# The Grid Component Model: an Overview

CoreGrid deliverable PM.002

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*

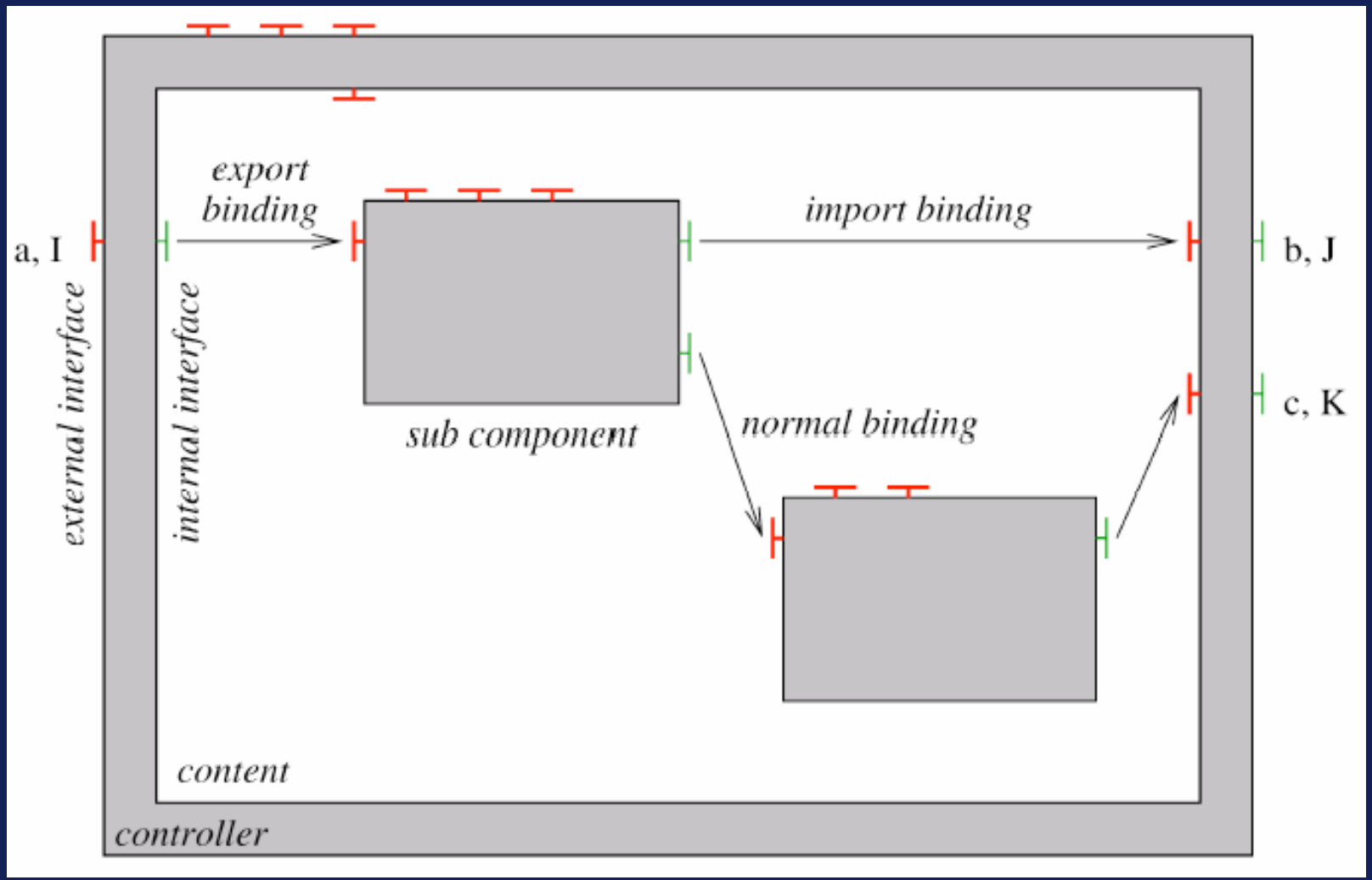
general features

- *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

```
<virtualNodesDefinition>
  <virtualNode name="Dispatcher" property="unique_singleAO"
      />
  <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

```
<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

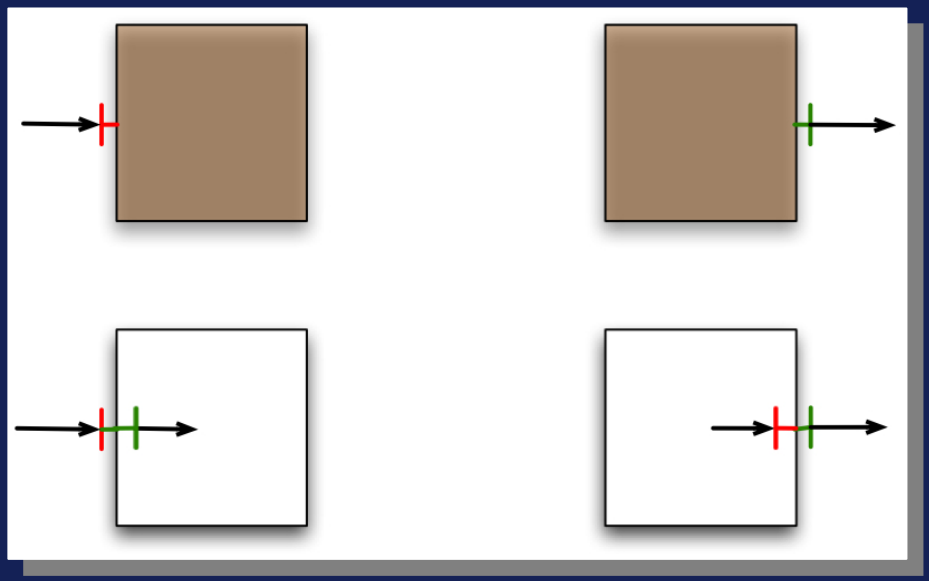
Component type ← types of its interfaces

Interface type :

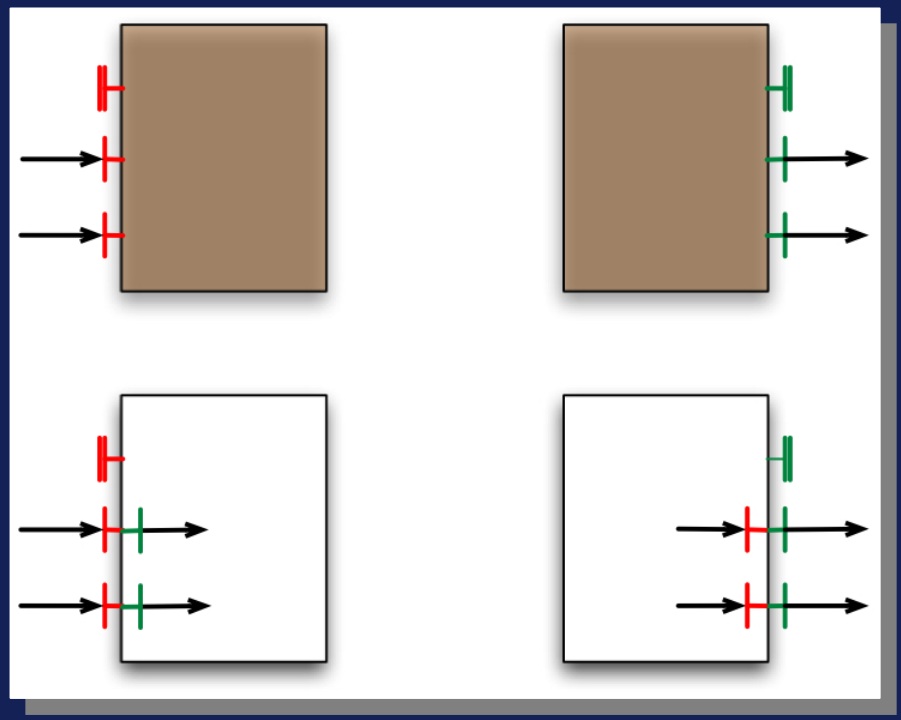
- Name
- Signature
- Role
- Contingency
- Cardinality

# Current situation in Fractal: 2 cardinalities

## Single



## Collection



# Proposal

Simplify the design and configuration of component systems

Expose the collective nature of interfaces

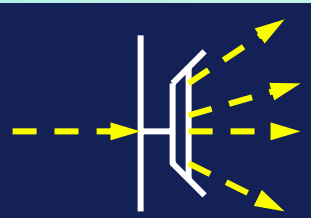
- Cardinality attribute
- Multicast, gathercast, gather-multicast

The framework handles collective behaviour at the level of the interface

Based on Fractal API :

- dedicated controller
- interface typing

# Multicast interfaces



*Transform a single invocation into a list of invocations*

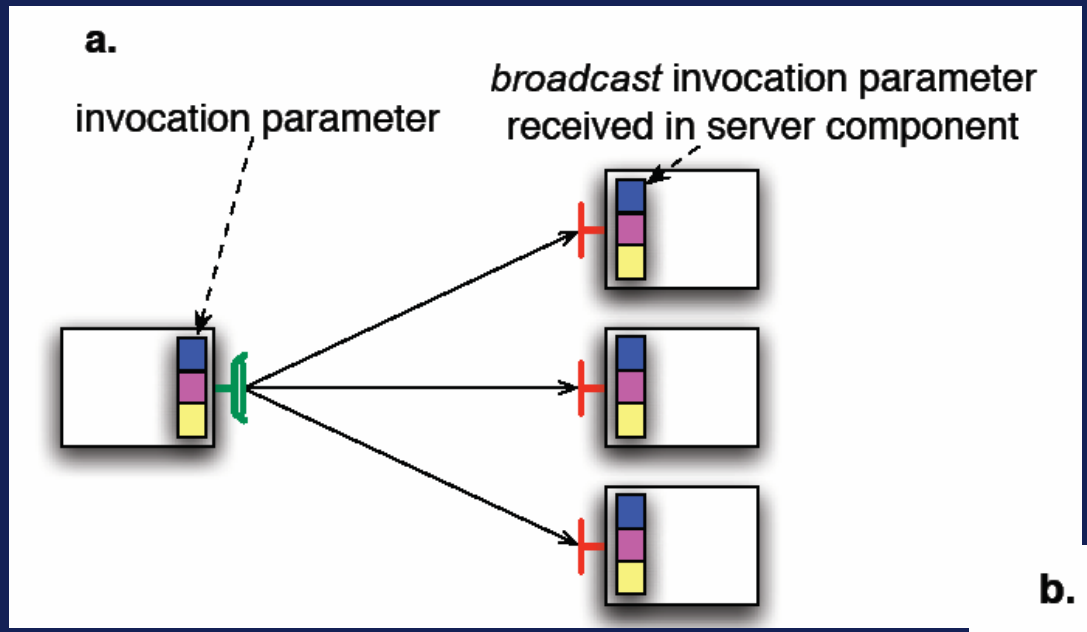
Multiple invocations

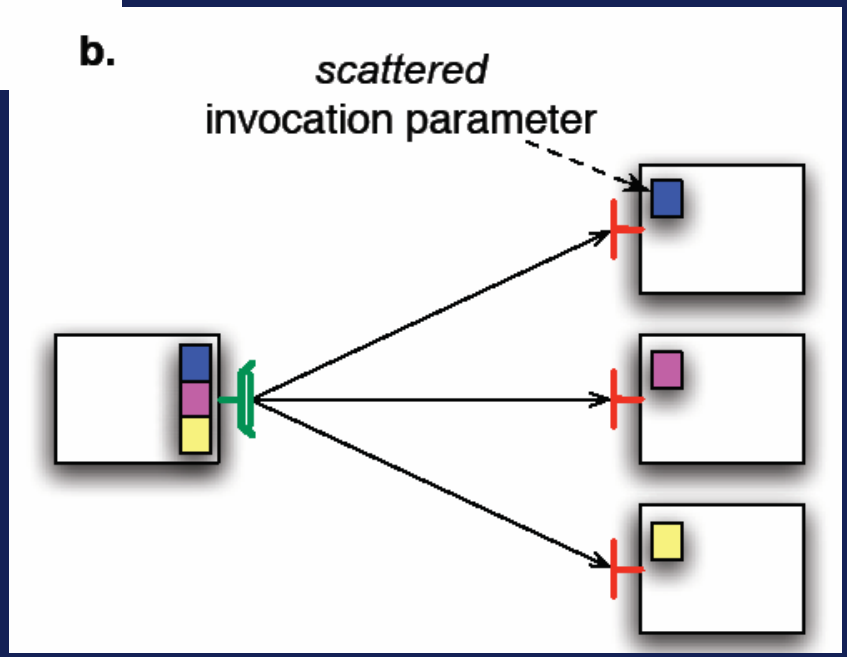
- Parallelism
- Asynchronism
- Dispatch

Data redistribution (invocation parameters)

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

Result = list of results

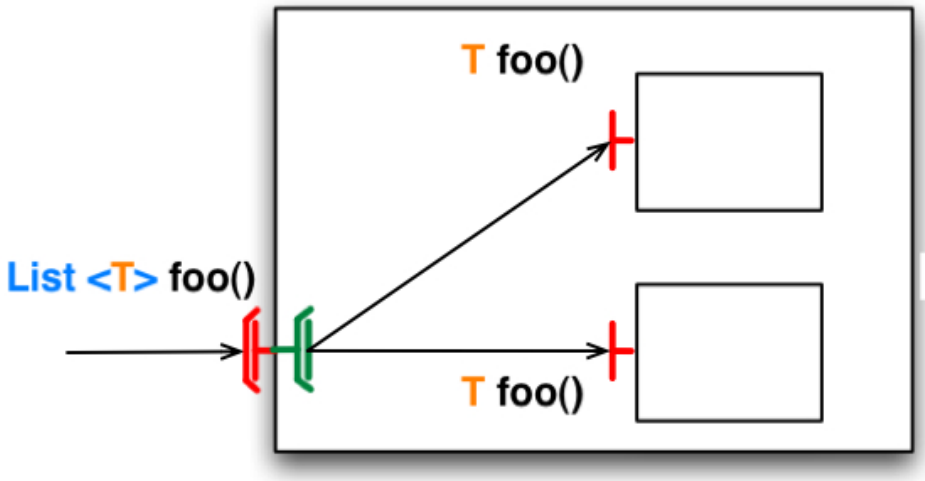

a.
invocation parameter
broadcast invocation parameter
received in server component

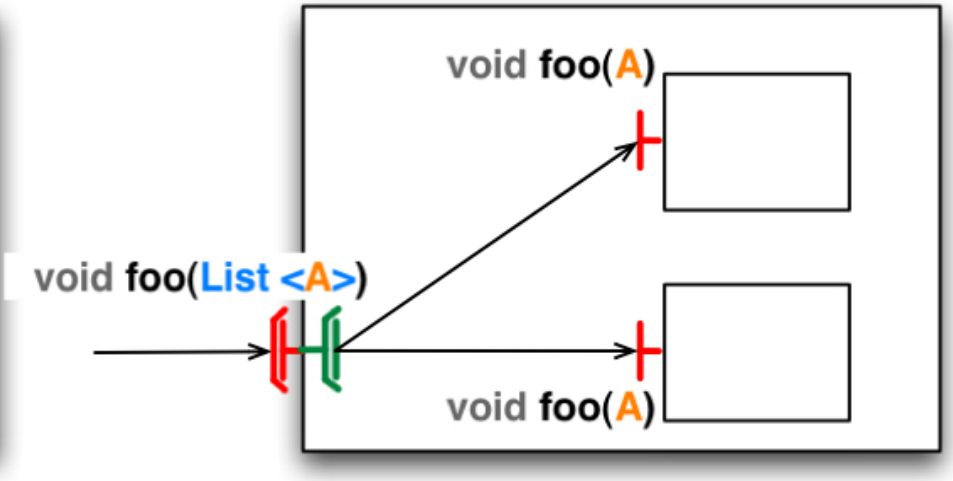

b.
scattered
invocation parameter

# Multicast interfaces

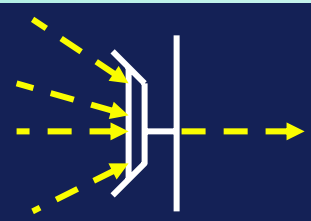
Results as lists of results

Invocation parameters may also be distributed from lists



# Gathercast interfaces



*Transform a list of invocations into a single invocation*

Result: redistribution of results

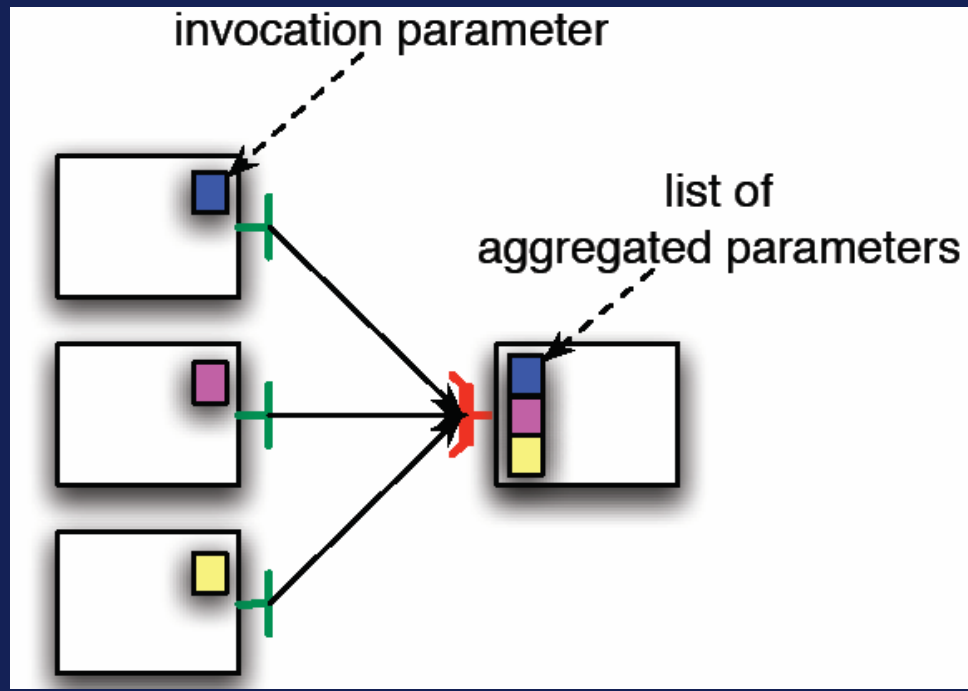
Redistribution function

Synchronization of incoming invocations

- ~ “join” invocations
- Timeout / drop policy
- Bidirectional bindings (callers ⇔ callee)

Data gathering

Aggregation of parameters into lists



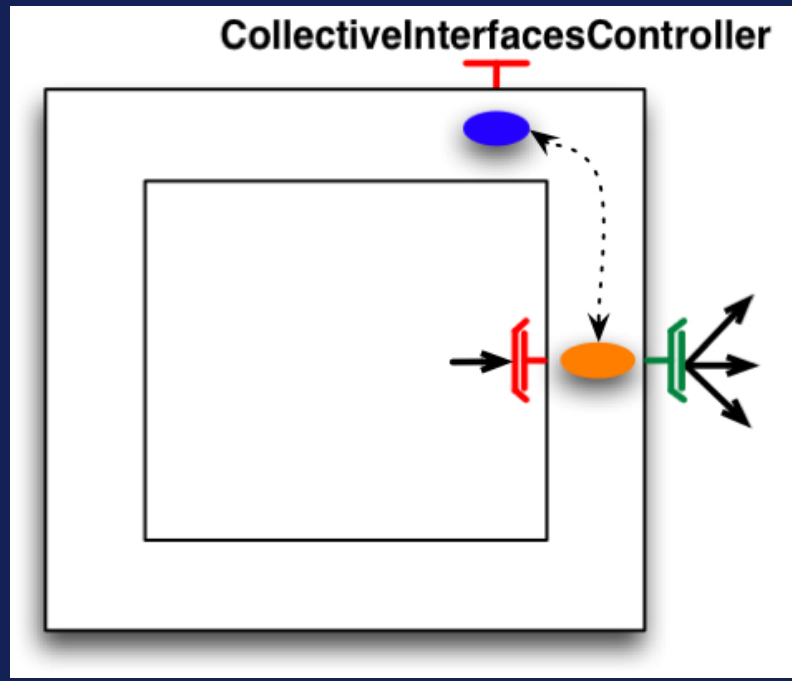
# API

`InterfaceType` interface:

```
string getFcItfCardinality();
```

`TypeFactory` interface:

```
InterfaceType createFcItfType (
    …
    string cardinality
)…
```




`CollectiveInterfacesController`

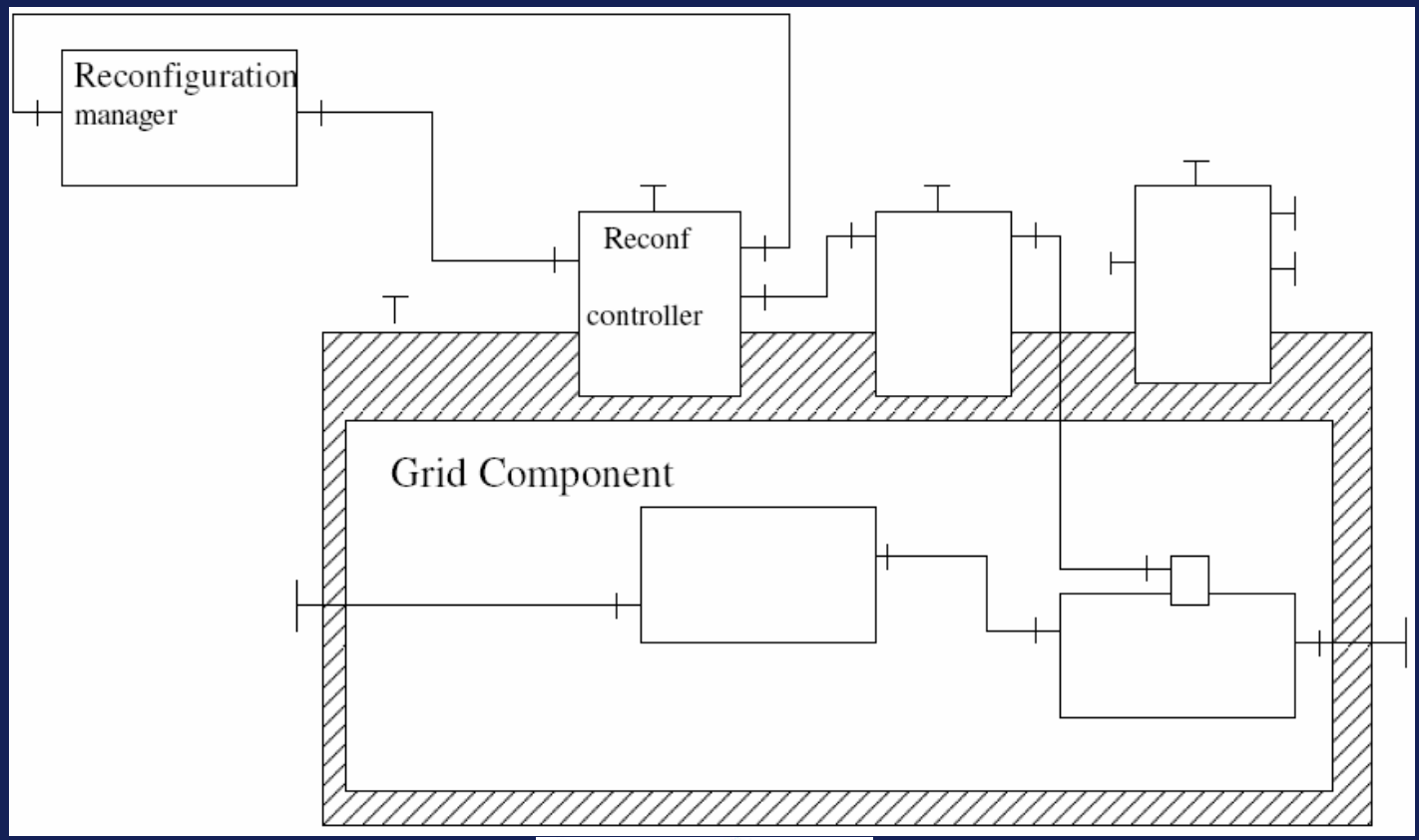
- Collective interface policy
- On a per-interface basis
- Specified at instantiation-time
- May be updated dynamically

# 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:

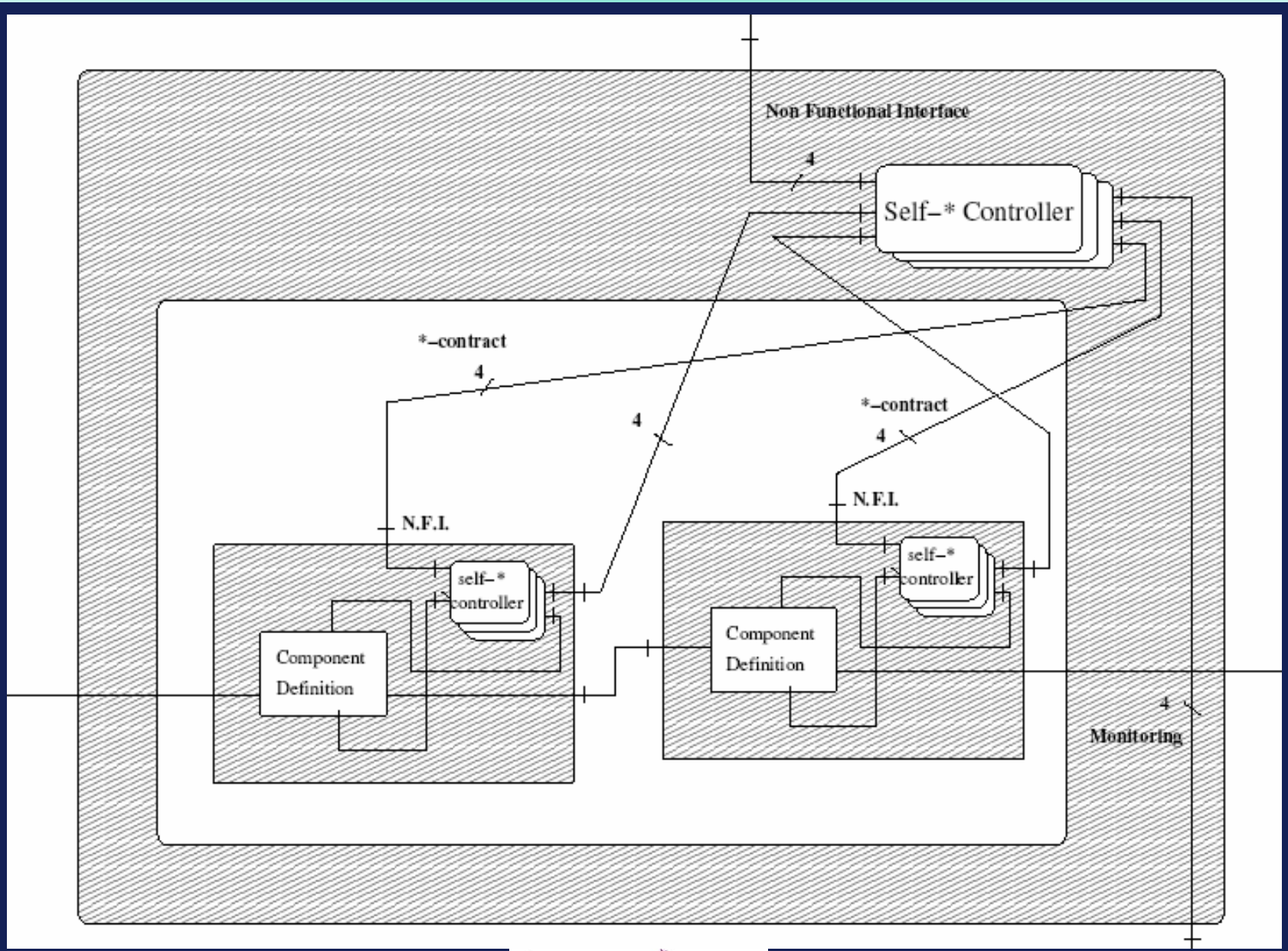
  - autonomic controllers implemented
  - autonomicity level implemented by each controller

# Autonomicity (2)

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

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

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


Non Functional Interface
4
Self–* Controller
*–contract
4
4
*–contract
4
N.F.I.
N.F.I.
self–*
controller
self–*
controller
Component
Definition
Component
Definition
4
Monitoring

# 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

# Requirements and Concepts

Hierarchical composition → Fractal

Extensibility → From Fractal design
→ dynamic controllers (for non-functional)
→ open and extensible communication mechanisms

Support for reflection → Fractal specification and API

Lightweight → Support for adaptivity and extensibility
→ Conformance levels
→ No controller imposed

ADL with support for deployment → Virtual Nodes

Packaging → packaging being defined by the Fractal community

Support for deployment → Notion of virtual nodes
→ ADL with support for deployment

Sequential and parallel implementation → XML component specifications, and Multicast-Gathercast interfaces allow plugging and unplugging several componentsto the same interface dynamically

Asynchronous ports and Extended/Extensible port semantics

→ Asynchronous Method Invocation as default but can be defined via tags; + Possibility to support method calls / message oriented / streaming / …

Group related communication on interfaces

→ Multicast / Gathercast interfaces

Interoperability → Exportation and importation as web-services

Language neutrality → API in various languages

→ Various interface specifications

→ exportation of a web-service port

Adaptivity: → Globally due to dynamic controllers

- Exploit Component Hierarchical abstraction for adaptivity
  → Dynamic controllers
- plug/unplug component → Fractal: content + binding controller
- Give a standard for adaptive behavior and unanticipated extension of the model → Dynamic controllers
- Give a standard for the autonomic management components
  → Autonomic controllers
- Plug/unplug non-functional interfaces → Dynamic controllers

Parallel binding: Well-defined and verifiable composition
→ Multicast / Gathercast