By defining the GCM, the V.I. aims at the precise specification of an effective Grid Component Model. The features are discussed taking Fractal as the reference model. The features are defined as extensions to the Fractal specification. The V.I. expects several different implementations of the GCM, not necessarily relying on existing Fractal implementations.
Outline

1. Requirements and strategy
2. A Summary of Fractal
3. Abstract Model of the GCM
4. Communication
5. Parallelism and Distribution
6. Dynamic Controllers
7. Support for Autonomicity
General Features

- Reflexivity
- Component hierarchy
- Extensibility of the model
- Support for adaptivity
- Language neutrality
- Interoperability

- Lightweight ➔ portable and compact implementations
- Well-defined semantics (allow future formalization)
Overall strategy

- What Grid specific features can already be taken into account by Fractal?
- What we need vs. what exists in Fractal?
- Propose extensions of the Fractal model for Grid specificities (some being proposed by the Fractal community)
- Ongoing work (inside CoreGrid): Specify precisely those extensions
- Future works/perspectives (outside CoreGrid): provide a (several) reference implementation(s)
Outline

1. Requirements and strategy
2. A Summary of Fractal
3. Abstract Model of the GCM
4. Communication
5. Parallelism and Distribution
6. Dynamic Controllers
7. Support for Autonomicity
GCM is Based on Fractal

Fractal provides:

- *Terminology, API and ADL ➔ Interoperability*
- *Hierarchical structure*
- *Separation of concerns*
- *Abstract component model ➔ no constrain on implementation: several implementations exist*
- *Multi-level specification: almost every object is a level 0 Fractal component*

We can imagine a multi-level specification of the GCM ➔ We focus on the Grid specific extensions of Fractal
A Fractal Component
Outline

1. Requirements and strategy
2. A Summary of Fractal
3. Abstract Model of the GCM
4. Communication
5. Parallelism and Distribution
6. Dynamic Controllers
7. Support for Autonomicity
Abstract Model

1. Component Specification as an XML schema or DTD
2. Run-Time API defined in several languages
3. Packaging described as an XML schema
   cf. Fractal packaging
XML description of a Component

- Definition of Primitive Components
- Definition of Composite Components (composition)
- Definition of Interfaces (ports)
  - Server, Client, event, stream, etc.
- Including external references to various specifications:
  - Java Interface, C++ .h, Corba IDL, WSDL, etc.
- Specification of Grid aspects:
  - Parallelism, Distribution, Virtual Nodes,
  - Performance Needs, QoS, etc.
Interoperability

(systematically) export WSDL interfaces.

Possibility to define glue components.

API as part of the GCM
Communications

Asynchronous communications

- Fractal:
  - somewhat “unspecified”,
  - notion of “address space” for distinguishing primitive and composite bindings,
  - most implementations use synchronous primitive bindings, but not all of them,
  - dream project use composite bindings

- In the GCM:
  - Semantics should be specified in the interfaces
  - asynchronous method call is the default
  - Implementation details purposely unspecified
Outline

1. Requirements and strategy
2. A Summary of Fractal
3. Abstract Model of the GCM
4. Communication
5. Parallelism and Distribution
6. Dynamic Controllers
7. Support for Autonomicity
Support for Parallelism and Distribution

- Parallel Components with the notion of Virtual Nodes
- Collective Interfaces
  - Multicast
  - Gathercast
Parallel Components: Distribution

- Notion of Virtual Nodes ➔ distribution
  - Maps the virtual architecture to a physical one
  - One can envisage more sophisticated information such as, for instance, topology information, QoS requirements between the nodes, etc.

- Parallel components can
  - Be distributed or not
  - Admit several implementation
  - ➔ adaptive implementations
Virtual Nodes

```xml
<virtualNodesDefinition>
  <virtualNode name="Dispatcher" property="unique_singleA0" />
  <virtualNode name="Renderer" property="Multiple"
              constraintFile="RendererConstraints.xml" />
</virtualNodesDefinition>
```

Permits a program to generate automatically a deployment plan: find the appropriate nodes on which processes should be launched.

In the future, we envisage the adjunction of more sophisticated descriptions of the application needs with respect to the execution platform: topology, QoS, …
Virtual Nodes in the ADL

```xml
<exportedVirtualNodes>
  <exportedVirtualNode name="VN1">
    <composedFrom>
      <composingVirtualNode component="this" name="myNode"/>
    </composedFrom>
  </exportedVirtualNode>
</exportedVirtualNodes>

<virtual-node name="myNode" cardinality="single"/>
```

- Renames a VN
- Exports a VN name

➤ The final version of the GCM specification will precisely define the syntax for the virtual node definition, and their composition.
Collective interfaces

- Multicast
- Gathercast
- Gathermulticast

» Allow MxN communications:
  - Redistribution
  - Direct communication or intermediate composite?
Current situation (Fractal model)

Simple type system
Component type ↔ types of its interfaces
Interface type:
- Name
- Signature
- Role
- Contingency
- Cardinality
Current situation in Fractal: 2 cardinalities

Single

Collection
Proposal

Simplify the design and configuration of component systems

Expose the collective nature of interfaces
- Cardinality attribute
- Multicast, gathercast, gather-multicast

The framework handles collective behaviour at the level of the interface

Based on Fractal API:
- dedicated controller
- interface typing
Multicast interfaces
Multicast interfaces

Transform a single invocation into a list of invocations

Multiple invocations
- Parallelism
- Asynchronism
- Dispatch

Data redistribution (invocation parameters)
- Parameterisable distribution function
- Broadcast, scattering
- Dynamic redistribution (dynamic dispatch)

Result = list of results
a. broadcast invocation parameter received in server component

b. scattered invocation parameter
Multicast interfaces

Results as lists of results
Invocation parameters may also be distributed from lists
Gathercast interfaces
Gathercast interfaces

*Transform a list of invocations into a single invocation*

Result: redistribution of results

Synchronization of incoming invocations
- ~ “join” invocations
- Timeout / drop policy
- Bidirectional bindings (callers ↔ callee)

Data gathering
Aggregation of parameters into lists
Gather-multicast interfaces
**API**

**InterfaceType** interface:
```
string getFcItfCardinality();
```

**TypeFactory** interface:
```
InterfaceType createFcItfType (  
    ...  
    string cardinality  
) ...  
```

**CollectiveInterfacesController**
- Collective interface policy
- On a per-interface basis
- Specified at instantiation-time
- May be updated dynamically
Outline

1. Requirements and strategy
2. A Summary of Fractal
3. Abstract Model of the GCM
4. Communication
5. Parallelism and Distribution
6. Dynamic Controllers
7. Support for Autonomicity
Dynamic Controllers

As suggested as an extension of Fractal, controllers can be components (they still belong to the membrane)
Dynamic Controllers (2)

- Adaptativity
- Dynamic reconfiguration of the controllers
- Controllers can now have server and client interfaces
  - N.F client interfaces
- Called controller components
- Better separation of concerns (layered components)
- Modification of the content controller (for the membrane)
- Controller components should be lightweight components
Outline

1. Requirements and strategy
2. A Summary of Fractal
3. Abstract Model of the GCM
4. Communication
5. Parallelism and Distribution
6. Dynamic Controllers
7. Support for Autonomicity
Autonomicity

- **Self-Configuring**: handles reconfiguration inside itself
- **Self-Healing**: provides its services in spite of failures
- **Self-Optimising**: adapts its configuration and structure in order to achieve the best/required performance.
- **Self-Protecting**: predicts, prevents, detects and identifies attacks, and to protect itself against them.

- Open and extensible specification
  ➔ Several levels of autonomicity depending on:
  - autonomic controllers implemented
  - autonomicity level implemented by each controller
Autonomicity (2)

- Level of autonomicity not defined in the GCM (type *any*)
- Already defined API:

```java
interface SelfConfigurationController {
    void setConfigurationGoal(any goalDescription);
}
```

- Should compose with *hierarchy* and use *component controllers*
Conclusion

Future works: model has to be refined – Technical issues – APIs – extended ADL …

Like in Fractal, we aim at a multi-level specification, an implementation of the GCM can be level 1.1 Fractal compliant and level 1.2.1 GCM compliant. GCM levels to be specified

Summary: See table
<table>
<thead>
<tr>
<th>Requested feature</th>
<th>Concept in GCM to achieve it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical composition</td>
<td>Fractal’s component hierarchy</td>
</tr>
<tr>
<td>Extensibility</td>
<td>• From Fractal design&lt;br&gt;• dynamic controllers (any non-functional feature can be added to any component)&lt;br&gt;• open and extensible communication standard</td>
</tr>
<tr>
<td>Support for reflexivity (introspection and intercession)</td>
<td>From Fractal specification and API</td>
</tr>
<tr>
<td>Lightweight</td>
<td>• Support for adaptivity in the component model and extensibility both in the component model and the specification&lt;br&gt;• Conformance levels&lt;br&gt;• No controller imposed</td>
</tr>
<tr>
<td>Well-defined Semantics</td>
<td>API + ADL + well-defined extensions – via the (future versions of the) current document</td>
</tr>
<tr>
<td>ADL with support for deployment</td>
<td>Virtual Nodes + cf Section 5.1</td>
</tr>
<tr>
<td>Packaging</td>
<td>cf. packaging being defined by the Fractal community</td>
</tr>
<tr>
<td>Support for Higher-order component programming skeleton-parallel, functional programming</td>
<td>language neutrality + partial support for higher-order via controllers, and especially controller components</td>
</tr>
<tr>
<td>Sequential and parallel implementation</td>
<td>XML component specifications, and Multicast-Gathercast interfaces allow plugging and unplugging several components to the same interface dynamically</td>
</tr>
</tbody>
</table>
| Asynchronous ports and Extended / Extensible port semantics | Asynchronous Method Invocation as the default semantics but any semantics can be defined via tags; special support for streaming  
⇒ Possibility to support method calls / message oriented / streaming / still unknown kinds of communication |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Group related communication on interfaces</td>
<td>Multicast / Gathercast interfaces</td>
</tr>
<tr>
<td>MxN communications</td>
<td>Multicast / gathercast interfaces (to be refined)</td>
</tr>
</tbody>
</table>
| Adaptivity:  
• Exploit Component Hierarchical abstraction for adaptivity  
• Ability to plug/unplug components dynamically  
• Give a standard for adaptive behavior and unanticipated extension of the model  
• Give a standard for the management autonomic components  
• Plug/unplug non-functional interfaces | Globally due to dynamic controllers  
Dynamic controllers  
Fractal’s content and binding controllers  
Dynamic controllers  
autonomic controllers  
dynamic controllers |
| Support for deployment | Notion of virtual nodes  
ADL with support for deployment |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel binding: Well-defined and verifiable composition</td>
<td>Multicast / gathercast interfaces</td>
</tr>
</tbody>
</table>
| Language neutrality | • API in various languages  
• various interface specifications to be used (IDL, Java, WSDL, etc.)  
• “Systematic” exportation of a web-service port |
| Interoperability | exportation and importation as web-services |