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

WP3

UPDATE ON NON FUNCTIONAL FEATURES

M A R C O  A L D I N U C C I
&
M. D A N E L U T T O , S. C A M P A ,

U N I P I S A & I S T I - C N R

e-mail: aldinuc@di.unipi.it
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
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
## Fractal Conformance levels

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>Component</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>
</tr>
<tr>
<td>Component Type</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>
</tr>
<tr>
<td>Attribute, Content, Binding</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>
</tr>
<tr>
<td>Factory</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>
</tr>
</tbody>
</table>

Conformance level Θ.κ
Fractal Conformance levels Rephrased and GCM

- Major \( (\Theta) \geq 1 \Leftrightarrow \) “it is a component”
  - Minor \( (\kappa) \geq 1 \Leftrightarrow \) “it exhibits AC, CC, BC, LC”
    - Minor \( (\kappa) = 2 \& 3 \) have a bit uneven meaning \((F, T)\)
- Add another counter describing NF behavior \( \Theta \cdot \kappa \cdot \alpha \) (as partial function)
  - \( \alpha = 0 \perp \), only if \((\Theta < 1 \text{ or } \kappa < 1)\) (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)
Several Aspect still not Clear

- Relation between Fractal and GCM
  - Conformance levels, Sharing, Client NF ports

- Introspection & Intercession
  - Intercession is mentioned just in the intro of Fractal specification, not sure the concept has been correctly interpreted in GCM
  - Life cycle too restrictive
    - Why require to stop all components to change bindings?

- Membrane, what is?
  - Is group communication sem implemented by controllers?
  - Are controllers components? *(No, if possible)*
  - How controllers interoperate and how are programmed?
  - Has it a distributed implementation? *(Yes, if possible)*
Partial Conclusions (GCM)

- On going refinement
  - Avoid choices that make implementation too complex, or inefficient
  - Personally, not really liking Fractal approach on “everything is optional and can be under-specified”
    - What is a cat? A thing, at level 0, an animal at level 1, a feline at level 2 ....
- Early experimentation in GridCOMP is important
  - Usability feedback
  - Performance feedback
Our Fractal/ProActive experience (First 6 months)

Understanding
- Install, learn, understand Fractal & ProActive
- Understand Fractal/Proactive architecture
  - Documentation; not layered architecture
- Fractal interoperability
  - Proactive vs Julia implementations
  - AOP with Fractlet

Case study
- Self-optimizing only (performance)
- pipe(S1, Farm(S2), S3)
- Fractal/ProActive features to support NF control
A simple three stages application, working on a data stream (e.g. video frames)

- pipe performance $\max(T_g, T_{farm(f)}, T_h)$
- farm performance $T_f/#n$, $n$ variable along run

Self-optimizing w.r.t. nodes power along time
A simple three stages application, working on a data stream (e.g. video frames)

- pipe performance max(Tg, Tf, Th)
- farm performance Tf/#n, n variable along run

Self-optimizing w.r.t. nodes power along time
A simple three stages application, working on a data stream (e.g. video frames)

- pipe performance $\text{max}(T_g, T_{\text{farm}(f)}, T_h)$
- farm performance $T_f/\#n$, $n$ variable along run

Self-optimizing w.r.t. nodes power along time
**Farm**

A clean implementation needs:

- **Unicast “programmable” communications**
  - send to a single ID in a set, collect from any (not all)
  - probably not excluded by GCM specification, not clear our to implement in the current version

- **Distributed implementation of the membrane**
  - is it a single Active Objects?

Currently two inner components act as distributor and collector
Two versions

Passive inner components
- Each component exposes server NF interface (GetBandwidth)
- They are periodically polled from a controller in the membrane, which then expose a GetBandwidth server port for the pipe component
- Implementation pretty tricky, polling is programmed at hand within the controller

Active inner components
- How to open server ports on the membrane toward the inner part (import-binding)? Is it possible?
- We simulated with a functional component

Both versions expose all ports through a single JVM
- Membrane and Active Objects
Pipe with Passive NF stages

- Implemented, works
- Overheads not yet measured
- Managing code completely up to the user
- NF binding programmatically described
Pipe with Active NF Stages

Not succeed to express this

- Maybe not impossible, but we don’t succeeded in several weeks
- Can be simulated by inserting an functional component (explicit manager)
- Import/export bindings for NF controllers appears under-specified (-studied, -implemented)
Points needing further investigation

Programming controllers
- GCM specification should be refined

Interactions among controllers
- Ports exposed by controllers, toward in and out
- Interaction among ports

Mapping membrane & controllers
- VN, ActiveObjects, JVM, nodes, ...

Low-level points
- Sent to Proactive Q&A
Conclusion

High-level research issues
- Formalization of QoS property ongoing
- Interaction among managers is still a black hole

Implementation issues
- Middleware expressiveness/effectiveness tradeoff can (should?) be improved
- Low-level issues submitted to Proactive Q&A

Layering of features
- In our idea, some of middleware features may require a promotion to QoS features (e.g. load balancing, communication synchronicity, group communication semantics, security ...) because they are supposed to be dependent by semantics of GCM application not on ProActive