Autonomic managers in GCM

M. Aldinucci, M. Danelutto
Dept. Computer Science - Univ. Pisa & CoreGRID Programming model Institute
P. Dazzi, N. Tonellotto
ISTI/CNR Pisa & CoreGRID Programming model Institute

CoreGRID Industrial Conference - Sophia Antipolis - Dec. 1st, 2006
CoreGRID GCM Non Functional Features

• Components made out of two distinct, interacting parts

  • functional part: implements the functions/services actually computed by the component

  • non functional part: takes care of all those aspects that contribute making the component successfully and efficiently implementing its functional part

    • taking care of component performance, security, adaptivity

    • managing grid middleware/platform interactions

    • in general, implementing some form of autonomic control

European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
Insulated AC element cycle

- **Monitor**: collect execution stats: machine load, service time, input/output queues lengths, ...
- **Analyze**: instantiate performance models with monitored data, detect broken contract, in and in the case try to individuate the problem
- **Plan**: select a (predefined or user defined) strategy to re-convey the contract to valid status. The strategy is actually a list of mechanism to apply.
- **Execute**: leverage on mechanisms to apply the plan
Autonomic components in GCM

functional interface
hierarchical composition
non functional interface
managers
Autonomic component management in GCM

• Several levels of compliance
  • GCM components should sport autonomic features
  • but we allow to wrap non GCM, legacy components as well ...

• Level 0 (legacy components)
  component introspection only reveals no autonomic management

• Level 1 (passive autonomic)
  components support queries through non functional interfaces (server only NF interfaces)

• Level 2 (active autonomic)
  full component autonomic management (server & client NF interfaces)
Sample usage: passive autonomic components

- Pipeline component
  - component may provide bandwidth (BW): manager queries internal component NF interfaces
Sample usage: active autonomic components

- Pipeline component
  - accepting performance contracts
  - signaling contract violations

[Diagram showing pipeline components with stages 1, 2, and 3, with signals for stage BW violation and set BW.]
“Long term” expectation

• Compose your grid application

  • picking up components from some repository

  • assembling application using some GCM “structural” component

• expect non functional aspects (performance, in particular) and GRID interaction are all dealt with by the autonomic manager, without any kind of “programmer” intervention (but supplying proper contracts)
Preliminary results (by ASSIST / GRID.it)
Any questions?

marcod@di.unipi.it
Component-Based Grid Programming: 

the **GCM** way

F. Baude, D. Caromel, F. Viale, A. Di Costanzo, L. Henrio, M. Morel, 
INRIA Sophia-Univ. Nice-CNRS 

M. Danelutto 
Univ. Pisa 

V. Getov, N. Parlavantzas 
Univ. Westminster 

CoreGRID Industrial Conference 
Sophia Antipolis, December 1st 2006
• **GCM: Grid Component Model**
  - GCM Being defined in the NoE CoreGRID (42 institutions)
  - Open Source ObjectWeb ProActive implements a preliminary version of GCM
  - Autonomic, Adaptive
  - Service Oriented: NESSI relation exp.

• **GridCOMP takes:**
  - GCM as a first specification,
  - ProActive as a starting point, and
  Open Source reference implementation.
Scopes and Objectives:

Grid Codes that Compose and Deploy

No programming, No Scripting, … No Pain

Innovation:

Composite Components
Multicast
GatherCast
Autonomic
ProActive: Open Source GCM ref. Impl.

- **Open to**
  - Industry Standards

- **Effective:**
  - Used for The Grid Plugtests
  - Over 2000 CPU across the world at once
Update on ProActive and GCM (5)

New GUIs

- IC2D Eclipse
- TimIt

GRID IDE: Integrated Development Environment

GRID IDE Overview
ProActive features an Integrated full-fledged Grid IDE (Integrated Development Environment) named IC2D. It features graphical monitoring and control, programming wizards, debugging and optimizing tools which all contribute to high-productivity grid development. Moreover, enterprise developers can work in a familiar setting within Eclipse plugins, increasing both productivity and reducing the need for new training.

Graphical Monitoring View
A graphical environment for remote monitoring and steering of distributed and grid applications. It provides a graphical visualisation for hosts, Java Virtual Machines, and active objects, including the topology and the volume of communications.

IDE HIGHLIGHTS
- Health of your applications including Graphical Monitoring and Control, Programming Wizards, Debugging and Optimizing Tools

Fractal GUI
A graphical tool to edit Fractal and Grid Component configurations.

The GUI allows for Client/Server Interface connections, and managing of Virtual Nodes for controlling mapping and co-allocation of deployment.

TimIT Bench
A complete solution to benchmark and optimize applications’ performance.
TimIT is able to produce a large variety of statistics, advanced timers with hierarchical capabilities.

TimIT automatically generates statistical charts.
GCM for Code Coupling: Vibro Acoustic (courtesy of EADS)
GCM for Electromagnetism: Jem3D
Installing JECs is like any other Java application. You will extract the contents of the JECs archive into a location, and point your CLASSPATH to the .jar file in that location.

When you unpack the .jar archive or get it from the CVS repository, you will get an application directory, in which the .jar file is located. You will find a doc directory containing the documentation. The src directory contains all source code. The build directory contains everything you need to reproduce JECs. Instruction is included in the archive file.
<table>
<thead>
<tr>
<th>Experiment</th>
<th>Mesh size</th>
<th>Number of Processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$41 \times 41 \times 41$</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>$81 \times 81 \times 81$</td>
<td>70</td>
</tr>
<tr>
<td>3</td>
<td>$201 \times 201 \times 201$</td>
<td>130</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>138</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>258</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>258</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>308</td>
</tr>
</tbody>
</table>

**Component based vs Object based**

![Bar chart comparison of component-based vs object-based execution times and computation times across different experiments on Grid5000 with several clusters.]
Conclusion

GCM Road Map:

- **CoreGrid**: Assess GCM in next weeks
- **GridCOMP**:
  - experiments with IBM, Atos, GridSystems, partners (Chinese, Australia, South America)
- **4th Grid Plugtests**: GCM Tests?
- Next GCM description in CoreGRID and GridCOMP