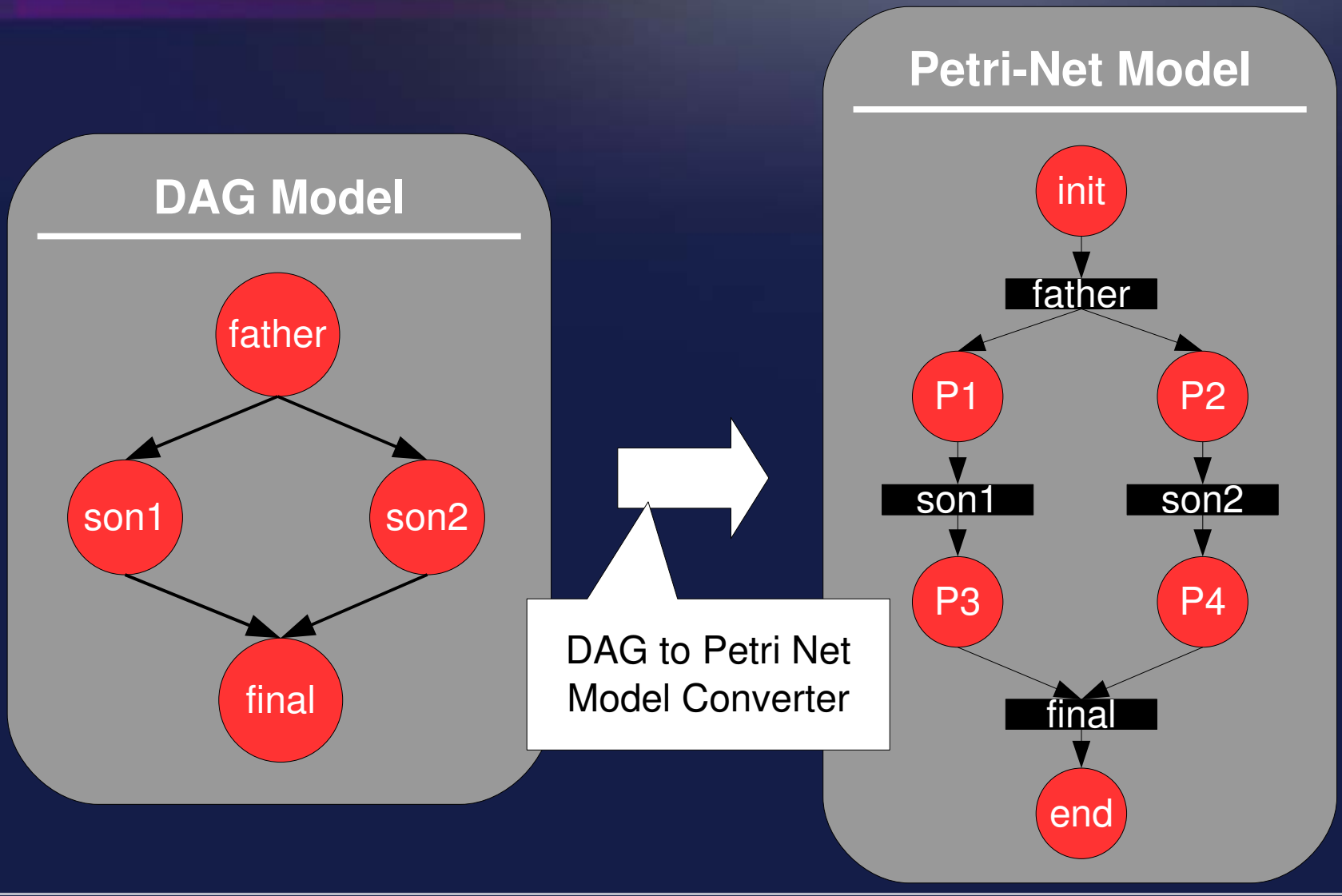
DAGs in gLite

- Many legacy workflows expressed in JDL exist;
- JDL workflows are managed by the Condor DAG Manager (DAGMan):
  - Acts as a *Meta-Scheduler* for Condor jobs
  - Submits job respecting their inter-dependencies
  - In case of job failure, DAGMan continues until it can no longer make progress.

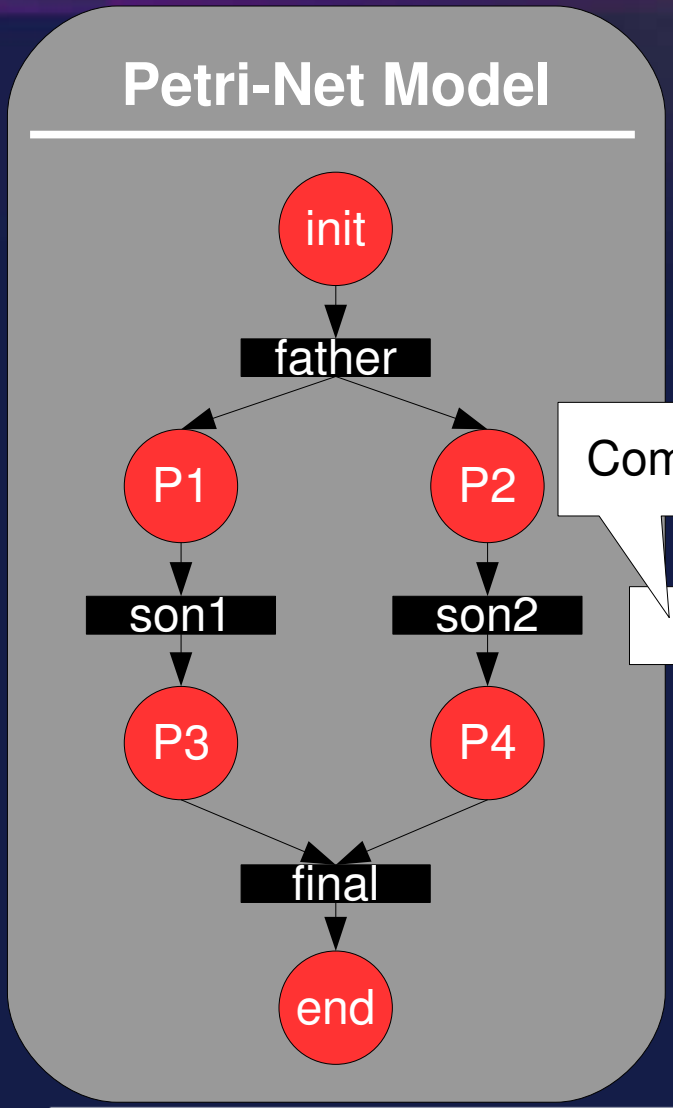
DAG -> Petri Net

- The current DAG model is very limited, e.g.
  - lack of error handling
  - lack of task types other than computation
- DAGs can be easily described using Petri Nets
  - a DAG *node* can be represented by a Petri Net *transition*
  - the flow of data among DAG nodes is modeled using data tokens

DAG -> Petri Net



Petri Net -> GWorkflowDL



Compiler

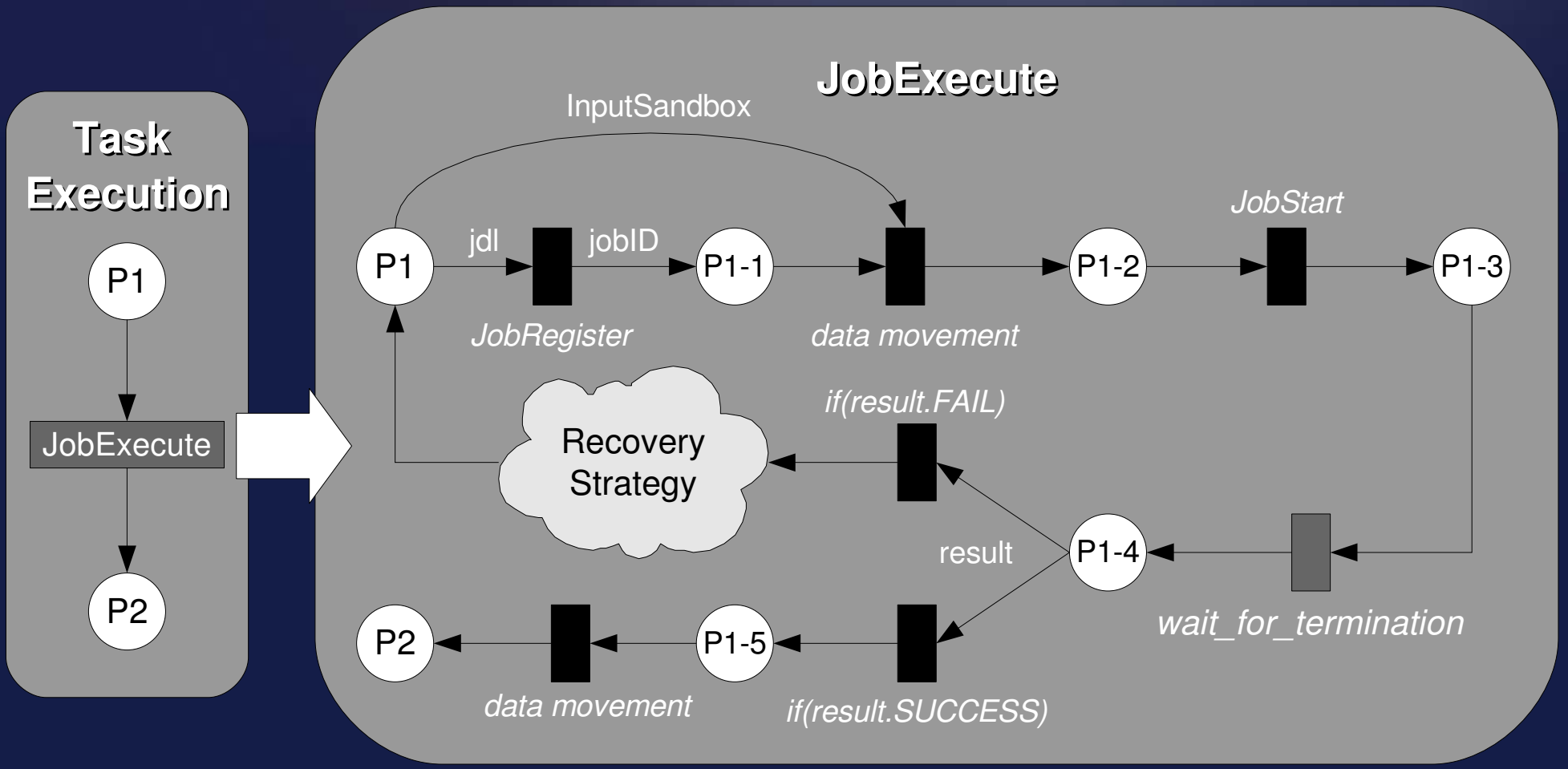
```
<workflow [...]>
   <place ID="init" />
   <place ID="P1" />
   <place ID="P2" />
   [...]
   <place ID="end" />
   <transition ID="father">
      <inputPlace placeID="init"/>
      <outputPlace placeID="P1"/>
      <outputPlace placeID="P2"/>
      <operation />
   </transition>
   [...]
</workflow>
```

DEMO

From abstract to concrete workflow (1/4)

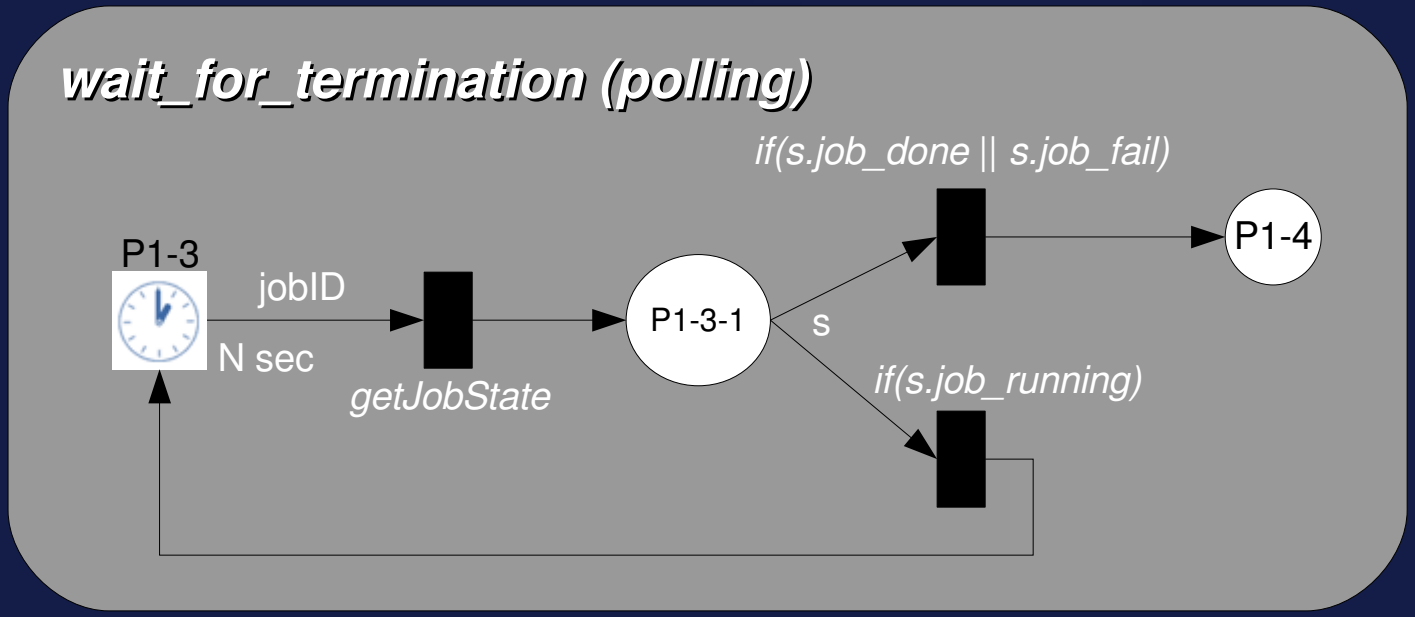
- The workflow description needs a *refinement process* in order to perform concrete task operations
- In the case of the gLite middleware task execution is asynchronous
  - The WMS serves job submission request returning an ID that identifies the job in its task queue
  - The ID is used to *query* (or register for notifications by) the LB service until task *termination* (or *failure*)

From abstract to concrete workflow (2/4)



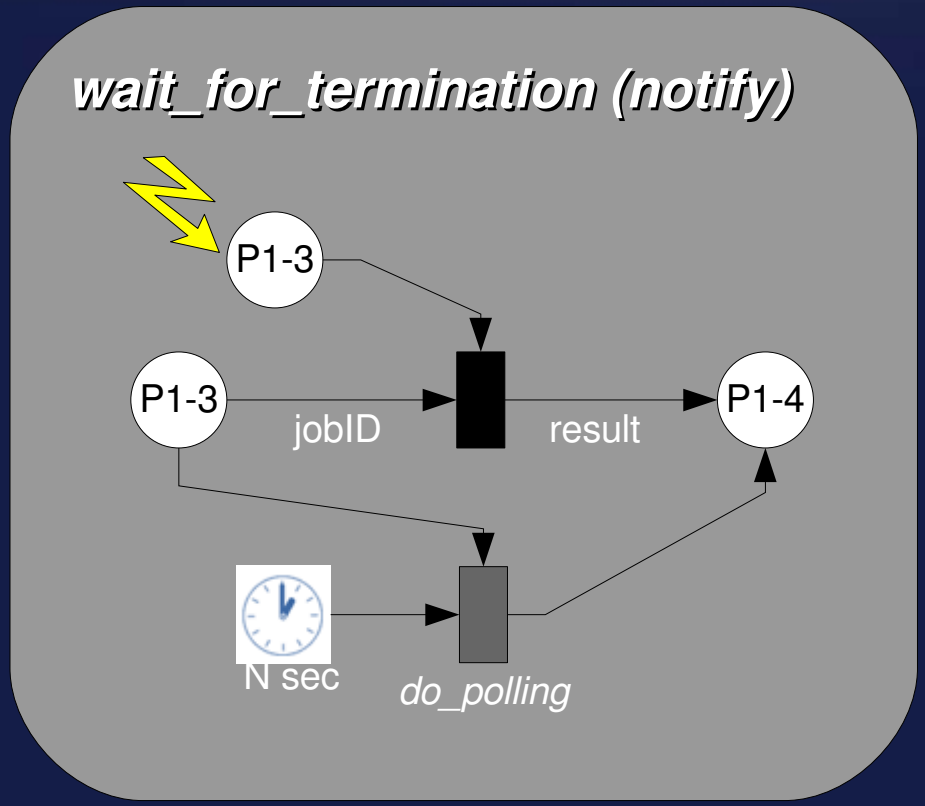
From abstract to concrete workflow (3/4)

- Waits until the task is done (or failed) using *polling:*
  - `getJobState()` operation of the LB service.



From abstract to concrete workflow (4/4)

- Same sub-workflow implementation using *notifications;*



Conclusions and Future Work

- *Workflow Gateway* as a standalone component
  - Other WfMS can *easily* take advantage from languages conversions
  - *customizable* target language depending on the underlying WfMS
- A *lightweight* WfMS with basic functionalities
  - solves low level aspects of workflow management
- Investigate WfMSs engine interoperability

Information Society

CoreGRID

Thank You!