Extending Grid-based Workflow Tools with Patterns and Operators

(1) Omer F. Rana (O.F.Rana@cs.cardiff.ac.uk)
(2) Cecilia Gomes (Cecilia.Gomes@di.fct.unl.pt)
(2) José C. Cunha (jcc@di.fct.unl.pt)

(5) School of Computer Science and Welsh eScience Centre, Cardiff University, United Kingdom
Workflow Lifecycle

- **Design**
  - Typical workflow is graph oriented
  - Language: how expressive is workflow
  - GUI: Visual Service Composition Environment

- **Deployment**
  - Workflow Description is sent to Workflow Engine
  - Possibly validated and compiled

- **Execution**
  - Workflow Engine enacts Workflow Description

- **Monitoring**
  - Events reflecting from workflow and services execution

- **Refinement**

From: Aleksander Slominski
Grid Workflow Taxonomy

- Grid Workflow System
  - Workflow design and specification
    - Component/Service Discovery
      - Structure
      - Composition
    - Model/spec
  - Scheduling and Enactment
  - Data Management
    - Operational Attributes

Rajkumar Buyya
Workflow Composition

Composition

User Directed
- Language-based
  - Markup
    - Scripting
  - Logic
    - Functional
- Process Calculi
- DAG
- UML
- Petri Net
- User defined (can include cycles)

Automated
- Planner
- Templates
- Design Patterns
- Sub-workflows
- Process Calculi
- Factory
Outline

1. Aspects of Workflow Optimization
   a. Composition time ("build time")
   b. Execution time
2. An Approach based on Patterns/Operators
3. An Example
4. Support for Patterns/Operators in Triana
5. Ongoing Work and Conclusions

Separation allows multiple "build time" mechanisms to be mapped to different run time mechanisms.
A Point of View...

- **Dynamic Optimization/Runtime-based**
  - Viewed as an improved scheduling approach
  - Consider aspects of computation and data
  - Workflow re-ordering, parallelism exploitation (task dependencies to support mapping)

- **Design time/Structure Optimization**
  - Reconfiguration of an existing workflow
  - Modifications to structure or execution order

Need some interaction between these two views
Motivation

- What we do:
  - identify and reuse common “idioms”
  - occur across different scientific domains

- An “idiom” captures common knowledge and experience and describe how a similar set of experiments are to be set-up and managed.
Approach
Approach

- Patterns are divided in **two categories** for flexibility:
  - Co-ordination (Behavioural) patterns
    - Capture interactions between software sub-systems
  - Structural patterns
    - Capture connectivity between particular types of Grid software/hardware components
Approach

1. Deploy Structural Patterns,
2. Refine them through Structural Operators,
3. Use Behavioural Patterns to define control/data flow/interactions,
4. Use Behavioural Operators to manage execution
Approach

Allow to build applications in a structured way:

-- select appropriate set of patterns, combine them according to operator semantics,

-- define new patterns and operators found useful and add them to the environment
Structural Pattern Templates

- **Encode component connectivity.** Ex: Pipeline, Ring, Star, Façade, Adapter, Proxy.

![Diagram of structural patterns]

- Pipeline PT
- Ring PT
- Star PT
- Proxy PT
- Facade PT
- Adapter PT
Structural Operators

- Manipulate structural patterns keeping their structural constraints.
- Examples:
  - Increase, Decrease,
  - Extend, Reduce,
  - Embed, Extract,
  - Group,
  - Rename/Reshape, ...
Increase Structural Operator

**Pattern**

- Real Subject
- Proxy

**Result Pattern**

- Increase(Pipeline, 2)
- Increase(Proxy, 2)

- Real Subject
  - Proxy
  - Proxy
  - Proxy
Extend Structural Operator

Pattern

Real Subject
Proxy
Facade

Result Pattern

Real Subject
Proxy
Proxy
Facade

Extend(Proxy, element)

Extend(Facade, element)
Embed Structural Operator

A star embedded in the first component place-holder of a pipeline template.
Behavioural Pattern Templates

- Capture temporal or (data/control) flow dependencies between components.
- Examples:
  - Client/Server,
  - Master/Slave,
  - Streaming,
  - Service Adapter,
  - Service Migration,
  - Broker Service
  - Service Aggregator/Decomposer, ...
Behavioural Operators

- Act over the temporal or flow dependencies for execution control and reconfiguration.

Examples:
- Start, Terminate,
- Log,
- Stop, Resume,
- Restart, Limit,
- Repeat, ...
### Supported Patterns/Operators

<table>
<thead>
<tr>
<th>Structural</th>
<th>Patterns</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pipeline, Star, Ring, Bus Adapter, Proxy, Facade</td>
<td>Rename, Replace, Increase, Decrease, Extend, Reduce, Replicate, Embed, Group/Aggregate</td>
</tr>
</tbody>
</table>
Pattern Operators - example

1: Initialize
2: Increase 1
3: Create Pipeline TransfVisSrv
4: RunStructuralScript TransfVisSrv
5: Instantiate DummyUnit
/toolboxes/SignalProc/Injection/Gaussian.xml
6: Instantiate DummyUnit1
/toolboxes/SignalProc/Algorithms/FFT.xml
7: EndStructuralScript
8: Embed TransfVisSrv DummyUnit1
9: Instantiate DummyUnit
/toolboxes/SignalProc/Input/Wave.xml
10: Instantiate DummyUnit2
/toolboxes/SignalProc/Output/Histogrammer.xml
11: Instantiate DummyUnit3
/toolboxes/SignalProc/Output/SGTGrapher.xml
12: Restart
20000
3- Implementation over Triana

- The Triana tool is a component-based front-end for Grid Environments (developed at Cardiff University).

- Extending Triana:
  - Structural patterns
  - Structural operators
• Applications are built by connecting services available in a toolbox
• The execution follows the dataflow model
The User’s View

- To configure and execute an application using the patterns library:
  2. Select a structural pattern (e.g., pipeline)
  3. Refine the chosen pattern with structural operators
  4. Select a behavioural pattern to specify interactions (e.g., dataflow)
4. All place-holders must be instantiated with units or group of units

5. Use a behavioural operator (eg start) for controlling the execution...
The System View

- Patterns are available at the GUI to be composed / manipulated through operators
- Behavioural patterns map to a Resource Management System that coordinates the execution
- A "pattern executor" enforces the selected behavioural pattern at each element
- Operator invocations are passed to the RMS interface API
Triana architecture
Patterns and Operators in Triana
The System View

- The GUI sends the user's requests for execution to the local TRIANA Service, as a taskgraph.
- Execution may be local or distributed.
- Each TRIANA service may delegate execution to a local RMS.
3- Implementation over Triana – Galaxy simulation example
Galaxy simulation example
Galaxy simulation example
Galaxy simulation example: a second configuration
Galaxy simulation example: a second configuration
Figure 15: Applying the Repeat Behaviour Pattern for launching the execution ten times
4- Mapping to the DRMAA API

Stop(Pattern)

Pattern Instance

\texttt{drmaa\_control(job\_id, DRMAA\_CONTROL\_SUSPEND, ...)}

job 1

Data and Control flow

job 2

Data and Control flow

job 3
Resume(Pattern)

drmaa_control(job_id, DRMAA_CONTROL_RESUME, ...)

job 1
Data and Control flow
job 2
Data and Control flow
job 3
for(int count=0; count<n; count++) {
    Start(Pipeline);
    drmaa_synchronize(job_identifiers, ...)
}

Repeat(n, Pattern)
Related Approaches

- **Skeletons** – “A skeleton is a programming template which is parameterised by the application programmer with problem-specific customising functions and commonly specified as a higher-order function [...] which can perform recursive computations”;
  - represent common behaviour in parallel programming and are used as components for building parallel programs. E.g.: *map, filter*. [Herrmann and Lengauer] in “Patterns and Skeletons for Parallel and Distributed Computing”

- **Service Design Patterns for Computational Grids** – identify how applications may be composed, shared, and managed over a Computational Grid. E.g.: *Broker Service Pattern, Service Adapter Pattern*. [Rana et al] in “Patterns and Skeletons for Parallel and Distributed Computing”
  - The *Enhance Project*, for example, aims to enhance the performance predictability of Grid applications with...
Related Approaches

- **Workflow Patterns**
  - common requirements and control flow schemas, e.g.
    - Basic Control Flow Patterns: Sequence, Parallel Split;

- **Patterns for Object-Oriented Middleware**
  - recurring structure and interaction schemas in middleware like CORBA, Web Services and Peer-to-Peer systems. E.g.:
    - Service Access and Configuration Patterns: Wrapper Façade, Component Configurator; Event Handling Patterns: Reactor [Schmidt et al] – Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects.
<table>
<thead>
<tr>
<th></th>
<th>Our approach</th>
<th>Skeletons</th>
<th>Service Design Patterns for Computation</th>
<th>Workflow Patterns</th>
<th>Patterns for OO Middleware</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application Abstractions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Design Abstractions</strong></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Software Engineering Abstractions</strong></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Programming Abstractions</strong></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
To Conclude

- Understand common patterns in Grid environments and applications
  - Grid app. == composition of services
  - Triana == tool to support workflow for distributed components

- Software Engineering support:
  - Structural and Behavioural Patterns
  - Structural and Behavioural Operators

- Ongoing work: implementation of behavioural patterns and operators.
Further Reading

- Pattern/Operator based Problem Solving Environments, Cecilia Gomes, Omer F. Rana, and Jose Cunha, EuroPar 2004, Springer Verlag, Pisa, Italy, August 2004.
- Patterns and Operators for Grid Software Development, Omer F. Rana, Maria Cecilia Gomes and Jose C. Cunha, IADIS International Conference WWW/INTERNET 2003, Algarve, Portugal, November 5-8 2003.