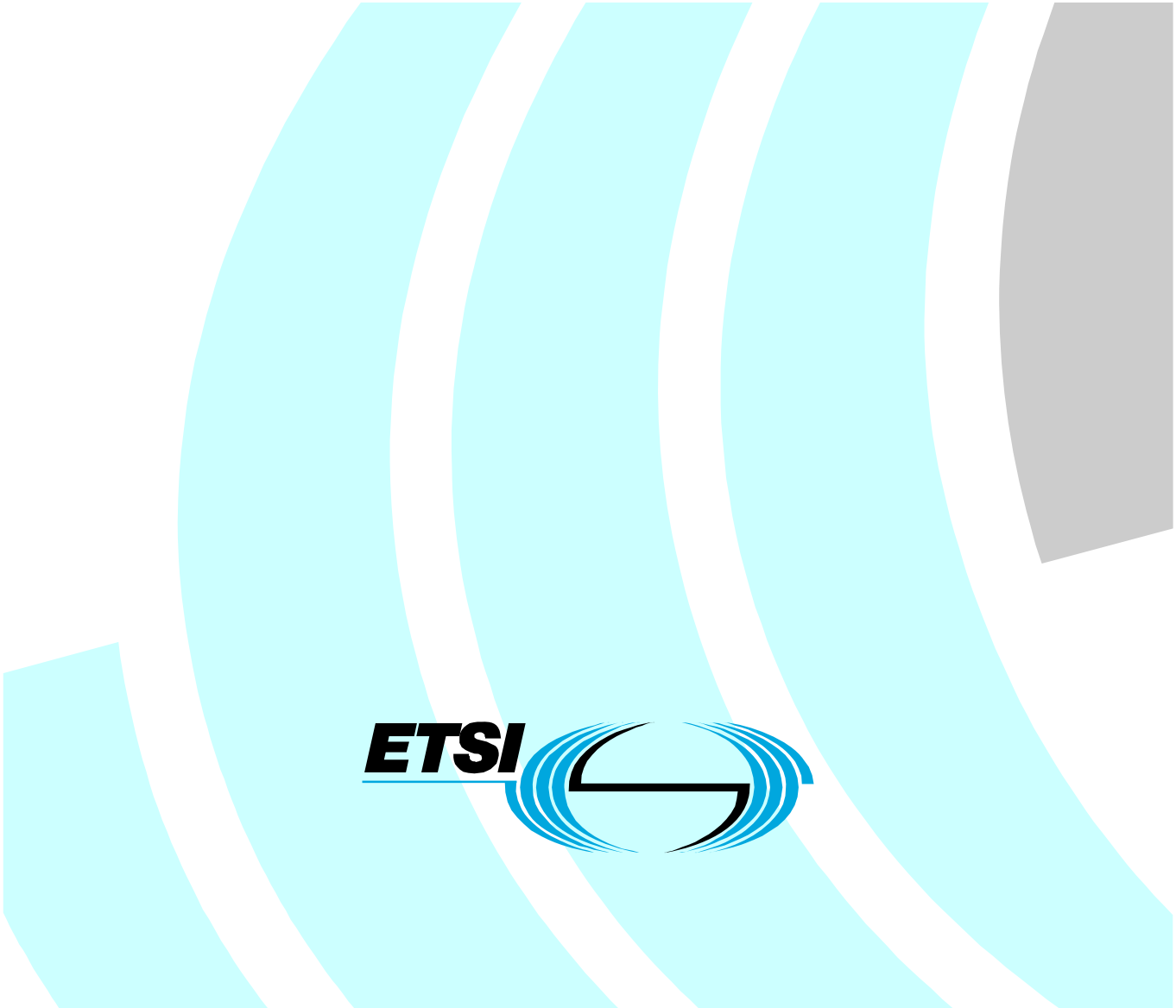
ETSI TS 102 829 V1.1.1 (2009-03)

Technical Specification

GRID;
Grid Component Model (GCM);
GCM Fractal Architecture Description Language (ADL)



Reference
DTS/GRID-0004-3 GCM_FractalADL

Keywords
architecture, network, interoperability, service

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights 4
Foreword 4
Introduction 4
1 Scope 5
2 References 5
2.1 Normative references 5
2.2 Informative references 5
3 Definitions and abbreviations 6
3.1 Definitions 6
3.2 Abbreviations 6
4 Overall Structure of the ADL 7
4.1 Principles 7
4.1.1 Component Structure 7
4.1.2 Interfaces 7
4.1.3 Deployment 8
4.1.4 Behaviour 8
4.1.5 Relationship with the Management API 8
4.2 component and definition 8
4.3 interface 11
4.4 virtualNode 12
4.5 exportedVirtualNode 12
4.6 composingVirtualNode 13
4.7 binding 13
4.8 content 13
4.9 attributes 14
4.10 controller 14
4.11 behaviour 15
4.12 comment 15
Annex A (normative): GCM ADL Schema 16
Annex B (informative): Examples of ADL files 19
B.1 Primitive components 19
B.2 Composite components 20
B.3 Virtual nodes 20
B.4 Component with multicast interfaces 21
B.5 Component with gathercast interfaces 22
B.6 Behaviour 22
Annex C (informative): Bibliography 23
History 24

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://webapp.etsi.org/IPR/home.asp).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee GRID (GRID).

The present document is related to documents TS 102 827 [2] (GCM Interoperability Deployment), TS 102 828 [1] (GCM Interoperability Application Description). It discusses the Fractal ADL, a description language for software components, based on XML.

Introduction

The GCM has been first defined in the NoE CoreGRID (42 institutions). The GridCOMP EU project (FP6, from June 2006 to February 2009) is working to further assess and experiment with the specification. A reference Open Source implementation has been tested in the 4 previous GRID Plugtests organized from 2004 to 2008 by ETSI. The 5th GRID plugtest in 2008 directly referred to the GCM, and was open to any GCM implementation.

In the present document we present a language for the description of software components. The GRID component model (GCM) aims to ease the development, deployment and maintenance of software systems. It is modular, extensible, and can be used in various programming languages. The main design principle is the separation of interface and implementation.

The GCM ADL is the base Architecture Description Language of this model. It is described here, in the form of an XML schema.

Historically, GCM is an extension of the Fractal component model, which defines an open component model for component systems, but has a very low support for distributed components.

1 Scope

The present document describes an XML based Architecture Description Language (ADL), used to define the composition of software components in distributed and parallel infrastructures.

The standard will help enterprises and laboratories to manage large-scale computer and telecom infrastructures with the necessary virtualization.

Its primary audience are grid system developers who need to specify complex applications by composing existing software components.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific.

- For a specific reference, subsequent revisions do not apply.
- Non-specific reference may be made only to a complete document or a part thereof and only in the following cases:
  - if it is accepted that it will be possible to use all future changes of the referenced document for the purposes of the referring document;
  - for informative references.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

2.1 Normative references

The following referenced documents are indispensable for the application of the present document. For dated references, only the edition cited applies. For non-specific references, the latest edition of the referenced document (including any amendments) applies.

[1] ETSI TS 102 828: "GRID; Grid Component Model (GCM); GCM Application Description".

[2] ETSI TS 102 827: "GRID; Grid Component Model (GCM); GCM Interoperability Deployment".

2.2 Informative references

The following referenced documents are not essential to the use of the present document but they assist the user with regard to a particular subject area**.** For non-specific references, the latest version of the referenced document (including any amendments) applies.

[i.1] "The Fractal Component Model".

NOTE: Available at http://fractal.objectweb.org/specification/index.html.

[i.2] CoreGrid NoE (FP6): "Basic Features of the Grid Component Model".

NOTE: Available at http://www.coregrid.net/mambo/images/stories/Deliverables/d.pm.04.pdf.

[i.3] GCM: "A Grid extension to Fractal for Autonomous Distributed Components", F. Baude, D. Caromel, C. Dalmasso, M. Danelutto, V. Getov, L. Henrio, C. Perez. Annals of Telecommunications - The Fractal Initiative, 2008.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

client interface: component interface that emits operation invocations

component: abstraction of a software entity with a well-defined interface for both server and client parts

component content: is (recursively) made of subcomponents and bindings

component controller: abstract entity that embodies one aspect of a component control : a component controller manages the component with respect to a given non-functional aspect

component model: specification of how components are defined and interact together

control interface: component interface that manages a "non functional aspect" of a component, such as introspection, configuration, and so on

definition: definition is the root element of a component structure

fractal: modular and extensible component model

functional interface: component interface that corresponds to a provided or required functionality of a component, as opposed to a control interface

gathercast interface: interface that is able to collect invocations from a set of sources and be plugged to a single destination

membrane: control part of a component, it is mainly constituted of a set of Component Controllers

multicast interface: interface that is able to transmit a single invocation, coming from a single source, to a set of destinations

server interface: component interface that receives operation invocations

virtual node: after the deployment of a GCM application, refers to a set of Nodes, which is seen as a single entity

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ADL	Architecture Description Language
API	Application Programming Interface
FC2	Format Commun (Version 2)
FIACRE	Intermediate Format for the Architectures of Embedded Distributed Components
GCM	Grid Component Model
VCE	Vercors Component Editor
VN	Virtual Node
XML	eXtensible Markup Language

4 Overall Structure of the ADL

4.1 Principles

The GCM ADL is an extensible language to define component architectures for the GCM component model ([i.2],[i.3]). It is an extension of the Fractal component model [i.1]. Consequently, the present document can also be considered as a definition of Fractal's ADL. Elements specific to GCM are the export and composition of virtual nodes, and the gathercast and multicast cardinalities. The ADL is made of a set of modules, each module defining the syntax for an architectural "aspect" of the language (e.g. interfaces, bindings, attributes, etc.).

4.1.1 Component Structure

The main element of the ADL is the Component. The most basic component is a primitive component that is a component entirely implemented in a programming language (typically Java). Primitive components can be combined into composite components. More generally, a composite component is a component built by aggregating several components, primitive or composite. Composite components are characterized by the fact that their internal structure is specified. Primitive components encapsulate the business code of the application.

When aggregating components into a composite component, the sub-components interfaces are bound one with another according to their respective nature and role (e.g. a component's provided interface X will be bound another component's required interface Y). Type compatibility can be required in order to bind together two interfaces.

A component is driven through controllers. Controllers are interfaces as well, but describing actions that are irrelevant to the component's main functionality. For instance, starting and stopping the component, binding it to another component, etc. The GCM Management API standardizes many of these controllers. Controllers specify the "non-functional" features of the components. Crucial controllers allow the introspection, modification of component structure, and lifecycle control of components.

The current version of the ADL does not allow for having structured controller components in the membrane. This should be standardized in future versions.

Likewise, a component may have attributes, which are key/value pairs that can be used to parameterize the component.

4.1.2 Interfaces

A component's relationship to the "external world" is defined by its interfaces. There are two kinds of interfaces, the ones the component provides ("server" role), and the ones it requires ("client" role). For instance a "calculator" component may provide an interface consisting of "add", "subtract", multiply", and "divide", while a "spreadsheet" component would require a component providing the "calculator" interface to work.

To a server interface of a composite component, a corresponding internal client interface with the same name is automatically generated. Similarly, a client interface of a composite component is automatically associated an internal server interface. Those interfaces allow the export of interfaces of inner components to the outside world, i.e. they are connected to complementary interfaces of inner components

Interfaces have four possible cardinalities: single, collection, gathercast, or multicast.

A *single* client interface can be bound to only one server interface.

A *collection* interface is an interface that will be duplicated several times at runtime. It can be viewed as an array of identical interfaces.

A *multicast* interface allows one-to-many communication patterns. A client multicast interface (possibly internal) can be plugged to many server interfaces instead of one for a single interface. The dispatch policy for the invocation can be parameterized. As ADL does not specify internal interfaces, a multicast server interface of a composite is in fact the composition of a single server interface and a multicast internal client interface.

A *gathercast* interface allows many-to-one communication patterns. A server gathercast interface can be bound from many client interfaces. Contrarily to single interfaces which necessarily transmit as many invocation as they receive, gathercast interfaces can act as synchronization interfaces (synchronization barriers). The gathering and synchronization policy for the invocations can be parameterized. As ADL does not specify internal interfaces, a gathercast client interface of a composite is in fact the composition of a gathercast internal server interface and a single client interface.

4.1.3 Deployment

To enable the deployment of components on a grid environment, it is possible to specify the Virtual Node on which the component will be deployed. Additionally, the ADL can rename and compose virtual nodes of sub-components of a composite in order to expose different assemblies, and names of virtual nodes at a higher level in the hierarchy.

4.1.4 Behaviour

The behaviour element allows specification of the behaviour of a component. This allows using different semantic formalisms and underlying models to reach similar goals, i.e. to guarantee that components in a composite system behave smoothly together and respect some user requirements. The <behaviour> element has two forms, depending on its attributes. It can either point to a file which contains that behaviour's description, or contain the description itself. The description may be written in any of the supported behaviour specification languages.

4.1.5 Relationship with the Management API

A GCM ADL definition provides a static description of a component architecture. The instantiation into an actual set of running components is done through the GCM API, by a Factory object: the factory typically receives an ADL from which it is able to instantiate a component system; the system is created by invoking methods from the GCM Management API. The GCM API also provides tools to manipulate the components at runtime.

4.2 component and definition

The root element of the GCM ADL is the definition. A definition describes a single component structure. Within it you can find (among other things) declarations of (sub) components, interfaces, bindings, and virtual nodes.

There can be several components declared in a definition, since a component may be an aggregate of sub-components. The components themselves can feature the same kind of elements as the definition. The set of elements a component may contain depends on its type (primitive or composite). The structure of the 'definition' and 'component' elements are defined in this clause.

Alternatively, a subcomponent within a hierarchy can be defined independently, using the 'definition' attribute. If the definition attribute is present, only the 'name' attribute is required. This mechanism allows for managing libraries of component definitions, and yields reusability.

Both primitive and composite components are defined with the same XML element type.

The <definition> type has the following attributes:

- **name:** (string) the name of the component (required);
- **arguments:** comma-separated list of argument names (optional). The arguments work as variables which value can be fixed at load time;
- **extends:** (string) the name of the component which this component extends.

The <component> type has the following attributes:

- **name:** (string) the name of the component (required);
- **definition:** (string) the reference to a component (toplevel) ADL definition.

The <component> and <definition> types have also the following child elements:

- **comment:** a free form comment documenting the component (0-unlimited);
- **interface:** the description of the interface the component provides (0-unlimited);
- **component:** reference to a sub-component (0-unlimited);
- **binding:** binding of a client interface to a server interface (0-unlimited);
- **content:** the class which represents this component (0-1);
- **attributes:** the list of attributes of the component (0-1);
- **controller:** the controller of the component (0-1);
- **virtualNodes:** (sequence of virtualNode elements) the list of virtual nodes this component should be deployed on (0-1);
- **exportedVirtualNodes:** (sequence of exportedVirtualNode elements) (0-1).

Most often, the content child is required for primitive components. By convention, it is also possible for composite components, for example to define the implementation class of the AttributeController of the component.

As a consequence, a primitive component is characterized by a non-empty 'content' child, and zero 'component' child. A composite component has at least one 'component' child, and possibly one 'content' child.

The types of the child elements are defined in detail in the next clauses.

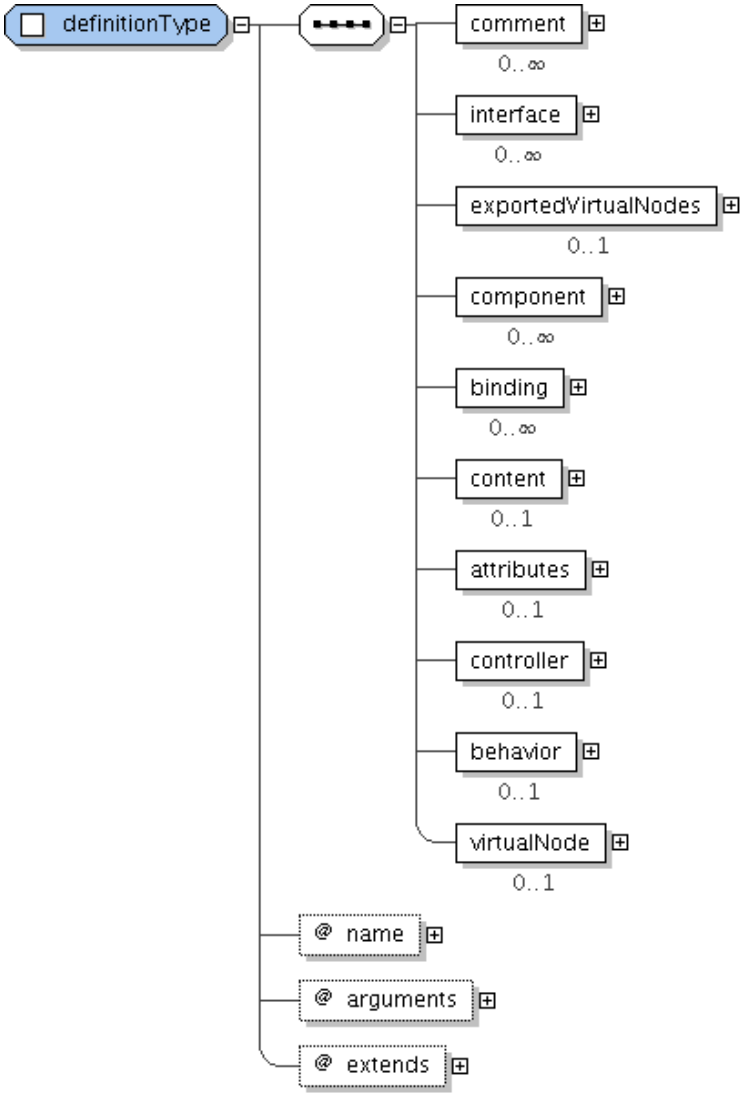


Figure 1

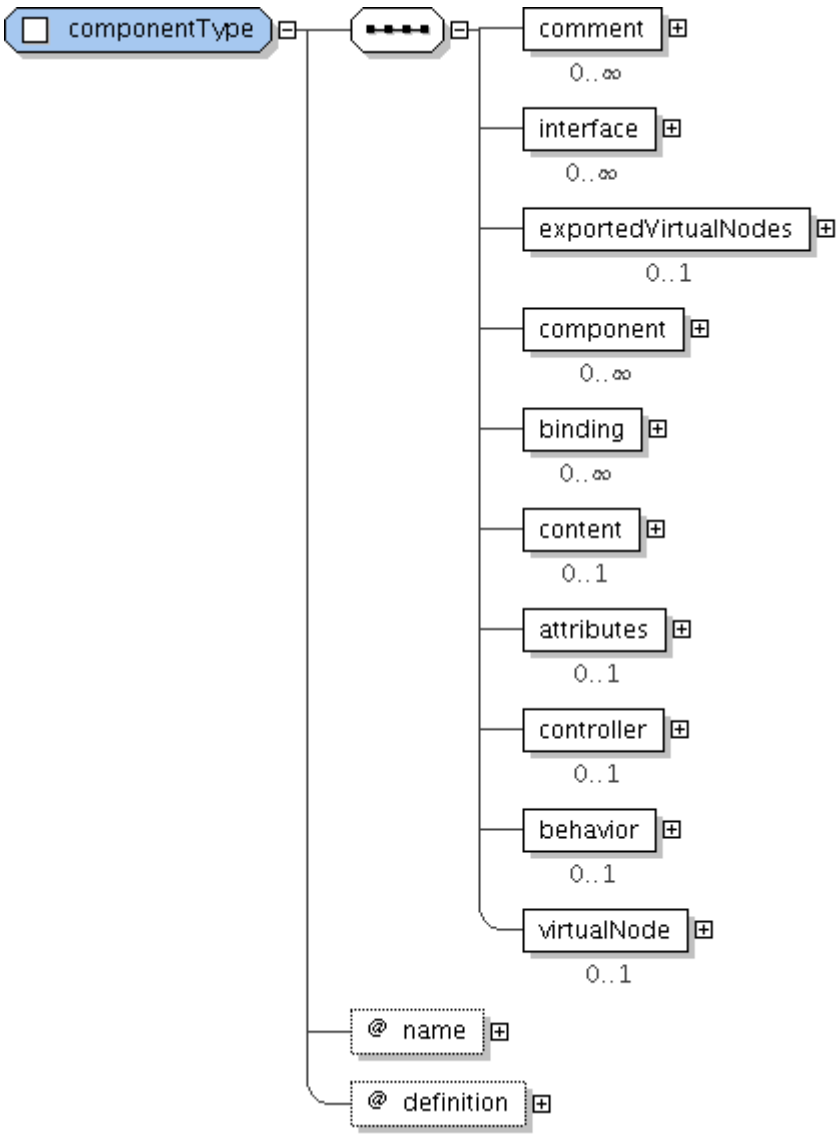


Figure 2

4.3 interface

Interfaces are the outlets which the components can be bound with. Components interfaces are of two kind: 'server', for interfaces which are offered by the component, and 'client', for interfaces which the component requires.

An interface element has the following attributes:

- **name:** (string) the name of the interface (required);
- **role:** ('client' or 'server') the role of the interface (required);
- **signature:** (string) the signature of the interface (optional);
- **contingency:** ('mandatory' or 'optional') (optional);
- **cardinality:** ('singleton', 'collection', 'gathercast', 'multicast') (optional);
- **comment:** a free form comment documenting the component (0-unlimited).

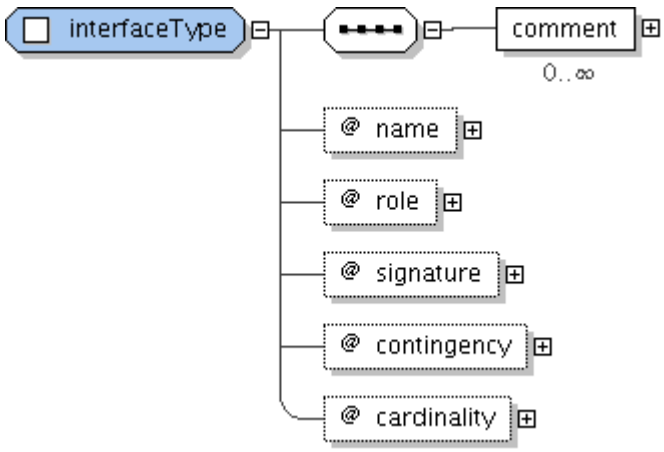


Figure 3

4.4 virtualNode

It is possible to specify among virtual nodes of a GCM deployment specification file, which of them should be used for the deployment of the component. This is done through a list of virtualNode elements.

A virtualNode element has no child element, and the following attributes:

- **name:** (string) the name of the virtual node (required);
- **cardinality:** ('single' or 'multiple') the cardinality of the virtual node. 'single' means the virtual node in the deployment descriptor should contain one node; 'multiple' means it should contain more than one node (required).

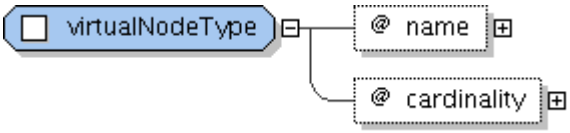


Figure 4

4.5 exportedVirtualNode

Virtual nodes can be exported and composed. Export and compose allow respectively to rename and merge virtual nodes. This extends reusability of existing components. When exported, a virtual node can take part in the composition of other exported virtual nodes.

An exportedVirtualNode element has the following attribute:

- **name:** (string) the name of the exported Virtual Node (required);
- **composedFrom:** (sequence of composingVirtualNode elements) (0-unlimited) This element contains a sequence of elements named composingVirtualNode.

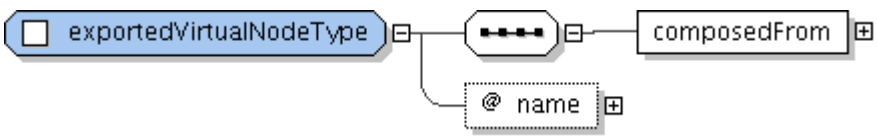


Figure 5

4.6 composingVirtualNode

A composingVirtualNode element has the following attributes:

- **component:** (string) the component which defines the composing virtual node (required);
- **name:** (string) the name of the composing virtual node (required).

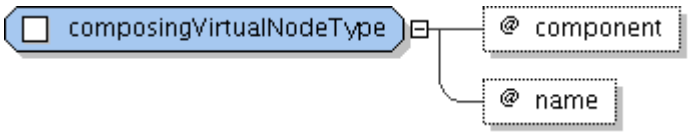


Figure 6

4.7 binding

Bindings define links between the client interface of a component to the server interface of another component.

A binding element may contain a sequence of elements of type comment. It also has the following attributes:

- **client:** (string) the name of the client interface from which the binding is made (required);
- **server:** (string) the name of the server interface to which the binding is made (required).

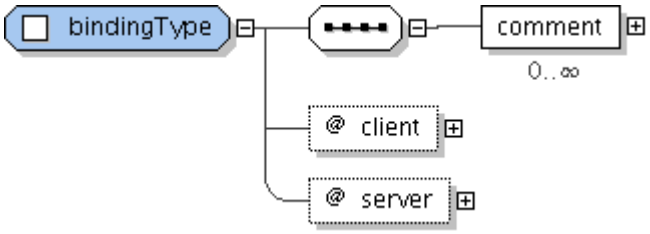


Figure 7

4.8 content

A content element may contain a sequence of elements of type comment. It also has the following attributes:

- **class:** (string) the class implementing the component (required).

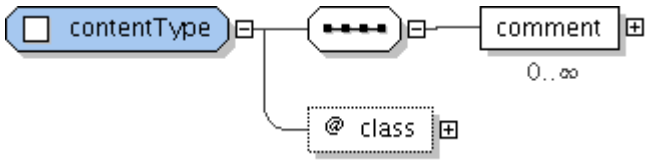


Figure 8

4.9 attributes

The attributes element has the following child elements:

- **comment:** (comment type) a comment (0-unlimited);
- **attribute:** (attribute type) an attribute (0-unlimited).

An attribute element may also contain a sequence of comments. It also has the following attributes:

- **name:** (string) the name of the attribute (required);
- **value:** (string) the value of the attribute (required).

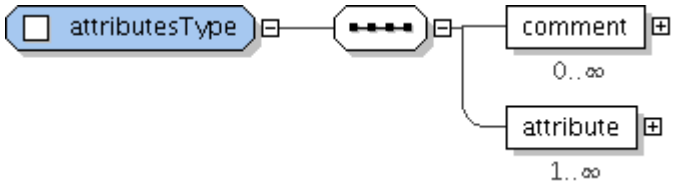


Figure 9

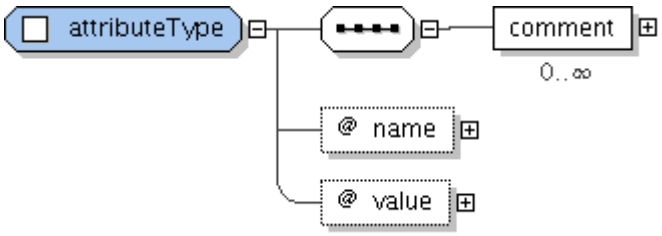


Figure 10

4.10 controller

A controller element may contain a sequence of comments. It also has the following attributes:

- **desc:** (string) the name of the component's controller (required).

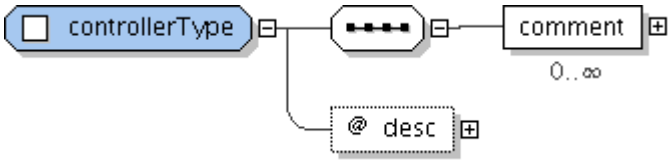


Figure 11

4.11 behaviour

The <behaviour> element cannot have any children. It can have the following attributes:

- **language:** (string): the specification language used by the behaviour file or value (required);
- **file:** (string) the path to the behaviour specification file;
- **value:** (string) the behaviour specification, specified inline.

File and **value** are mutually exclusive, only one should be specified.

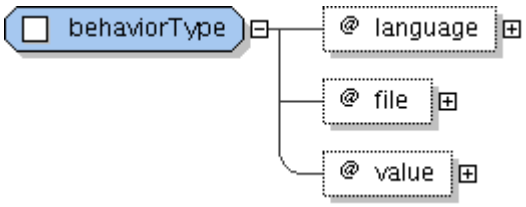


Figure 12

4.12 comment

A comment element has the following attributes:

- **language:** (string) the language of the comment's text (required)
- **text:** (string) the comment itself (required)

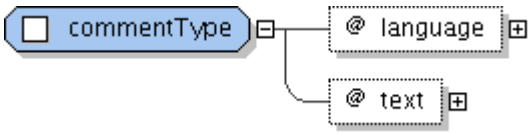


Figure 13

Annex A (normative): GCM ADL Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:gcm:adl:1.0"
xmlns="urn:gcm:adl:1.0" elementFormDefault="qualified" >

    <xsd:complexType name="interfaceType">
        <xsd:sequence>
            <xsd:element name="comment" type="commentType" minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="role" use="optional">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="client" />
                    <xsd:enumeration value="server" />
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="signature" type="xsd:string" use="optional" />
        <xsd:attribute name="contingency" use="optional">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="mandatory" />
                    <xsd:enumeration value="optional" />
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="cardinality">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="singleton" />
                    <xsd:enumeration value="collection" />
                    <xsd:enumeration value="multicast" />
                    <xsd:enumeration value="gathercast" />
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>

    <xsd:complexType name="componentType">
        <xsd:sequence>
            <xsd:element name="comment" type="commentType" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="interface" type="interfaceType" minOccurs="0" maxOccurs="unbounded"
/>
            <xsd:element name="exportedVirtualNodes" type="exportedVirtualNodesType" minOccurs="0"
maxOccurs="1" />
            <xsd:element name="component" type="componentType" minOccurs="0" maxOccurs="unbounded"
/>
            <xsd:element name="binding" type="bindingType" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="content" type="contentType" minOccurs="0" maxOccurs="1" />
            <xsd:element name="attributes" type="attributesType" minOccurs="0" maxOccurs="1" />
            <xsd:element name="controller" type="controllerType" minOccurs="0" maxOccurs="1" />
            <xsd:element name="behaviour" type="behaviorType" minOccurs="0" maxOccurs="1" />
            <xsd:element name="virtualNode" type="virtualNodeType" minOccurs="0" maxOccurs="1" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="definition" type="xsd:string" use="optional" />

    </xsd:complexType>

    <xsd:complexType name="bindingType">
        <xsd:sequence>
            <xsd:element name="comment" type="commentType" minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:attribute name="client" type="xsd:string" use="required" />
        <xsd:attribute name="server" type="xsd:string" use="required" />
    </xsd:complexType>

    <xsd:complexType name="contentType">
        <xsd:sequence>
            <xsd:element name="comment" type="commentType" minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
```

```
        <xsd:attribute name="class" type="xsd:string" use="required" />
    </xsd:complexType>

    <xsd:complexType name="attributeType">
        <xsd:sequence>
            <xsd:element name="comment" type="commentType" minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="value" type="xsd:string" use="required" />
    </xsd:complexType>

    <xsd:complexType name="attributesType">
        <xsd:sequence>
            <xsd:element name="comment" type="commentType" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="attribute" type="attributeType" minOccurs="1" maxOccurs="unbounded"
/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="controllerType">
        <xsd:sequence>
            <xsd:element name="comment" type="commentType" minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:attribute name="desc" type="xsd:string" use="required" />
    </xsd:complexType>

    <xsd:complexType name="behaviorType">
        <xsd:attribute name="language" type="xsd:string" use="required" />
        <xsd:attribute name="file" type="xsd:string" use="required" />
        <xsd:attribute name="value" type="xsd:string" use="required" />
    </xsd:complexType>


    <xsd:complexType name="virtualNodeType">
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="cardinality">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="single" />
                    <xsd:enumeration value="multiple" />
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>


    <xsd:complexType name="exportedVirtualNodesType">
        <xsd:sequence>
            <xsd:element name="exportedVirtualNode" type="exportedVirtualNodeType" minOccurs="0"
maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="exportedVirtualNodeType">
        <xsd:sequence>
            <xsd:element name="composedFrom" type="composedFromType" minOccurs="1" maxOccurs="1" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required" />
    </xsd:complexType>

    <xsd:complexType name="composedFromType">
        <xsd:sequence>
            <xsd:element name="composingVirtualNode" type="composingVirtualNodeType" minOccurs="1"
maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="composingVirtualNodeType">
        <xsd:attribute name="component" use="required" />
        <xsd:attribute name="name" use="required" />
    </xsd:complexType>

    <xsd:complexType name="commentType">
        <xsd:attribute name="language" type="xsd:string" use="required" />
        <xsd:attribute name="text" type="xsd:string" use="required" />
    </xsd:complexType>

    <xsd:complexType name="definitionType">
```

```
        <xsd:sequence>
            <xsd:element name="comment" type="commentType" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="interface" type="interfaceType" minOccurs="0" maxOccurs="unbounded"
/>
            <xsd:element name="exportedVirtualNodes" type="exportedVirtualNodesType" minOccurs="0"
maxOccurs="1" />
            <xsd:element name="component" type="componentType" minOccurs="0" maxOccurs="unbounded"
/>
            <xsd:element name="binding" type="bindingType" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="content" type="contentType" minOccurs="0" maxOccurs="1" />
            <xsd:element name="attributes" type="attributesType" minOccurs="0" maxOccurs="1" />
            <xsd:element name="controller" type="controllerType" minOccurs="0" maxOccurs="1" />
            <xsd:element name="behaviour" type="behaviorType" minOccurs="0" maxOccurs="1" />
            <xsd:element name="virtualNode" type="virtualNodeType" minOccurs="0" maxOccurs="1" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="arguments" type="xsd:string" use="optional" />
        <xsd:attribute name="extends" type="xsd:string" use="optional" />
    </xsd:complexType>

    <xsd:element name="definition" type="definitionType" />

</xsd:schema>
```

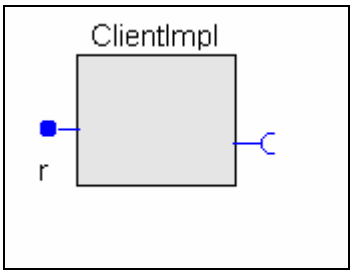
Annex B (informative): Examples of ADL files

We give here an example of ADL description for simple application, together with a graphical representation using VCE (when applicable). The VCE tool provides a graphical editor for component architecture that imports and exports ADL compliant with the present document. The graphical elements themselves are not normative.

B.1 Primitive components

A primitive component is defined as a set of required interfaces, a set of provided interfaces, and the class that implements those provided interfaces.

EXAMPLE:

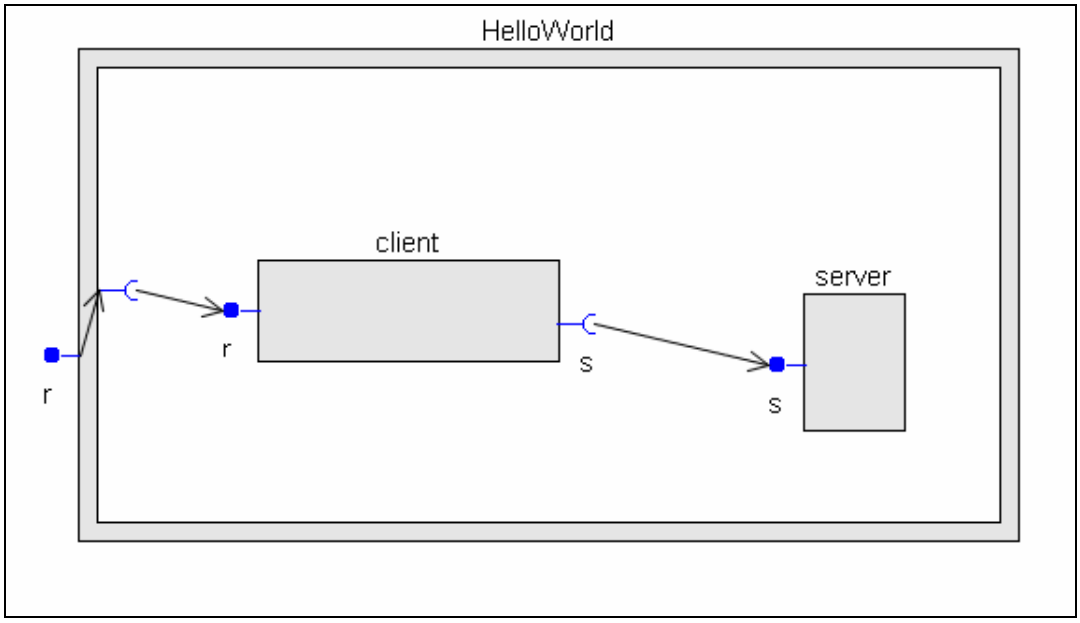



```
<definition name="ClientImpl">
   <interface name="r" role="server" signature="java.lang.Runnable"/>
   <interface name="s" role="client" signature="Service"/>
   <content class="ClientImpl"/>
    <controller desc="primitive"/>
 </definition>
```

B.2 Composite components

A composite components is defined as a set of required interfaces, a set of provided interfaces, and references to the sub-components it contains.

EXAMPLE:




```
<definition name="HelloWorld">
   <interface name="r" role="server" signature="java.lang.Runnable"/>
   <component name="client">
       <interface name="r" role="server" signature="java.lang.Runnable"/>
       <interface name="s" role="client" signature="Service"/>
       <content class="ClientImpl"/>
       <controller desc="primitive"/>
   </component>
   <component name="server">
       <interface name="s" role="server" signature="Service"/>
       <content class="ServerImpl"/>
       <controller desc="primitive"/>
   </component>
   <binding client="this.r" server="client.r"/>
   <binding client="client.s" server="server.s"/>
   <controller desc="composite"/>
 </definition>
```

B.3 Virtual nodes

In the following example, from component definitions "c1" and "c2", a virtual-node "client-node" is defined.

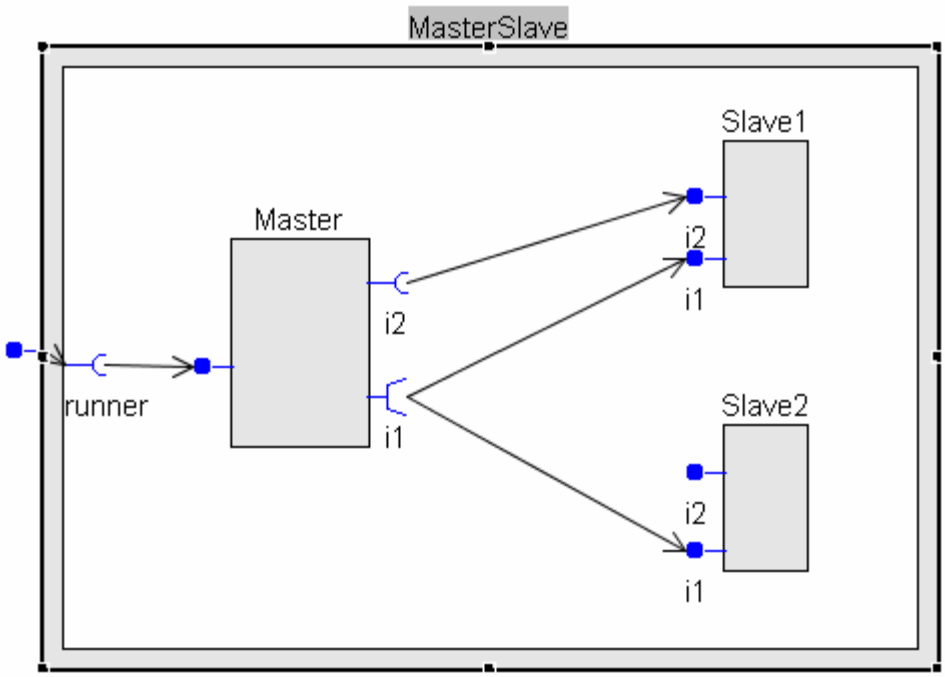
EXAMPLE:

```
<exportedVirtualNodes>
<exportedVirtualNode name="client-node">
<composingVirtualNode component="c1" name="client1" />
<composingVirtualNode component="c2" name="client2" />
<exportedVirtualNode>
</exportedVirtualNodes>
```

B.4 Component with multicast interfaces

Multicast interfaces provide abstractions for one-to-many communication. A single invocation on a multicast interface is transformed into a set of invocations. Multicast interfaces are defined in the ADL by using the 'multicast' cardinality in an interface definition.

The behaviour of a multicast interface (argument handling, scattering, broadcasting, etc. and return values collecting) will be defined at API level.




```
<definition name="org.objectweb.proactive.examples.components.userguide.adl.Composite">
  <interface signature="org.objectweb.proactive.examples.components.userguide.Runner" role="server"
name="runner"/>
  <component name="Master"
definition="org.objectweb.proactive.examples.components.userguide.adl.Master"/>
  <component name="Slave1"
definition="org.objectweb.proactive.examples.components.userguide.adl.Slave"/>
  <component name="Slave2"
definition="org.objectweb.proactive.examples.components.userguide.adl.Slave"/>
  <binding client="this.runner" server="Master.runner"/>
  <binding client="Master.i1" server="Slave1.i1"/>
  <binding client="Master.i1" server="Slave2.i1"/>
  <binding client="Master.i2" server="Slave1.i2"/>
<controller desc="composite"/>
</definition>

<definition name="org.objectweb.proactive.examples.components.userguide.adl.Slave">
  <interface signature="org.objectweb.proactive.examples.components.userguide.Itf1" role="server"
name="i1"/>
  <interface signature="org.objectweb.proactive.examples.components.userguide.Itf2" role="server"
name="i2"/>
  <content class="org.objectweb.proactive.examples.components.userguide.SlaveImpl"/>
<controller desc="primitive"/>
</definition>

<definition name="org.objectweb.proactive.examples.components.userguide.adl.Master">
  <interface signature="org.objectweb.proactive.examples.components.userguide.Runner" role="server"
name="runner"/>
  <interface signature="org.objectweb.proactive.examples.components.userguide.Itf1Multicast"
role="client" name="i1" cardinality="multicast"/>
  <interface signature="org.objectweb.proactive.examples.components.userguide.Itf2" role="client"
name="i2"/>
  <content class="org.objectweb.proactive.examples.components.userguide.MasterImpl"/>
<controller desc="primitive"/>
```

```
</definition>
```

B.5 Component with gathercast interfaces

Gathercast interfaces are the counterpart of multicast interface, they are abstractions for many-to-one communication. Their behaviour is symmetrical to the one of multicast interfaces. A gathercast interface coordinates incoming invocations before continuing the invocation flow, then return values are redistributed to the invoking components. Gathercast interfaces are defined in the ADL by using the 'gathercast' cardinality in an interface definition.

B.6 Behaviour

The Behaviour element is used to attach a behaviour specification to a component. The intended use is to check compatibility of component assemblies at a dynamic level (protocol compatibility), without knowing the implementation of the component (black-box specification).

The behaviour specification is to be written in an external file. Reading this file requires a specific parser. Example of behaviour languages currently used are: 'fc2', 'lotos', 'behaviour-protocol', 'fiacre'.

EXAMPLE:

```
<component name="client">
  <interface name="r" role="server" signature="java.lang.Runnable"/>
  <interface name="s" role="client" signature="Service"/>
 <behaviour language="lotos" file="client.lotos">
</component>
```

Annex C (informative): Bibliography

- "A formal specification of the Fractal component model".

NOTE: Available at http://hal.inria.fr/inria-00338987/.

- Francoise Baude, Denis Caromel, Ludovic Henrio and Matthieu Morel: "Collective Interfaces for Distributed Components" - proceedings of CCGrid 2007.

- CoreGrid Network of excellence: Deliverable D.PM.02: "Proposals for a Grid Component Model".

NOTE: Available at http://www.coregrid.net/mambo/content/view/428/292/.

- CoreGrid Network of excellence: Deliverable D.PM.07: "Innovative Features of GCM (with sample case studies): a Technical Survey".

- CoreGrid Network of excellence: Deliverable D.STE.05: "Design of the Integrated Toolkit with Supporting Mediator Components".

- Maciej Malawski, Marian Bubak, Francoise Baude, Denis Caromel, Ludovic Henrio, and Matthieu Morel: "Interoperability of grid component models: GCM and CCA case study" - Towards Next Generation Grids: Proceedings of the CoreGRID Symposium, August 2007. Springer.

- Antonio Cansado, Ludovic Henrio, Eric Madelaine: "Towards real case Model-checking: an extension to Fractal ADL", 5th Fractal Workshop, Nantes, 2006.

- The Syntax enad Semantics of FIACRE.

NOTE: Available at http://www.laas.fr/~bernard/fiacre/.

- GridComp project: http://gridcomp.ercim.org/ and GridComp use-cases: http://gridcomp.ercim.org/content/view/41/39/.

- ETSI Grid PlugTests

NOTE: Available at http://www.etsi.org/plugtests/GRID2008/GRID.htm.

History

Document history		
V1.1.1	March 2009	Publication