Patterns and Operators for Grid Software Development

(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
Outline

- Motivation
- Approach
- An implementation over Triana
- Mapping to the DRMAA API
- Conclusions
1- Motivation

- Grid Environments: complex; heterogeneous; dynamic.
- The Computational Grid hardware and software infrastructures are becoming stable.
- Next challenge: to build scientific applications that take advantage of the disparate collection of resources and services provided by Computational Grids.
1- Motivation

- Possible solution: to identify and reuse common “idioms” in some scientific domain and 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.
1- Motivation

- Our goal:
  1. To allow computational scientists and developers to capture design patterns that express common usage of software infrastructure within scientific domains
  2. To provide a software engineering tool that supports:
     - application configuration,
     - execution control, and
     - reconfiguration of software services
2- 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
2- Approach

- **Patterns as first class entities** both at design, execution, and reconfiguration times.

- **Pattern templates** are manipulated through **Pattern Operators**:
  - Structural operators
  - Behavioural operators
Structural Pattern Templates

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

![Diagram showing connectivity templates]

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

- Real Subject
- Proxy
- Facade
- Nucleus
- Satellite
- Adapter
- Adaptee
Structural Operators

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

Pattern

Increase(Pipeline, 2)

Result Pattern

Increase(Proxy, 2)

Real Subject

Proxy

Real Subject

Proxy

Proxy
Extend Structural Operator

Pattern

Real Subject → Proxy

Facade

Extend(Proxy, element)

Result Pattern

Real Subject → Proxy → Proxy

Facade

Facade

Extend(Facade, element)
Embed Structural Operator

A star embedded in the first component place-holder of a pipeline template.
Behavourial 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, ...
Pattern Operators - example

```c
main{
    P1;P2;
    section1{
        Rename(P1,P2);
        Replicate(P2,3);
        Owner(P2, "scmofr");
    }
    section2{
        Start(P2);
        Log(P2);
        Limit(30,P2);
    }
    section1; section2;
}
```
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
A Grid snapshot

Triana Interface: node in Cardiff

node in Lisbon

node in Barcelona

The Grid
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
4- Mapping to the DRMAA API

```
Pattern Instance

drmaa_control(job_id, DRMAA_CONTROL_SUSPEND, ...

job 1  Data and Control flow  job 2  Data and Control flow  job 3
```

Stop(Pattern)
Resume(Pattern)

Pattern Instance

drmaa_control(job_id, DRMAA_CONTROL_RESUME, ...)

job 1

Data and Control flow

job 2

Data and Control flow

job 3
Repeat(n, Pattern)

```java
for(int count=0; count<n; count++) {
    Start(Pipeline);
    drmaa_synchronize(job_identifiers, ...)
}
```

Pattern Instance

Job 1 Data and Control flow Job 2 Data and Control flow Job 3
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.