Grid programming with components: an advanced component platform for an effective invisible grid

**GCM Non-Functional Features and ProActive**

Marco Aldinucci

&

M. Danelutto, S. Campa, D. Laforenza, N. Tonellotto, P. Dazzi

Unipisa & ISTI-CNR

e-mail: aldinuc@di.unipi.it
Outline

- Not really Proactive user case
  - Bringing some ideas
  - Proposed for GCM (CoreGRID/GridCOMP)
  - Experienced with ASSIST
  - Also, currently experimenting using ProActive

- Proactive User case
  - Already described last monday
  - I repeat if time
GridComp MODEL key points

- Hierarchic model
  - Expressiveness
  - Structured composition

- Interactions among components
  - Collective/group
  - Configurable/programmable
  - Not only RPC, but also stream/event

- NF aspects and QoS control
  - Autonomic computing paradigm
GCM Implementation Aspects
(in my viewpoint at least)

- Membrane is an active object
  - Centralized implementation
- Controller are components
  - One possible choice, among the others
  - Lightweight components
- Communication protocol
  - Asynchronous communications

Krakow feedback. Rodolfo Toledo, Eric Tanter, Jose Piquer: USING REFLEXD FOR A GRID SOLUTION TO THE N-QUEENS PROBLEM: A CASE STUDY. CoreGRID Integration Workshop, Karkow, October 2006
**Autonomic Computing Paradigm (AC)**

- Aims to tackle the complexity of QoS management providing self-managing components, i.e.:
  - Self-configuring
  - Self-optimizing
  - Self-healing
  - Self-protection

- Basically control loops
  - Basic theory dates back to last mid-century decade
  - Recently re-vamped and propelled by IBM
AC Bare Bones

A complex system is usually set up by distinct elements

- composed in horizontal fashion (i.e. used_by/provided_to)
- nested in vertical fashion (i.e. implemented_by)

AC idea:

- Each entity exhibits certain self-management capability
- At each level, entities cooperate to self-manage their aggregation
- Each level subsumes capability at the next level down
AN AC ELEMENT & ITS “HORIZONTAL” COMPANIONS

- AC element
  - Managed Element
  - Autonomic Manager

- AC elements co-operate to achieve a common goal
  - Possibly with dynamic patterns along running time
**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 mechanism to apply the plan
AC Element - ASSIST Experience

Some experiences already done

- Based on QoS contracts
- Autonomic parmod
- Autonomic supercomponents
  - Higher order components
  - DAG, Farm


....
## QoS contract Example (ASSIST)

<table>
<thead>
<tr>
<th>Perf. features</th>
<th>$QL_i$ (input queue level), $QL_o$ (input queue level), $T_{ISM}$ (ISM service time), $T_{OSM}$ (OSM service time), $N_w$ (number of VPMs), $T_w[i]$ (VPM$_i$ avg. service time), $T_p$ (parmod avg. service time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perf. model</td>
<td>$T_p = \max{T_{ISM}, \sum_{i=1}^{n} T_w[i]/n, T_{OSM}}, T_p &lt; K$ (goal)</td>
</tr>
<tr>
<td>Deployment</td>
<td>arch = (i686-pc-linux-gnu ∨ powerpc-apple-darwin*)</td>
</tr>
<tr>
<td>Adapt. policy</td>
<td>goal_based</td>
</tr>
</tbody>
</table>

Grid programming with components: an advanced COMPONENT platform for an effective invisible grid

CoreGRID: The European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and P2P Technologies
Exp 1: Stateless FARM

**Contract:**
- keep a given service time
- contract change along the run
Exp 2: Data-Parallel(STP)

Expected work balance among platforms

Platforms

A B C D

P3@868MHz  P4@2.5GHz  P4@2GHz  P4@2.8GHz

BogoMIPS

0 1,500 3,000 4,500 6,000

Platforms

A 11%  B 30%  C 24%  D 35%
Exp 2: Data-Parallel(STP)

Distribution of load among platforms (n. of VPs)

Relative Unbalance

Iteration time
### Overhead? (mSecs)

<table>
<thead>
<tr>
<th>parmod kind</th>
<th>Data-parallel (with shared state)</th>
<th>Farm (without shared state)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>add PEs</td>
<td>add PEs</td>
</tr>
<tr>
<td></td>
<td>remove PEs</td>
<td>remove PEs</td>
</tr>
<tr>
<td># of PEs involved</td>
<td>1→2 2→4 4→8</td>
<td>1→2 2→4 4→8</td>
</tr>
<tr>
<td></td>
<td>2→1 4→2 8→4</td>
<td>2→1 4→2 8→4</td>
</tr>
<tr>
<td>$R_l$ on-barrier</td>
<td>1.2 1.6 2.3</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>0.8 1.4 3.7</td>
<td>--</td>
</tr>
<tr>
<td>$R_l$ on-stream-item</td>
<td>4.7 12.0 33.9</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>3.9 6.5 19.1</td>
<td>--</td>
</tr>
<tr>
<td>$R_t$</td>
<td>24.4 30.5 36.6</td>
<td>24.0 32.7 48.6</td>
</tr>
<tr>
<td></td>
<td>21.2 35.3 43.5</td>
<td>17.1 21.6 31.9</td>
</tr>
</tbody>
</table>

GrADS papers reports overhead in the order of hundreds of seconds (K. Kennedy et al. 2004), this is mainly due to the stop/restart behavior, not to the different running env.
A distributed App is an assembly of components, which may be primitive or formed by other components.

The QoS of a component depends on its nested components and their functional relations. Components may include either sequential or distributed code.

Provided QoS can be synthesized in a bottom-up fashion, while requested QoS is imposed in top-down fashion. Application management can be distributed along the hierarchy to improve management locality.
Autonomic Cycle & Vertical

- Autonomic cycle manage some further points
  - Accepts new QoS contracts from father manager
  - Raises locally unmanageable contract violations
  - At each level, implements cooperation with other partners
- Formalization is an open problem
Horizontal & Vertical orchestration

Open problems

A satisfactory formalization is missing

- how describe QoS proprieties
- Describe distributed parametric analysis strategies & reconfiguration plans
  - How to generate them automatically, how to enforce locality of actions

Some experiences already done with ASSIST, some promising ideas

- Exploiting structured orchestration of activities (super-components)
Rationale

- AC promising
- Something can be already done
  - Experiences in ASSIST given good feedbacks in terms of reactivity, low-overhead, ...
  - Documented in literature
- Several, very interesting open problems
  - At the border with Global Computing community
  - Very interesting for EU VII FP
CoreGRID GCM NF features

- Autonomic behavior
  - EU 7 FP, NGG3, blah blah ...
- Renewed proposal based on:
  - Fractal style level of compliance
  - Passive or active vertical interaction
## Fractal Conformance levels

<table>
<thead>
<tr>
<th></th>
<th>Minor (κ)</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major (Θ)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Component</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component Type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Interface Type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Attribute, Content, Binding</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LifeCycle Controller</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Factory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Template</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Conformance level Θ.κ
Fractal Conformance levels Rephrased and GCM

- Major ($\Theta$) $\geq 1 \iff \text{“it is a component”}$

- Minor ($\kappa$) $\geq 1 \iff \text{“it exhibits AC, CC, BC, LC”}$

  - Minor ($\kappa$) = 2 & 3 have a bit uneven meaning (F, T)

- Add another counter describing NF behavior $\Theta.\kappa.\alpha$ (as partial function)

  - $\alpha = 0 \perp$, only if ($\Theta < 1$ or $\kappa < 1$) (observationally undecidable)
  - $\alpha = 1$ No autonomicity
  - $\alpha = 2$ Passive autonomicity (low-level, server only NF intf)
  - $\alpha = 3$ Active autonomicity (high-level, client/server NF intf)
Some Aspect still not Clear

Main concerns

- How much the model should be specified?
  - Not that much, at the end this is why we adopted Fractal ...
  - It should be a Model not the specification of an implementation
    - OO Model is not Java specification
  - Membrane

- Fractal/ProActive implementation
  - Maps 1:1 to GCM reference implementation?
  - Are group communications implemented by controllers?
  - Controllers=components? (in which component model?)
  - How controllers interoperate and how are programmed?
  - Is membrane admitting a distributed implementation?