ProActive:
Distributed and Mobile Objects for the GRID

Denis Caromel, et al.
www.inria.fr/oasis/ProActive

OASIS Team
INRIA -- CNRS - I3S -- Univ. of Nice Sophia-Antipolis, IUF
Vers. Mai 12th 2003

- Remote Objects, Asynchronous Method Calls, Futures,
- Group Communications, Mobile Objects,
- Graphical Interface (IC2D), XML Deploiement,
- Interfaced with Globus, rsh, ssh, LSF, RMIregistry, Jini

Table of Contents

1. ProActive Basic Model, Features, Architecture, and Tools
- 1.1 Basic Model
  - 1.1.1 Active Objects, Asynchronous Calls, Futures, Sharing
  - 1.1.2 API for AO creation
  - 1.1.3 Polymorphism and Wait-by-necessity
  - 1.1.4 Intra-object synchronization
  - 1.1.5 Optimization: SharedOnRead
- 1.2 Collective Communications: Groups
- 1.3 Architecture: a simple MOP
- 1.4 Meta-Objects for Distribution
- 1.5 Abstract Deployment Model
- 1.6 IC2D: Interactive Control & Debug for Distribution
- 1.7 DEMO: IC2D with C3D : Collaborative 3D renderer in //

Table of Contents (2)

2. Mobility
- 2.1 Principles:
  Active Objects with: passive objects, pending requests and futures
- 2.2 API and Abstraction for mobility
- 2.3 Optimizations
- 2.4 Performance Evaluation of Mobile Agent
- 2.5 Automatic Continuations

ProActive:
A Java API + Tools for Parallel, Distributed Computing

- A uniform framework: **An Active Object pattern**
- A formal model behind: **Prop. Determinism, insensitivity to deploy.**

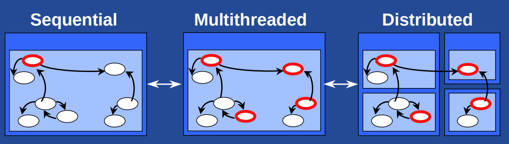
Main features:

- **Remotely accessible Objects (Classes, not only Interfaces, Dynamic)**
- **Asynchronous Communications with synchro: automatic Futures**
- **Group Communications, Migration (mobile computations)**
- **XML Deployment Descriptors**
- **Interfaced with various protocols:** `rsh,ssh,LSF,Globus,Jini,RMIregistry`
- **Visualization and monitoring: IC2D**

In the www. ObjectWeb .org Consortium (Open Source middleware)
since April 2002 (**LGPL license**)

ProActive PDC
Objectives and Rationale

Seamless

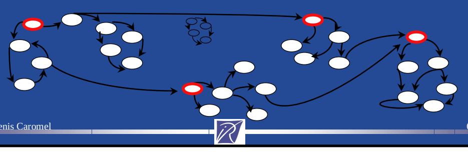


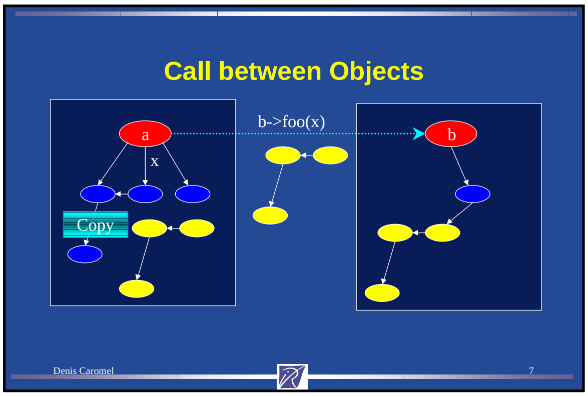
- Most of the time, activities and distribution are not known at the beginning, and change over time
- Seamless implies reuse, smooth and incremental transitions

Library !

ProActive : model

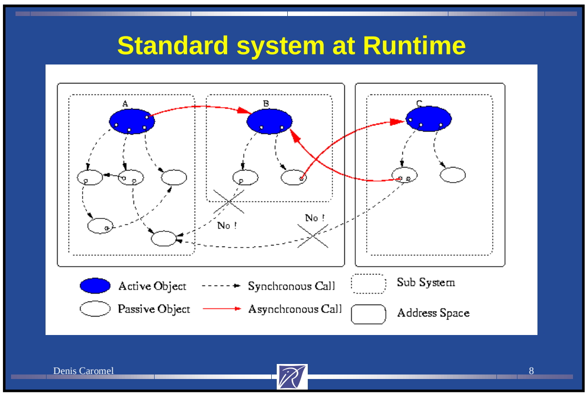
- Active objects : coarse-grained structuring entities (subsystems)
- Each active object: - possibly owns many passive objects
  - has exactly one thread.
- No shared passive objects -- Parameters are passed by deep-copy
- Asynchronous Communication between active objects
- Future objects and wait-by-necessity.
- Full control to serve incoming requests (reification)





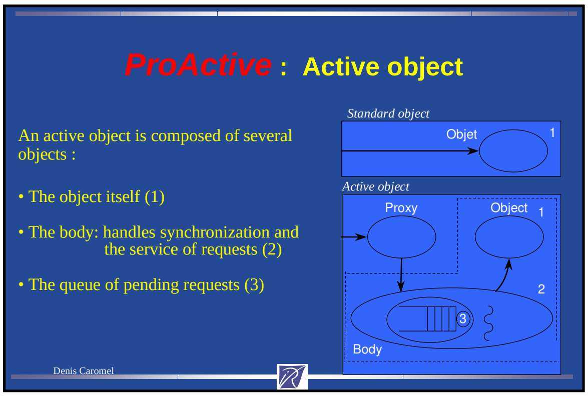
Call between Objects

b->foo(x)
a
x
Copy
b



Standard system at Runtime

A
B
C
No !
No !
Active Object
Passive Object
Synchronous Call
Asynchronous Call
Sub System
Address Space

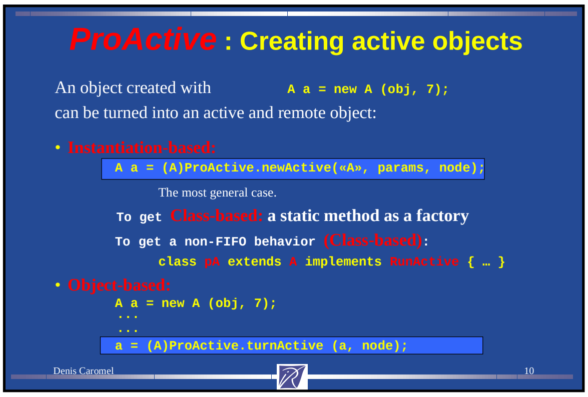


ProActive : Active object

An active object is composed of several objects :

- The object itself (1)
- The body: handles synchronization and the service of requests (2)
- The queue of pending requests (3)

Standard object
Objet
1
Active object
Proxy
Object
1
2
3
Body



ProActive : Creating active objects

An object created with `A a = new A (obj, 7);`
can be turned into an active and remote object:

- **Instantiation-based:**

```
A a = (A)ProActive.newActive(«A», params, node);
```

The most general case.

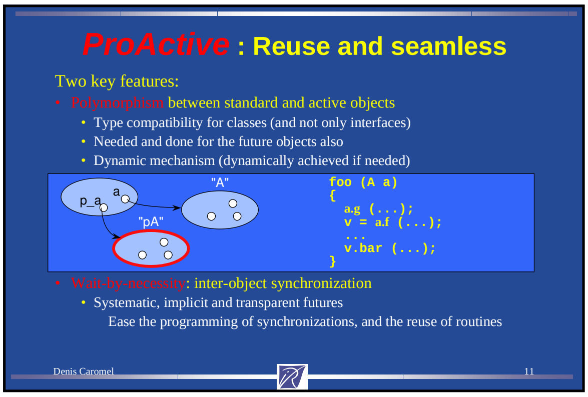
`To get` **Class-based: a static method as a factory**

`To get a non-FIFO behavior` **(Class-based):**

```
class pA extends A implements RunActive { … }
```

- **Object-based:**

```
A a = new A (obj, 7);
...
...
a = (A)ProActive.turnActive (a, node);
```



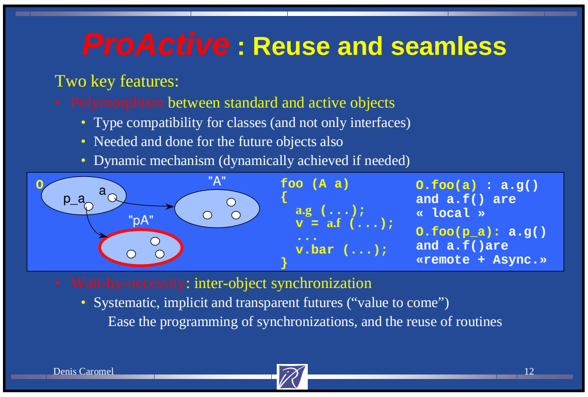
ProActive : Reuse and seamless

Two key features:

- Polymorphism between standard and active objects
  - Type compatibility for classes (and not only interfaces)
  - Needed and done for the future objects also
  - Dynamic mechanism (dynamically achieved if needed)

```
foo (A a)
{
  a.g (...);
  v = a.f (...);
  ...
  v.bar (...);
}
```

- Wait-by-necessity: inter-object synchronization
  - Systematic, implicit and transparent futures
    Ease the programming of synchronizations, and the reuse of routines



ProActive : Reuse and seamless

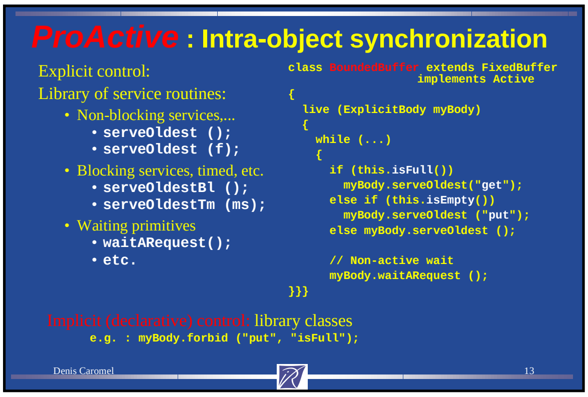
Two key features:

- Polymorphism between standard and active objects
  - Type compatibility for classes (and not only interfaces)
  - Needed and done for the future objects also
  - Dynamic mechanism (dynamically achieved if needed)

```
foo (A a)
{
  a.g (...);
  v = a.f (...);
  ...
  v.bar (...);
}
```

```
O.foo(a) : a.g()
and a.f() are
« local »
O.foo(p_a): a.g()
and a.f()are
«remote + Async.»
```

- Wait-by-necessity: inter-object synchronization
  - Systematic, implicit and transparent futures (“value to come”)
    Ease the programming of synchronizations, and the reuse of routines



ProActive : Intra-object synchronization

Explicit control:

Library of service routines:

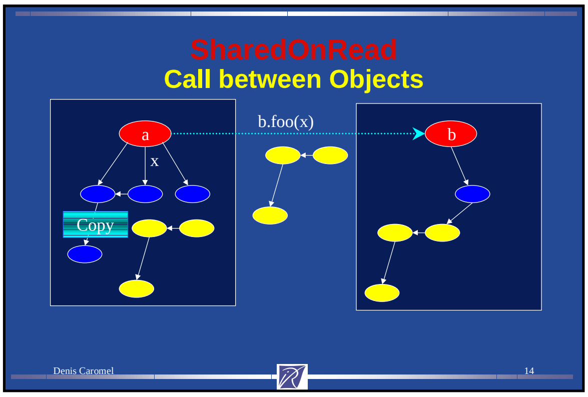
- Non-blocking services,...
  - **`serveOldest ();`**
  - **`serveOldest (f);`**
- Blocking services, timed, etc.
  - **`serveOldestBl ();`**
  - **`serveOldestTm (ms);`**
- Waiting primitives
  - **`waitARequest();`**
  - **`etc.`**

```
class BoundedBuffer extends FixedBuffer
                    implements Active
{
  live (ExplicitBody myBody)
  {
    while (...)
    {
      if (this.isFull())
        myBody.serveOldest("get");
      else if (this.isEmpty())
        myBody.serveOldest ("put");
      else myBody.serveOldest ();

      // Non-active wait
      myBody.waitARequest ();
}}}
```

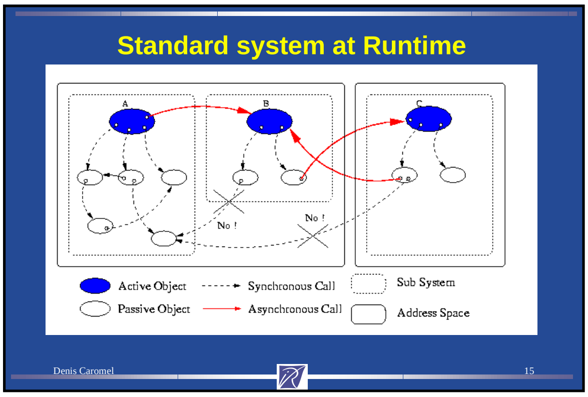
Implicit (declarative) control: library classes

`e.g. : myBody.forbid ("put", "isFull");`



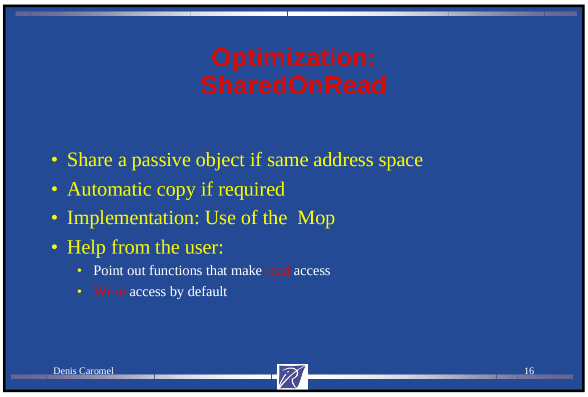
SharedOnRead
Call between Objects

a
x
b.foo(x)
b
Copy



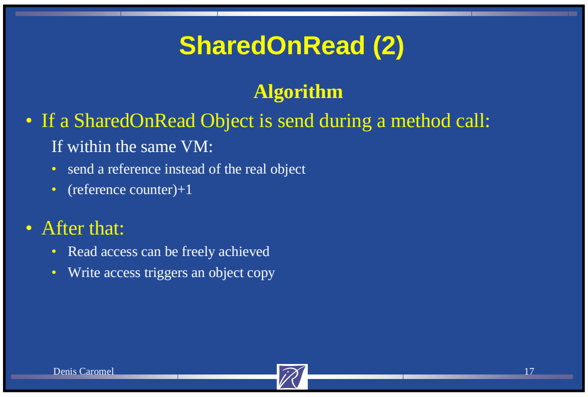
Standard system at Runtime

A
B
C
No !
No !
Active Object
Passive Object
Synchronous Call
Asynchronous Call
Sub System
Address Space



Optimization:
SharedOnRead

- Share a passive object if same address space
- Automatic copy if required
- Implementation: Use of the Mop
- Help from the user:
  - Point out functions that make read access
  - Write access by default

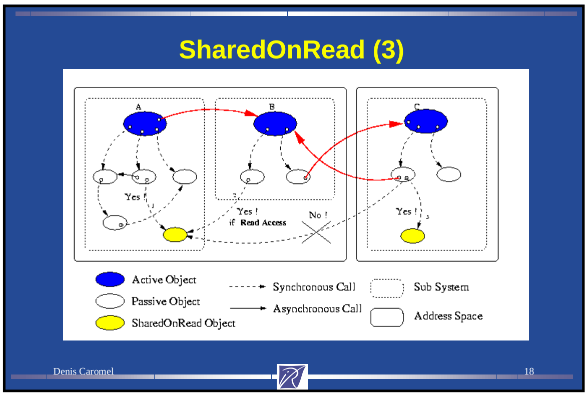


SharedOnRead (2)

Algorithm

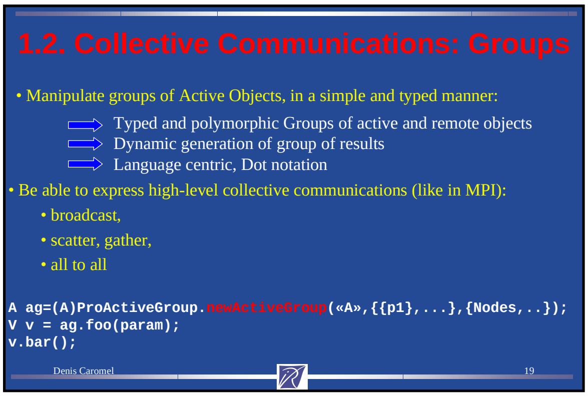
- If a SharedOnRead Object is send during a method call:

  If within the same VM:
  - send a reference instead of the real object
  - (reference counter)+1
- After that:
  - Read access can be freely achieved
  - Write access triggers an object copy



SharedOnRead (3)

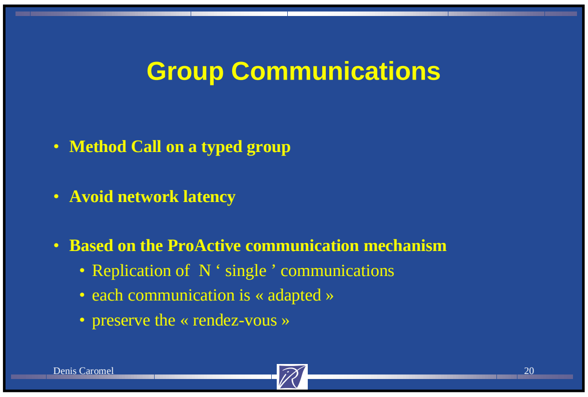
A
B
C
Yes !
Yes !
if Read Access
No !
Yes !
Active Object
Passive Object
SharedOnRead Object
Synchronous Call
Asynchronous Call
Sub System
Address Space



1.2. Collective Communications: Groups

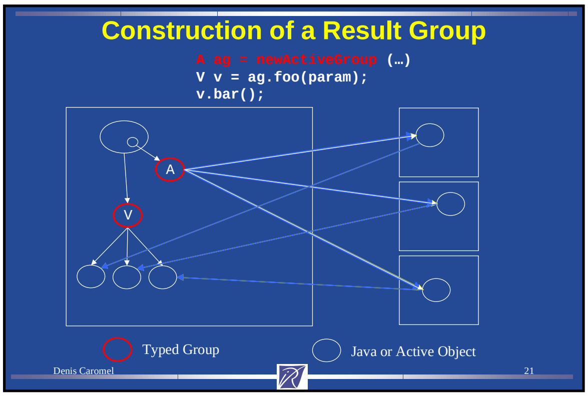
- Manipulate groups of Active Objects, in a simple and typed manner:
  - ➡ Typed and polymorphic Groups of active and remote objects
  - ➡ Dynamic generation of group of results
  - ➡ Language centric, Dot notation
- Be able to express high-level collective communications (like in MPI):
  - broadcast,
  - scatter, gather,
  - all to all

```
A ag=(A)ProActiveGroup.newActiveGroup(«A»,{{p1},...},{Nodes,..});
V v = ag.foo(param);
v.bar();
```



Group Communications

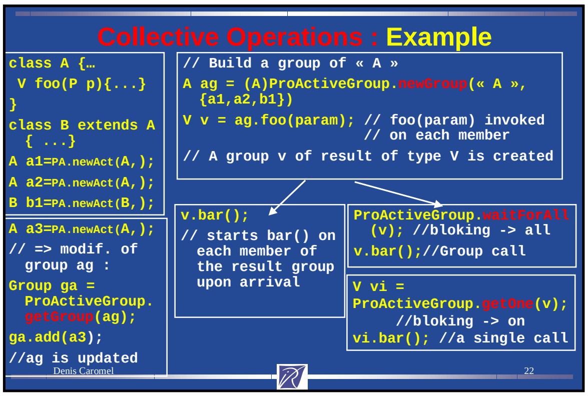
- **Method Call on a typed group**
- **Avoid network latency**
- **Based on the ProActive communication mechanism**
  - Replication of N ‘ single ’ communications
  - each communication is « adapted »
  - preserve the « rendez-vous »



Construction of a Result Group

```
A ag = newActiveGroup (…)
V v = ag.foo(param);
v.bar();
```

A
V
Typed Group
Java or Active Object



Collective Operations : Example

```
class A {…
 V foo(P p){...}
}
class B extends A
  { ...}
A a1=PA.newAct(A,);
A a2=PA.newAct(A,);
B b1=PA.newAct(B,);
```

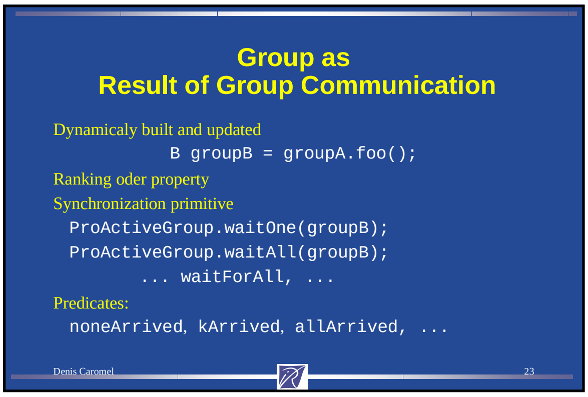
```
A a3=PA.newAct(A,);
// => modif. of
  group ag :
Group ga =
  ProActiveGroup.
  getGroup(ag);
ga.add(a3);
//ag is updated
```

```
// Build a group of « A »
A ag = (A)ProActiveGroup.newGroup(« A »,
  {a1,a2,b1})
V v = ag.foo(param); // foo(param) invoked
                     // on each member
// A group v of result of type V is created
```

```
v.bar();
// starts bar() on
  each member of
  the result group
  upon arrival
```

```
ProActiveGroup.waitForAll
  (v); //bloking -> all
v.bar();//Group call
```

```
V vi =
ProActiveGroup.getOne(v);
     //bloking -> on
vi.bar(); //a single call
```



Group as
Result of Group Communication

Dynamicaly built and updated

`B groupB = groupA.foo();`

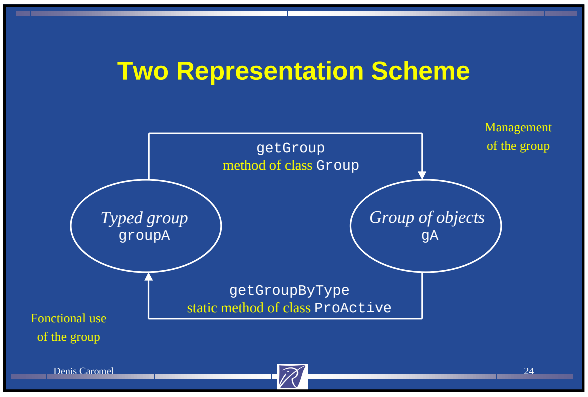
Ranking oder property

Synchronization primitive

```
ProActiveGroup.waitOne(groupB);
ProActiveGroup.waitAll(groupB);
        ... waitForAll, ...
```

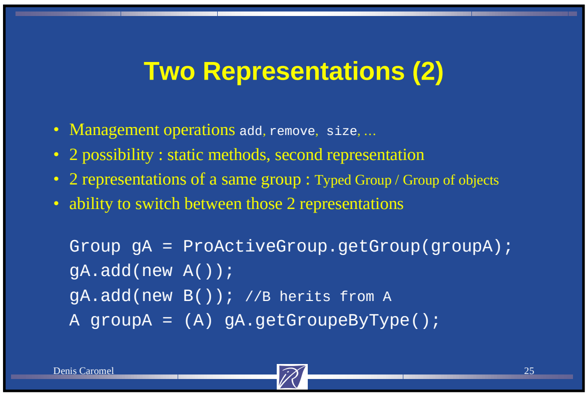
Predicates:

`noneArrived, kArrived, allArrived, ...`



Two Representation Scheme

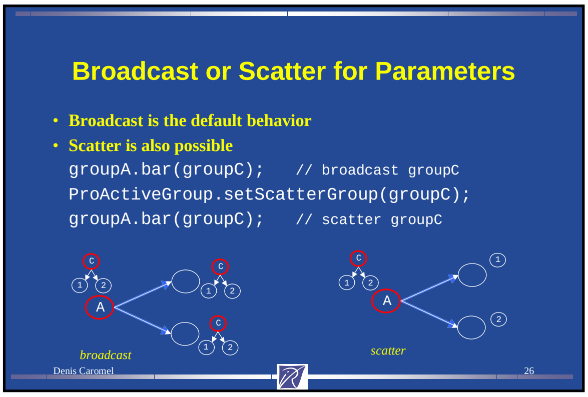
getGroup
method of class Group
Management
of the group
Typed group
groupA
Group of objects
gA
getGroupByType
static method of class ProActive
Fonctional use
of the group



Two Representations (2)

- Management operations `add`, `remove`, `size`, ...
- 2 possibility : static methods, second representation
- 2 representations of a same group : Typed Group / Group of objects
- ability to switch between those 2 representations

```
Group gA = ProActiveGroup.getGroup(groupA);
gA.add(new A());
gA.add(new B()); //B herits from A
A groupA = (A) gA.getGroupeByType();
```

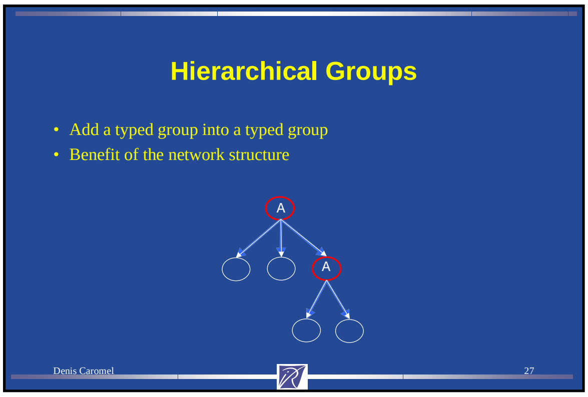


Broadcast or Scatter for Parameters

- **Broadcast is the default behavior**
- **Scatter is also possible**

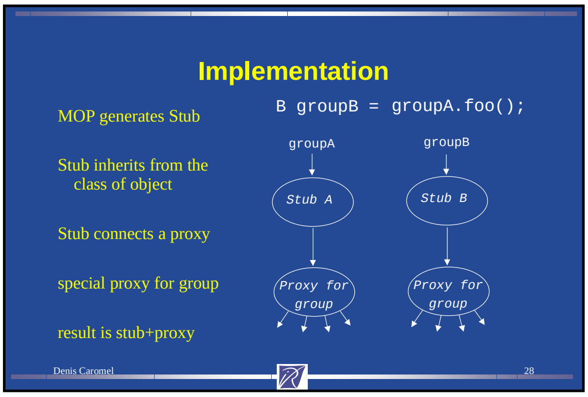
```
groupA.bar(groupC);   // broadcast groupC
ProActiveGroup.setScatterGroup(groupC);
groupA.bar(groupC);   // scatter groupC
```

C
1
2
A
C
1
2
C
1
2
broadcast
C
1
2
A
1
2
scatter



Hierarchical Groups

- Add a typed group into a typed group
- Benefit of the network structure



Implementation

`B groupB = groupA.foo();`

MOP generates Stub

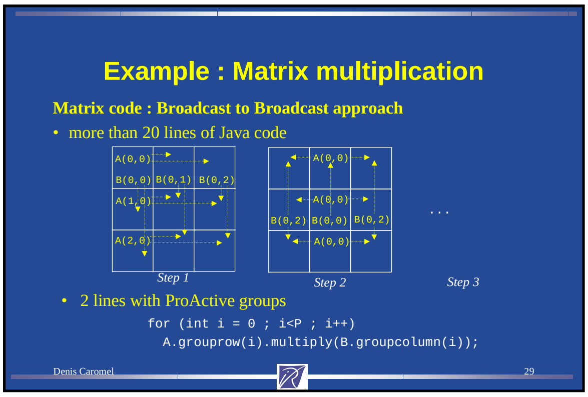
Stub inherits from the class of object

Stub connects a proxy

special proxy for group

result is stub+proxy

groupA
groupB
Stub A
Stub B
Proxy for group
Proxy for group



Example : Matrix multiplication

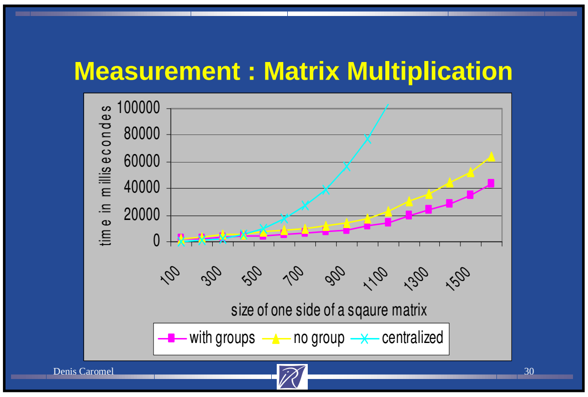
Matrix code : Broadcast to Broadcast approach

- more than 20 lines of Java code

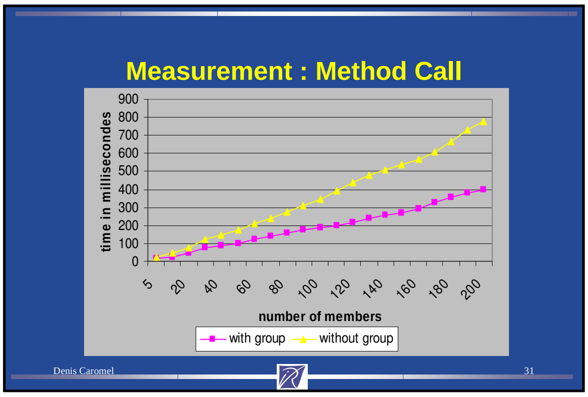
A(0,0)
B(0,0)
B(0,1)
B(0,2)
A(1,0)
A(2,0)
Step 1
A(0,0)
A(0,0)
B(0,2)
B(0,0)
B(0,2)
A(0,0)
Step 2
. . .
Step 3

- 2 lines with ProActive groups

```
for (int i = 0 ; i<P ; i++)
  A.grouprow(i).multiply(B.groupcolumn(i));
```



Measurement : Matrix Multiplication



Measurement : Method Call

Other features for Groups

Optimization

- Parallel calls within a group (latency hiding)
- Treatment in the result order (if needed)
- Scatter (a group as a parameter to be dispatched in Group. Com.)
- A single serialization of parameters

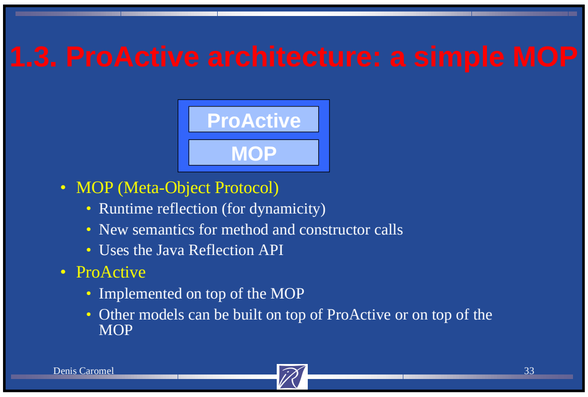
Conceptuel : Active Group

- A group becomes remotely accessible so: updatable and consistent

  ---> migration

  ---> Dynamic changes

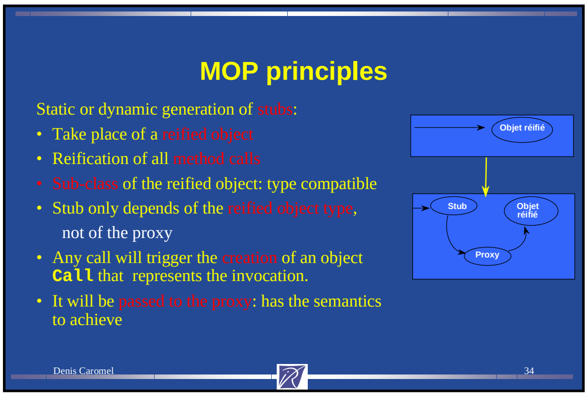
Perspective: using network multicast



1.3. ProActive architecture: a simple MOP

ProActive
MOP

- MOP (Meta-Object Protocol)
  - Runtime reflection (for dynamicity)
  - New semantics for method and constructor calls
  - Uses the Java Reflection API
- ProActive
  - Implemented on top of the MOP
  - Other models can be built on top of ProActive or on top of the MOP



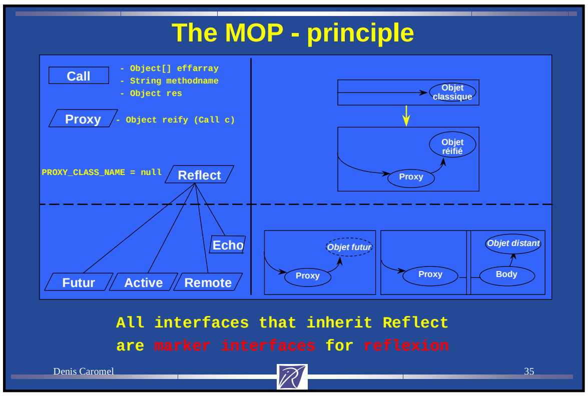
MOP principles

Static or dynamic generation of stubs:

- Take place of a reified object
- Reification of all method calls
- Sub-class of the reified object: type compatible
- Stub only depends of the reified object type,

  not of the proxy
- Any call will trigger the creation of an object **`Call`** that represents the invocation.
- It will be passed to the proxy: has the semantics to achieve

Objet réifié
Stub
Objet réifié
Proxy



The MOP - principle

Call
- Object[] effarray
- String methodname
- Object res
Proxy
- Object reify (Call c)
PROXY_CLASS_NAME = null
Reflect
Echo
Futur
Active
Remote
Objet classique
Objet réifié
Proxy
Objet futur
Proxy
Proxy
Objet distant
Body

`All interfaces that inherit Reflect`
`are marker interfaces for reflexion`

User Interface of the MOP

Instantiation of reified objects with static method newInstance of the MOP class

- Programming *class par class* with interfaces deriving from **`Reflect`**

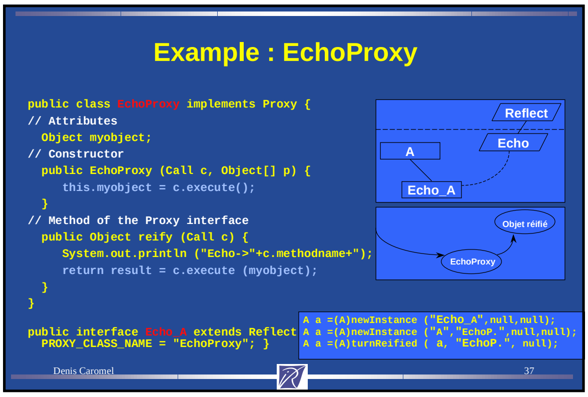
```
Vector v = (Vector) MOP.newInstance ( <name of reified class (impl. Reflect)>,
                                      <parameters passed to proxy>,
                                      < parameters of reified object constructor > );
```

- Or *instance per instance* :

```
Vector v = (Vector) MOP.newInstance ( <name of class standard>,
                                      <class proxy name to use>,
                                      <parameters given to proxy>,
                                      <parameters of reified object constructor > );
```

- Or *object per object* :

```
Vector v = (Vector) MOP.turnReified  ( <objet standard>,
                                       <class proxy name to use>,
                                       <parameters given to proxy> );
```



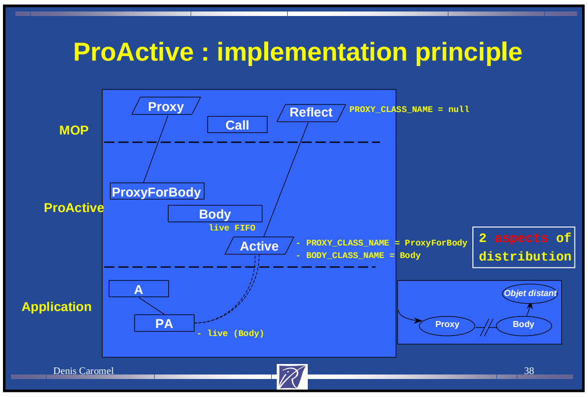
Example : EchoProxy

```
public class EchoProxy implements Proxy {
// Attributes
  Object myobject;
// Constructor
  public EchoProxy (Call c, Object[] p) {
     this.myobject = c.execute();
  }
// Method of the Proxy interface
  public Object reify (Call c) {
     System.out.println ("Echo->"+c.methodname+");
     return result = c.execute (myobject);
  }
}

public interface Echo_A extends Reflect
  PROXY_CLASS_NAME = "EchoProxy"; }
```

```
A a =(A)newInstance ("Echo_A",null,null);
A a =(A)newInstance ("A","EchoP.",null,null);
A a =(A)turnReified ( a, "EchoP.", null);
```

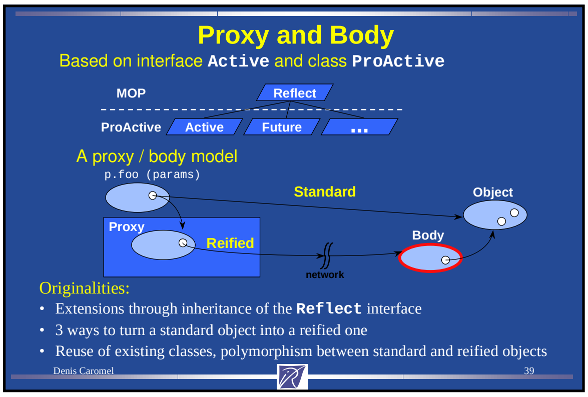
Reflect
Echo
A
Echo_A
Objet réifié
EchoProxy



ProActive : implementation principle

MOP
ProActive
Application
Proxy
Call
Reflect
PROXY_CLASS_NAME = null
ProxyForBody
Body
live FIFO
Active
- PROXY_CLASS_NAME = ProxyForBody
- BODY_CLASS_NAME = Body
A
PA
- live (Body)
Objet distant
Proxy
Body

2 aspects of distribution



Proxy and Body

Based on interface **`Active`** and class **`ProActive`**

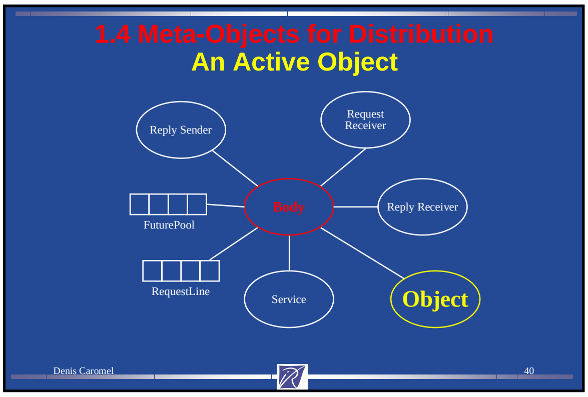
MOP
Reflect
ProActive
Active
Future
...

A proxy / body model

p.foo (params)
Standard
Object
Proxy
Reified
Body
network

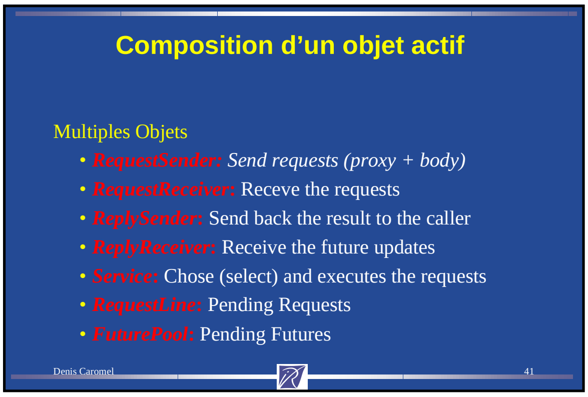
Originalities:

- Extensions through inheritance of the **`Reflect`** interface
- 3 ways to turn a standard object into a reified one
- Reuse of existing classes, polymorphism between standard and reified objects



1.4 Meta-Objects for Distribution
An Active Object

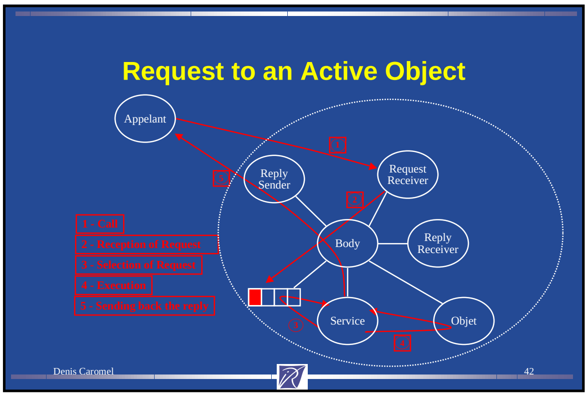
Reply Sender
Request Receiver
FuturePool
Body
Reply Receiver
RequestLine
Service
Object



Composition d’un objet actif

Multiples Objets

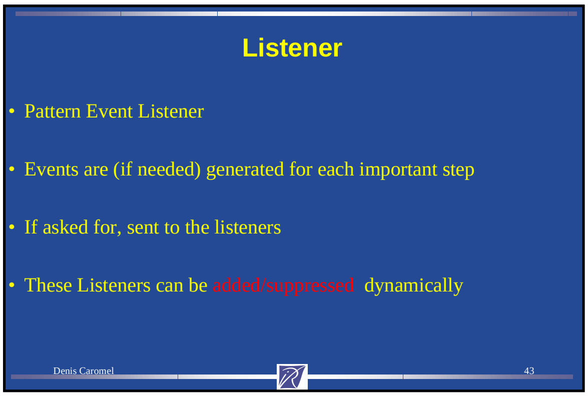
- *RequestSender: Send requests (proxy + body)*
- *RequestReceiver:* Receve the requests
- *ReplySender:* Send back the result to the caller
- *ReplyReceiver:* Receive the future updates
- *Service:* Chose (select) and executes the requests
- *RequestLine:* Pending Requests
- *FuturePool:* Pending Futures



Request to an Active Object

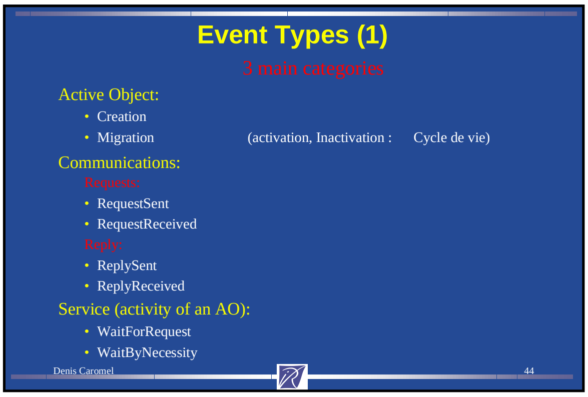
Appelant
1
Reply Sender
Request Receiver
5
2
Body
Reply Receiver
Service
Objet
3
4

1 - Call
2 - Reception of Request
3 - Selection of Request
4 - Execution
5 - Sending back the reply



Listener

- Pattern Event Listener
- Events are (if needed) generated for each important step
- If asked for, sent to the listeners
- These Listeners can be added/suppressed dynamically



Event Types (1)

3 main categories

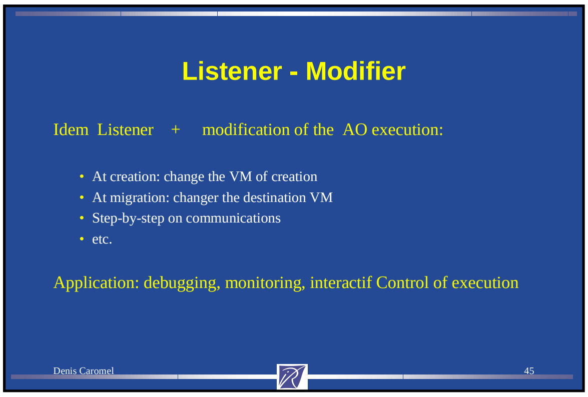
Active Object:
- Creation
- Migration (activation, Inactivation : Cycle de vie)

Communications:

Requests:
- RequestSent
- RequestReceived

Reply:
- ReplySent
- ReplyReceived

Service (activity of an AO):
- WaitForRequest
- WaitByNecessity

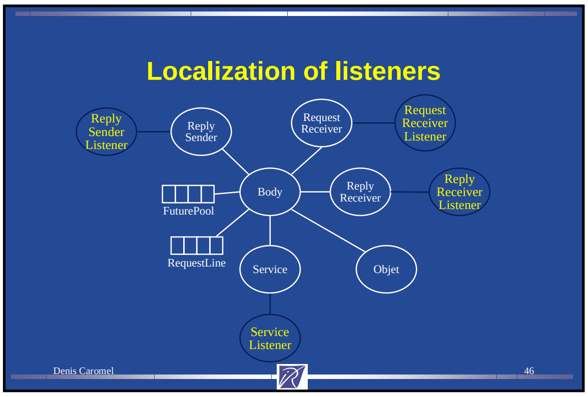


Listener - Modifier

Idem Listener + modification of the AO execution:

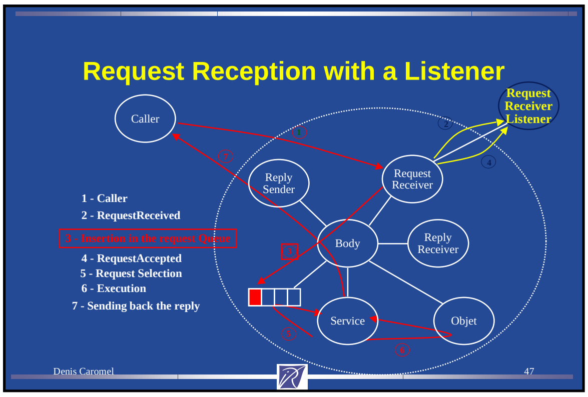
- At creation: change the VM of creation
- At migration: changer the destination VM
- Step-by-step on communications
- etc.

Application: debugging, monitoring, interactif Control of execution



Localization of listeners

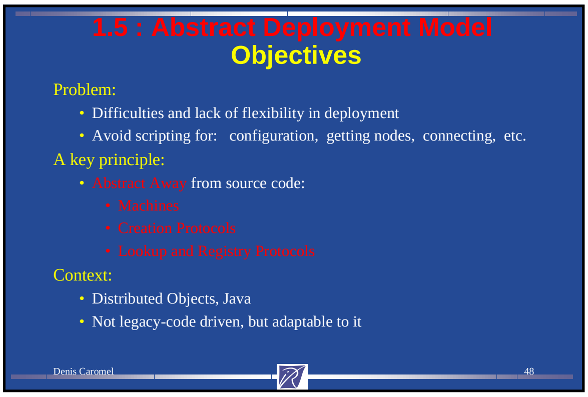
Reply Sender Listener
Reply Sender
Request Receiver
Request Receiver Listener
Body
Reply Receiver
Reply Receiver Listener
FuturePool
RequestLine
Service
Objet
Service Listener



Request Reception with a Listener

Caller
Request Receiver Listener
Reply Sender
Request Receiver
Body
Reply Receiver
Service
Objet
1
2
3
4
5
6
7

1 - Caller
2 - RequestReceived
3 - Insertion in the request Queue
4 - RequestAccepted
5 - Request Selection
6 - Execution
7 - Sending back the reply



1.5 : Abstract Deployment Model
Objectives

Problem:
- Difficulties and lack of flexibility in deployment
- Avoid scripting for: configuration, getting nodes, connecting, etc.

A key principle:
- Abstract Away from source code:
  - Machines
  - Creation Protocols
  - Lookup and Registry Protocols

Context:
- Distributed Objects, Java
- Not legacy-code driven, but adaptable to it

Descriptors: based on Virtual Nodes

Virtual Node (VN):

- Identified as a string name
- Used in program source
- Configured (mapped) in an XML descriptor file --> Nodes

Operations specified in descriptors:

- Mapping of VN to JVMs (leads to Node in a JVM on Host)
- Register or Lookup VNs
- Create or Acquire JVMs

```
      Program Source                 Descriptor   (RunTime)
|------------------------------| |----------------------------------------|
Activities (AO)   -->   VN              VN   -->   JVMs    -->   Hosts
```

Runtime structured entities: 1 VN --> n Nodes in n JVMs

Descriptors: Mapping Virtual Nodes

Example of an XML file descriptor:

```
Component Dependencies:
   Provides: …  Uses: ...
VirtualNodes:
   Dispatcher  <RegisterIn RMIregistry, Globus, Grid Service, … >
   RendererSet
Mapping:
   Dispatcher --> DispatcherJVM
   RendererSet --> JVMset
JVMs:
   DispatcherJVM = Current  // (the current JVM)
   JVMset=//ClusterSophia.inria.fr/ <Protocol GlobusGram … 10 >
...
```

Descriptors: Virtual Nodes in Programs

```
Descriptor pad = ProActive.getDescriptor ("file:.ProActiveDescriptor.xml");
VirtualNode vn = pad.activateMapping ("Dispatcher");  // Triggers the JVMs
Node node = vn.getNode();
...
C3D c3d = ProActive.newActive("C3D", param, node);
              log (  ... "created at:  " + node.name() + node.JVM() + node.host() );
```

Descriptors: Virtual Nodes in Programs

```
Descriptor pad = ProActive.getDescriptor ("file:.ProActiveDescriptor.xml");
VirtualNode vn = pad.activateMapping ("Dispatcher");  // Triggers the JVMs
Node node = vn.getNode();
...
C3D c3d = ProActive.newActive("C3D", param, node);
              log (  ... "created at:  " + node.name() + node.JVM() + node.host() );

                    // Cyclic mapping: set of nodes
VirtualNode vn = pad.activateMapping ("RendererSet");
while  (  … vn.getNbNodes … ) {
    Node node = vn.getNode();
    Renderer  re = ProActive.newActive(”Renderer", param, node);
```

1.6 IC2D

Interactive Control & Debug for Distribution

Features:

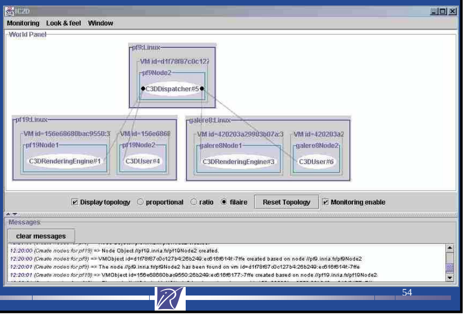
- Graphical visualization
- Textual visualization
- Monitoring and Control

IC2D: Interactive Control and Debugging of Distribution

Main Features:

- Hosts, JVM,
- Nodes
- Active Objects
- Topology
- Migration

- Logical Clock


IC2D
Monitoring
Look & feel
Window
World Panel
pf9:Linux
VM id=d1f78f87c0c127
pf9Node2
C3DDispatcher#5
pf19:Linux
VM id=156e68680bac9550:3
VM id=156e6868
pf19Node1
pf19Node2
C3DRenderingEngine#1
C3DUser#4
galere8:Linux
VM id=420203a29983b07a:3
VM id=420203a2
galere8Node1
galere8Node2
C3DRenderingEngine#3
C3DUser#6
Reset Topology
Messages
clear messages

☑ Display topology ○ proportional ○ ratio ● filaire ☑ Monitoring enable

```
12:20:00 (Create nodes for pf19) => Node Object //pf19.inria.fr/pf19Node2 created.
12:20:00 (Create nodes for pf9) => VMObject id=d1f78f87c0c127b4:26b249:ec616f614f:-7ffe created based on node //pf9.inria.fr/pf9Node2
12:20:01 (Create nodes for pf9) => The node //pf9.inria.fr/pf9Node2 has been found on vm id=d1f78f87c0c127b4:26b249:ec616f614f:-7ffe
12:20:01 (Create nodes for pf19) => VMObject id=156e68680bac9550:26b249:ec616f6177:-7ffe created based on node //pf19.inria.fr/pf19Node2
```

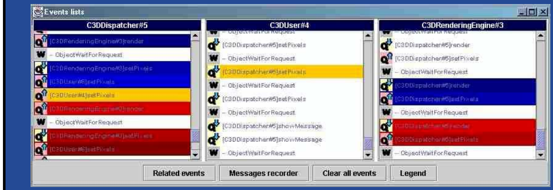
IC2D: Basic features

Graphical Visualisation:
- Hosts, Java Virtual Machines, Nodes, Active Objects
- Topology: reference and communications
- Status of active objects (executing, waiting, etc.)
- Migration of activities

Textual Visualisation:
- Ordered list of messages
- Status: waiting for a request or for a data
- Causal dependencies between messages
- Related events (corresponding send and receive, etc.)

Control and Monitoring:
- Drag and Drop migration of executing tasks
- Creation of additional JVMs and nodes

IC2D: Related Events


Events lists
C3DDispatcher#5
C3DUser#4
C3DRenderingEngine#3
Related events
Messages recorder
Clear all events
Legend

C3DDispatcher#5:

```
[C3DRenderingEngine#3]render
– ObjectWaitForRequest
[C3DRenderingEngine#3]setPixels
[C3DUser#5]setPixels
[C3DUser#4]setPixels
[C3DRenderingEngine#3]render
– ObjectWaitForRequest
[C3DRenderingEngine#3]setPixels
[C3DUser#5]setPixels
```

C3DUser#4:

```
[C3DDispatcher#5]setPixels
– ObjectWaitForRequest
[C3DDispatcher#5]setPixels
– ObjectWaitForRequest
[C3DDispatcher#5]setPixels
– ObjectWaitForRequest
[C3DDispatcher#5]showMessage
[C3DDispatcher#5]showMessage
– ObjectWaitForRequest
```

C3DRenderingEngine#3:

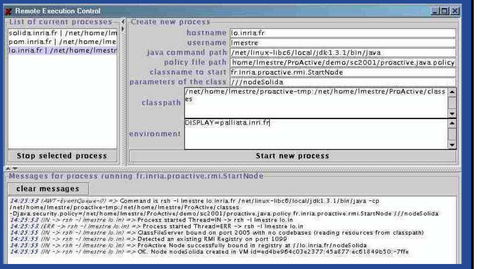
```
[C3DDispatcher#5]render
[C3DDispatcher#5]setPixels
– ObjectWaitForRequest
[C3DDispatcher#5]render
[C3DDispatcher#5]setPixels
– ObjectWaitForRequest
[C3DDispatcher#5]render
[C3DDispatcher#5]setPixels
– ObjectWaitForRequest
```

Events:
- Textual and ordered list of events for each Active Object
- Logical clock: related events, ==> Gives a Partial Order

IC2D: Dynamic change of Deployment New JVMs

Creation,
Acquisition
of
new JVMs,
and Nodes

Protocols:
rsh, ssh
Globus,
LSF


Remote Execution Control
List of current processes
Create new process
Stop selected process
Start new process
Messages for process running fr.inria.proactive.rmi.StartNode
clear messages

List of current processes:

```
solida.inria.fr | /net/home/lm
pom.inria.fr | /net/home/lme
lo.inria.fr | /net/home/lmestr
```

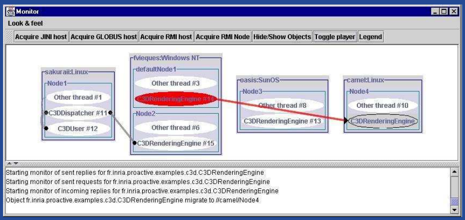
Create new process:

hostname: lo.inria.fr
username: lmestre
java command path: /net/linux-libc6/local/jdk1.3.1/bin/java
policy file path: home/lmestre/ProActive/demo/sc2001/proactive.java.policy
classname to start: fr.inria.proactive.rmi.StartNode
parameters of the class: ///nodeSolida
classpath: /net/home/lmestre/proactive-tmp:/net/home/lmestre/ProActive/classes
environment: DISPLAY=palliata.inri.fr

```
14:25:53 (AWT-EventQueue-0) => Command is rsh -l lmestre lo.inria.fr /net/linux-libc6/local/jdk1.3.1/bin/java -cp
/net/home/lmestre/proactive-tmp:/net/home/lmestre/ProActive/classes
-Djava.security.policy=/net/home/lmestre/ProActive/demo/sc2001/proactive.java.policy fr.inria.proactive.rmi.StartNode ///nodeSolida
14:25:53 (IN -> rsh -l lmestre lo.in) => Process started Thread=IN -> rsh -l lmestre lo.in
14:25:53 (ERR -> rsh -l lmestre lo.in) => Process started Thread=ERR -> rsh -l lmestre lo.in
14:25:55 (IN -> rsh -l lmestre lo.in) => ClassFileServer bound on port 2005 with no codebases (reading resources from classpath)
14:25:55 (IN -> rsh -l lmestre lo.in) => Detected an existing RMI Registry on port 1099
14:25:55 (IN -> rsh -l lmestre lo.in) => ProActive Node successfully bound in registry at //lo.inria.fr/nodeSolida
14:25:55 (IN -> rsh -l lmestre lo.in) => OK. Node nodeSolida created in VM id=ed4be964c03e2377:45a877:ec61849b50:-7ffe
```

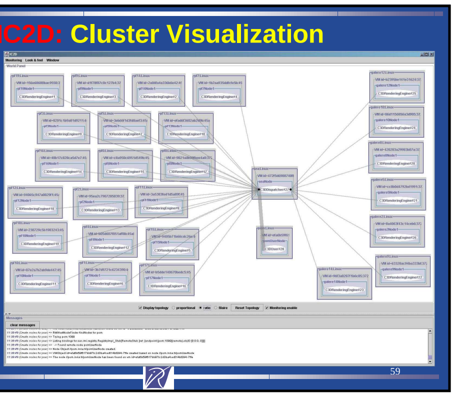
IC2D: Dynamic change of Deployment Drag-n-Drop Migration

Drag-n-Drop
tasks
around the
world


Monitor
Look & feel
Acquire JINI host
Acquire GLOBUS host
Acquire RMI host
Acquire RMI Node
Hide/Show Objects
Toggle player
Legend
sakuraii:Linux
Node1
Other thread #1
C3DDispatcher #11
C3DUser #12
fvieques:Windows NT
defaultNode1
Other thread #3
C3DRenderingEngine #14
Node2
Other thread #6
C3DRenderingEngine #15
oasis:SunOS
Node3
Other thread #8
C3DRenderingEngine #13
camel:Linux
Node4
Other thread #10
C3DRenderingEngine

```
Starting monitor of sent replies for fr.inria.proactive.examples.c3d.C3DRenderingEngine
Starting monitor of sent requests for fr.inria.proactive.examples.c3d.C3DRenderingEngine
Starting monitor of incoming replies for fr.inria.proactive.examples.c3d.C3DRenderingEngine
Object fr.inria.proactive.examples.c3d.C3DRenderingEngine migrate to //camel/Node4
```

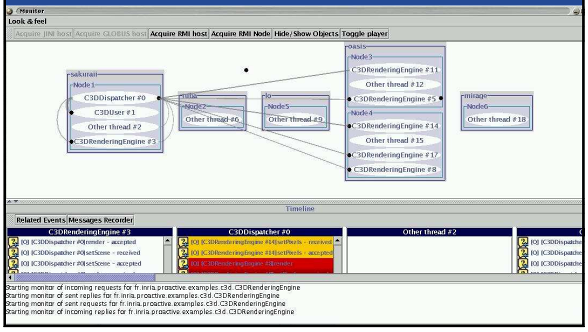
IC2D: Cluster Visualization



Visualization
of 2 clusters
(1Gbits links)

Featuring
the current
communications
(proportional)

IC2D on several machines (2)

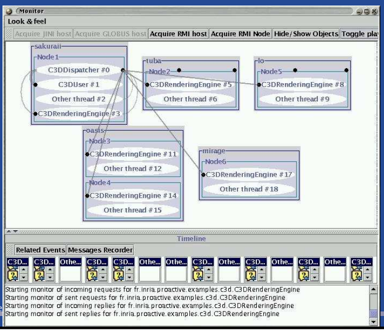

Monitor
Look & feel
Acquire JINI host
Acquire GLOBUS host
Acquire RMI host
Acquire RMI Node
Hide/Show Objects
Toggle player
sakuraii
Node1
C3DDispatcher #0
C3DUser #1
Other thread #2
C3DRenderingEngine #3
tuba
Node2
Other thread #6
lo
Node5
Other thread #9
oasis
Node3
C3DRenderingEngine #11
Other thread #12
C3DRenderingEngine #5
Node4
C3DRenderingEngine #14
Other thread #15
C3DRenderingEngine #17
C3DRenderingEngine #8
mirage
Node6
Other thread #18
Timeline
Related Events
Messages Recorder
C3DRenderingEngine #3
C3DDispatcher #0
Other thread #2

```
[O] [C3DDispatcher #0]render - accepted
[O] [C3DDispatcher #0]setScene - received
[O] [C3DDispatcher #0]setScene - accepted
```

```
[O] [C3DRenderingEngine #14]setPixels - received
[O] [C3DRenderingEngine #14]setPixels - accepted
[O] [C3DRenderingEngine #3]render
```

```
Starting monitor of incoming requests for fr.inria.proactive.examples.c3d.C3DRenderingEngine
Starting monitor of sent replies for fr.inria.proactive.examples.c3d.C3DRenderingEngine
Starting monitor of sent requests for fr.inria.proactive.examples.c3d.C3DRenderingEngine
Starting monitor of incoming replies for fr.inria.proactive.examples.c3d.C3DRenderingEngine
```

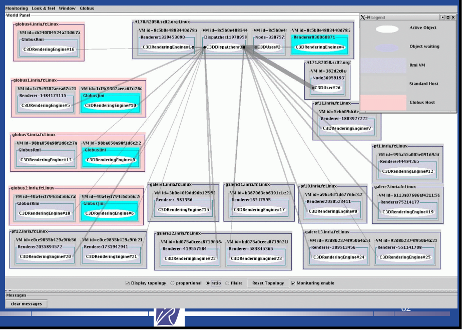
IC2D on several machines (1)


Monitor
Look & feel
Acquire JINI host
Acquire GLOBUS host
Acquire RMI host
Acquire RMI Node
Hide/Show Objects
Toggle pla
sakurai
Node1
C3DDispatcher #0
C3DUser #1
Other thread #2
C3DRenderingEngine #3
tuba
Node2
C3DRenderingEngine #5
Other thread #6
lo
Node5
C3DRenderingEngine #8
Other thread #9
oasis
Node3
C3DRenderingEngine #11
Other thread #12
Node4
C3DRenderingEngine #14
Other thread #15
mirage
Node6
C3DRenderingEngine #17
Other thread #18
Timeline
Related Events
Messages Recorder
C3D...
Othe...

```
Starting monitor of incoming requests for fr.inria.proactive.examples.c3d.C3DRenderingEngine
Starting monitor of sent requests for fr.inria.proactive.examples.c3d.C3DRenderingEngine
Starting monitor of incoming replies for fr.inria.proactive.examples.c3d.C3DRenderingEngine
Starting monitor of sent replies for fr.inria.proactive.examples.c3d.C3DRenderingEngine
```

Monitoring of RMI, Globus, Jini, LSF cluster Nice -- Baltimore at SC'02

Width of links proportional to the number of com-munications


World Panel
Messages
clear messages
Display topology
proportional
ratio
filaire
Reset Topology
Monitoring enable
Active Object
Object waiting
Rmi VM
Standard Host
Globus Host

1.7 DEMO: Applis with the IC2D monitor

- 1. C3D : Collaborative 3D renderer in //
  a standard ProActive application
- 2. Penguin
  a mobile agent application

IC2D: Interactive Control & Debug for Distribution

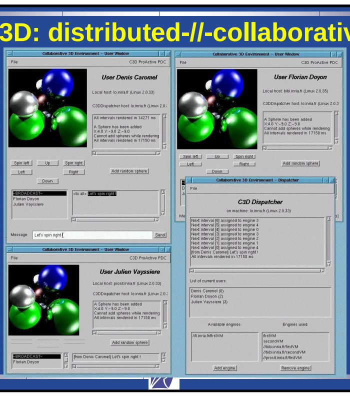
work with any ProActive application

Features:

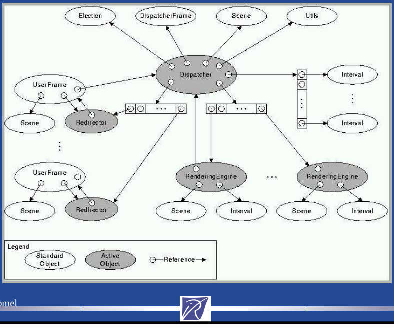
Graphical and Textual visualization

Monitoring and Control

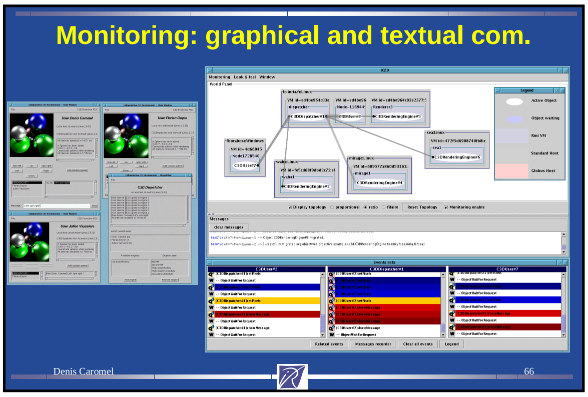
C3D: distributed-//-collaborative


Collaborative 3D Environment - User Window
User Denis Caromel
User Florian Doyon
User Julien Vayssiere
Collaborative 3D Environment - Dispatcher
C3D Dispatcher
Spin left
Up
Spin right
Left
Right
Down
Add random sphere
Message
Send
Add engine
Remove engine

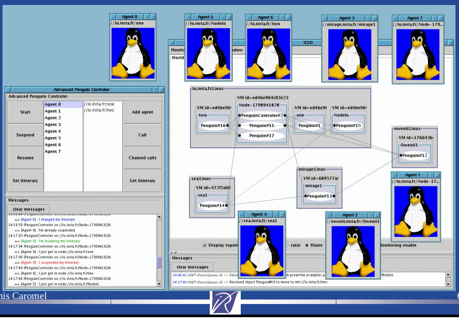
Object Diagram for C3D


Election
DispatcherFrame
Scene
Utils
UserFrame
Dispatcher
Interval
Scene
Redirector
Interval
UserFrame
RenderingEngine
RenderingEngine
Scene
Redirector
Scene
Interval
Scene
Interval
Legend
Standard Object
Active Object
Reference

Monitoring: graphical and textual com.


IC2D
World Panel
Legend
Active Object
Object waiting
Rmi VM
Standard Host
Globus Host
Display topology
proportional
ratio
filaire
Reset Topology
Monitoring enable
Messages
clear messages
Events with
Related events
Messages recorder
Clear all events
Legend

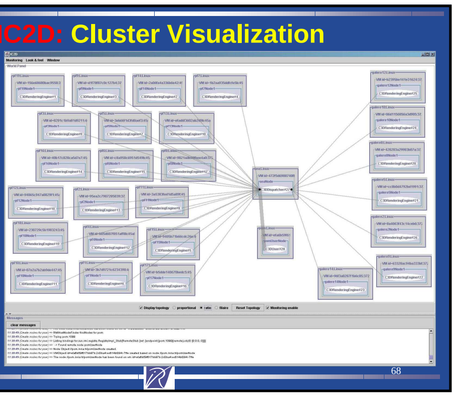
Mobile Application executing on 7 JVMs



IC2D: Cluster Visualization

Visualization
of 2 clusters
(1Gbits links)

Featuring
the current
communications
(proportional)



---

2. *ProActive* : Migration of active objects

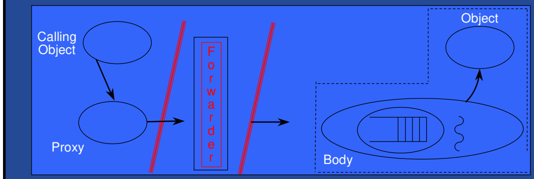
Migration is initiated by the active object itself through a primitive: migrateTo

Can be initiated from outside through any public method

The active object migrates with:
- all pending requests
- all its passive objects
- all its future objects

Automatic and transparent forwarding of:
- requests (remote references remain valid)
- replies (its previous queries will be fullfilled)

ProActive : Migration of active objects



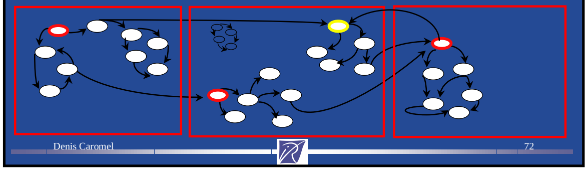
Migration is initiated by the active object through a request

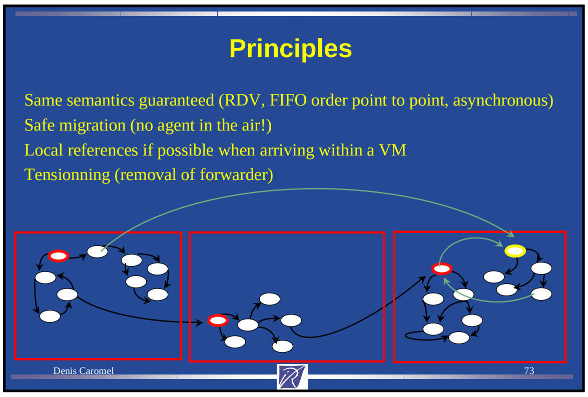
The active object migrates with
- its passive objects - the queue of pending requests - its future objects

2 Techniques : Forwarders or Centralized server

Principles

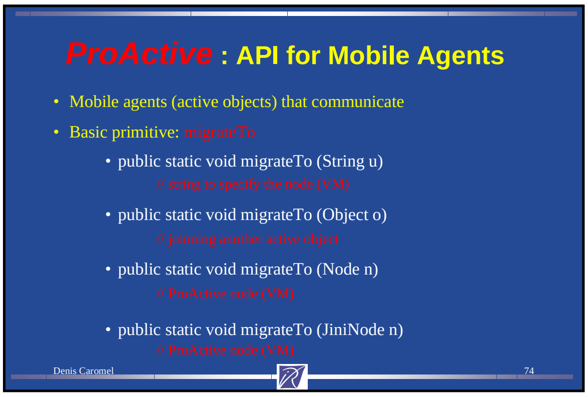
Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
Safe migration (no agent in the air!)
Local references if possible when arriving within a VM
Tensionning (removal of forwarder)





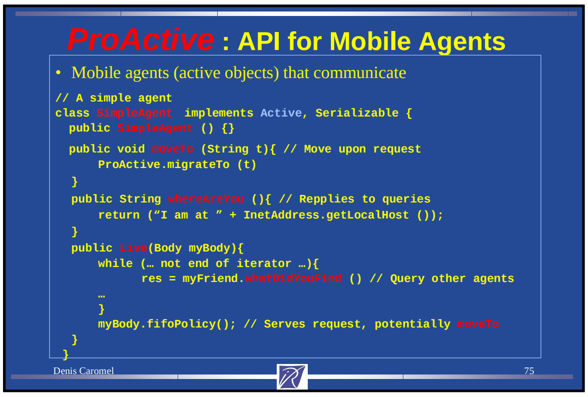
Principles

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
Safe migration (no agent in the air!)
Local references if possible when arriving within a VM
Tensionning (removal of forwarder)



ProActive : API for Mobile Agents

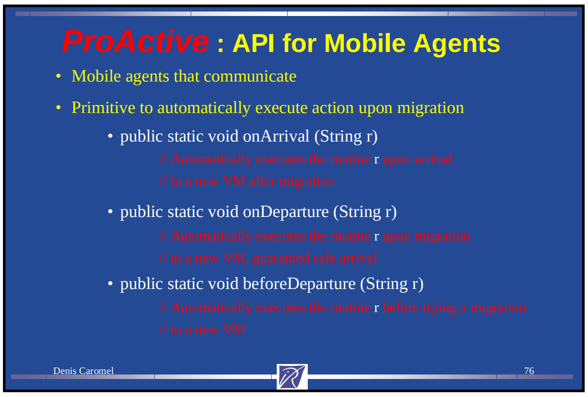
- Mobile agents (active objects) that communicate
- Basic primitive: migrateTo
    - public static void migrateTo (String u)
      // string to specify the node (VM)
    - public static void migrateTo (Object o)
      // joinning another active object
    - public static void migrateTo (Node n)
      // ProActive node (VM)
    - public static void migrateTo (JiniNode n)
      // ProActive node (VM)



ProActive : API for Mobile Agents

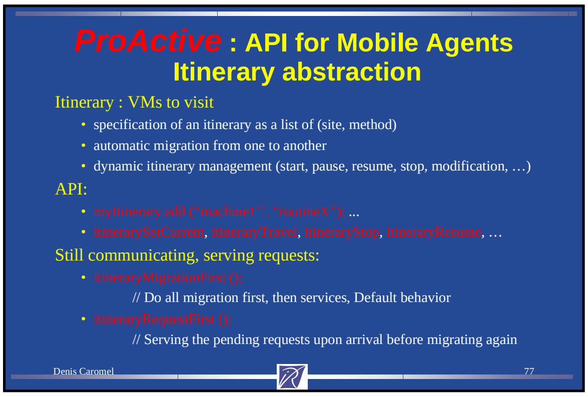
- Mobile agents (active objects) that communicate

```
// A simple agent
class SimpleAgent  implements Active, Serializable {
  public SimpleAgent () {}
  public void moveTo (String t){ // Move upon request
      ProActive.migrateTo (t)
  }
  public String whereAreYou (){ // Repplies to queries
      return (“I am at ” + InetAddress.getLocalHost ());
  }
  public Live(Body myBody){
      while (… not end of iterator …){
            res = myFriend.whatDidYouFind () // Query other agents
      …
      }
      myBody.fifoPolicy(); // Serves request, potentially moveTo
  }
 }
```



ProActive : API for Mobile Agents

- Mobile agents that communicate
- Primitive to automatically execute action upon migration
    - public static void onArrival (String r)
      // Automatically executes the routine r upon arrival
      // in a new VM after migration
    - public static void onDeparture (String r)
      // Automatically executes the routine r upon migration
      // to a new VM, guaranted safe arrival
    - public static void beforeDeparture (String r)
      // Automatically executes the routine r before trying a migration
      // to a new VM



ProActive : API for Mobile Agents
Itinerary abstraction

Itinerary : VMs to visit

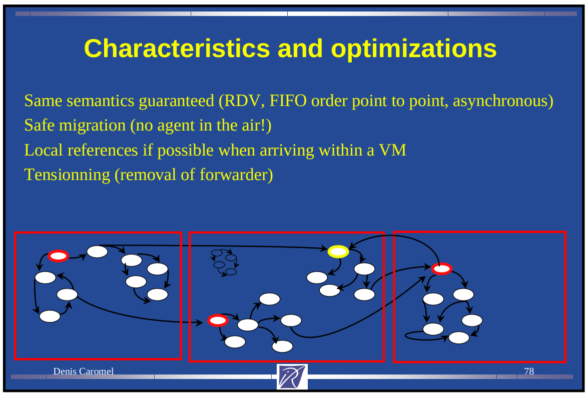
- specification of an itinerary as a list of (site, method)
- automatic migration from one to another
- dynamic itinerary management (start, pause, resume, stop, modification, …)

API:

- myItinerary.add (“machine1’’, “routineX”); ...
- itinerarySetCurrent, itineraryTravel, itineraryStop, itineraryResume, …

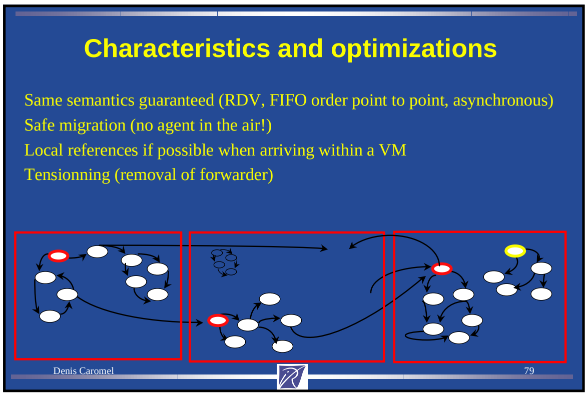
Still communicating, serving requests:

- itineraryMigrationFirst ();
  // Do all migration first, then services, Default behavior
- itineraryRequestFirst ();
  // Serving the pending requests upon arrival before migrating again



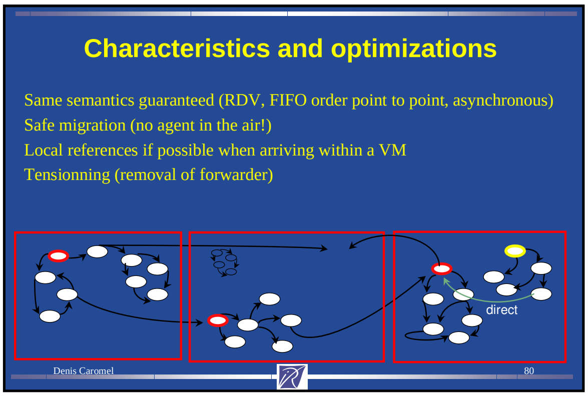
Characteristics and optimizations

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
Safe migration (no agent in the air!)
Local references if possible when arriving within a VM
Tensionning (removal of forwarder)



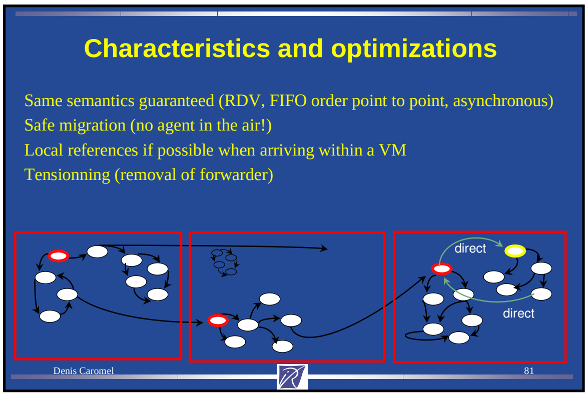
Characteristics and optimizations

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
Safe migration (no agent in the air!)
Local references if possible when arriving within a VM
Tensionning (removal of forwarder)



Characteristics and optimizations

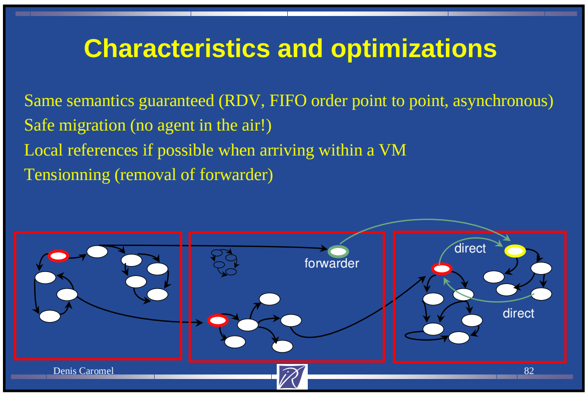
Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
Safe migration (no agent in the air!)
Local references if possible when arriving within a VM
Tensionning (removal of forwarder)



Characteristics and optimizations

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
Safe migration (no agent in the air!)
Local references if possible when arriving within a VM
Tensionning (removal of forwarder)

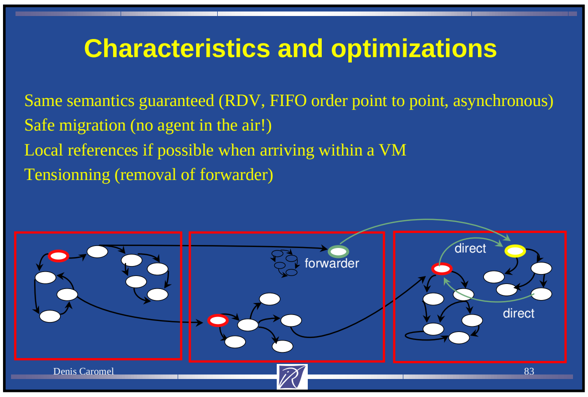
direct
direct



Characteristics and optimizations

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
Safe migration (no agent in the air!)
Local references if possible when arriving within a VM
Tensionning (removal of forwarder)

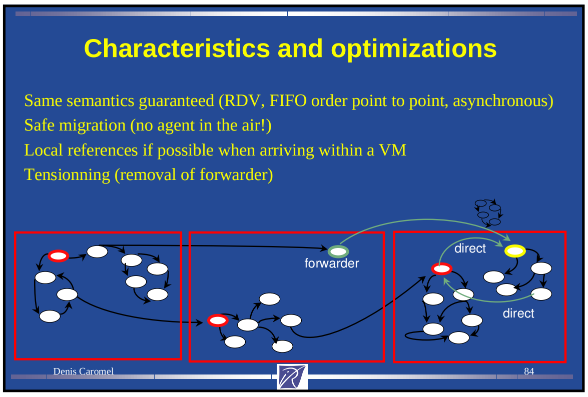
forwarder
direct
direct



Characteristics and optimizations

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
Safe migration (no agent in the air!)
Local references if possible when arriving within a VM
Tensionning (removal of forwarder)

forwarder
direct
direct



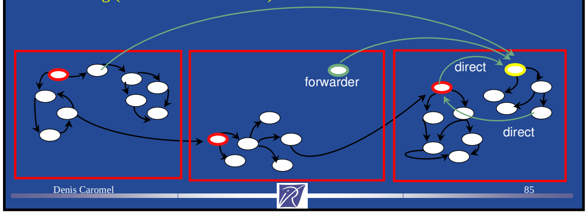
Characteristics and optimizations

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
Safe migration (no agent in the air!)
Local references if possible when arriving within a VM
Tensionning (removal of forwarder)

forwarder
direct
direct

Characteristics and optimizations

Same semantics guaranteed (RDV, FIFO order point to point, asynchronous)
Safe migration (no agent in the air!)
Local references if possible when arriving within a VM
Tensionning (removal of forwarder)


forwarder
direct
direct

Performance Evaluation of Mobile Agent

Together with Fabrice Huet and Mistral Team

Objectives:

- Formally study the performance of Mobile Agent localization mechanism
- Investigate various strategies (forwarder, server, etc.)
- Define adaptative strategies

1-

Analyse Markovienne des répéteurs

Paramètre	Description
$1/\lambda$	Temps moyen d'inactivité de la source
$1/\nu$	Temps moyen d'inactivité de l'agent
$1/\delta$	Durée moyenne de migration
$1/\gamma$	Délai moyen inter-sites

TAB. 4.1 – *Paramètres de la modélisation du mécanisme des répéteurs*

2-

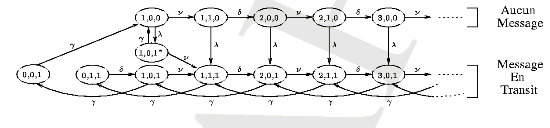

1,0,0
1,1,0
2,0,0
2,1,0
3,0,0
1,0,1*
0,0,1
0,1,1
1,0,1
1,1,1
2,0,1
2,1,1
3,0,1
ν
δ
λ
γ
Aucun Message
Message En Transit

FIG. 4.2 – *États et transitions du mécanisme des répéteurs*

---

3 -

Départ	Transition	Arrivée	Description
(1,0,0)	$\xrightarrow{\nu}$	(1,1,0)	Début de migration
	$\xrightarrow{\lambda}$	(1,0,1*)	Nouveau message généré
$(i,0,0)$ avec $i \geq 2$	$\xrightarrow{\nu}$	$(i,1,0)$	Début de migration
	$\xrightarrow{\lambda}$	$(i,0,1)$	Nouveau message généré
$(i,1,0)$ avec $i \geq 1$	$\xrightarrow{\delta}$	$(i+1,0,0)$	Fin de migration
	$\xrightarrow{\lambda}$	$(i,1,1)$	Nouveau message généré
(1,0,1*)	$\xrightarrow{\nu}$	(1,1,1)	Début de migration
	$\xrightarrow{\gamma}$	(1,0,0)	Message a atteint l'agent
$(i,0,1)$ avec $i \geq 1$	$\xrightarrow{\nu}$	$(i,1,1)$	Début de migration
	$\xrightarrow{\gamma}$	$(i-1,0,1)$	Message a effectué un saut
$(i,1,1)$ avec $i \geq 1$	$\xrightarrow{\delta}$	$(i+1,0,1)$	Fin de migration
	$\xrightarrow{\gamma}$	$(i-1,1,1)$	Message a effectué un saut
(0,1,1)	$\xrightarrow{\delta}$	(1,0,1)	Fin de migration
(0,0,1)	$\xrightarrow{\gamma}$	(1,0,0)	Message a atteint la source

TAB. 4.2 – *Détails des transitions dans le modèle des répéteurs*

---

4 -

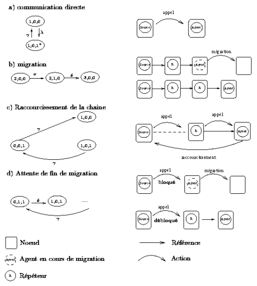

a) communication directe
1,0,0
1,0,1*
γ
λ
appel
Source
Agent
b) migration
2,0,0
2,1,0
3,0,0
ν
δ
migration
R
c) Raccourcissement de la chaine
1,0,0
0,0,1
1,0,1
γ
appel
raccourcissement
d) Attente de fin de migration
0,1,1
1,0,1
.....
bloqué
débloqué
Noeud
Agent en cours de migration
Répéteur
Référence
Action

FIG. 4.3 – *Détails des transitions du diagramme des répéteurs*

5 -

Analyse Markovienne du serveur centralisé

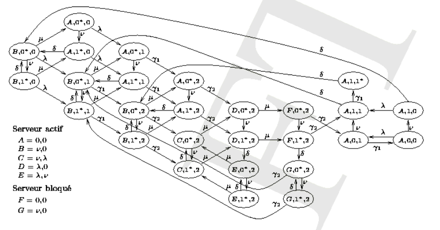

A,0*,0
B,0*,0
A,1*,0
A,0*,1
B,1*,0
B,0*,1
A,1*,1
A,0*,2
A,1,1*
B,1*,1
B,0*,2
A,1*,2
D,0*,2
F,0*,2
A,1,1
A,1,0
B,1*,2
C,0*,2
D,1*,2
F,1*,2
A,0,1
A,0,0
C,1*,2
E,0*,2
G,0*,2
E,1*,2
G,1*,2
Serveur actif
A = 0,0
B = ν,0
C = ν,λ
D = λ,0
E = λ,ν
Serveur bloqué
F = 0,0
G = ν,0

FIG. 4.6 – *État du système et taux de transitions dans le mécanisme du serveur.*

6 -

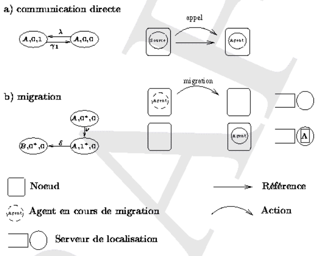

a) communication directe
A,0,1
A,0,0
λ
γ1
appel
Source
Agent
b) migration
migration
A,0*,0
B,0*,0
A,1*,0
ν
δ
A
Noeud
Référence
Agent en cours de migration
Action
Serveur de localisation

FIG. 4.7 – *Détails des transitions du modèle du serveur - source et agent*

7 -

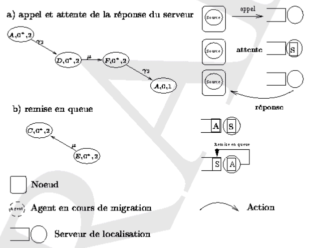

a) appel et attente de la réponse du serveur
A,0*,2
D,0*,2
F,0*,2
A,0,1
γ2
μ
appel
attente
réponse
Source
S
b) remise en queue
C,0*,2
E,0*,2
A
Remise en queue
Noeud
Agent en cours de migration
Action
Serveur de localisation

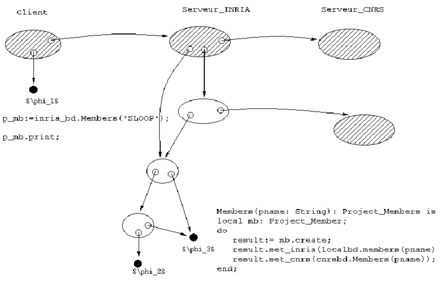
FIG. 4.8 – *Détails des transitions du modèle du serveur - le serveur*

8 -

Paramètre	Description
λ	Attente de la source
ν	Attente de l'agent
δ	Inverse de la durée de migration
γ_1	Inverse de la latence vers l'agent
γ_2	Inverse de la latence vers le serveur
μ	Taux de service

TAB. 4.5 – *Description des paramètres de la modélisation du serveur de localisation*

Automatic Continuations


Client
Serveur_INRIA
Serveur_CNRS
ϕ_1
ϕ_2
ϕ_3

```
p_mb:=inria_bd.Members("SLOOP");

p_mb.print;
```

```
Members(pname: String): Project_Members is
local mb: Project_Member;
do
    result:= mb.create;
    result.set_inria(localbd.members(pname)
    result.set_cnrs(cnrsbd.Members(pname));
end;
```

Transparent Future transmissions (Request,Reply)

ProActive Non Functional Properties

Currently in ProActive:

- Remotely accessible Objects
  - (Classes, not only Interfaces, Dynamic)
- Asynchronous Communications, Futures
- Group Communications (worked on)
- Migration
- Visualization and monitoring (IC2D)
- Non-Functional Exceptions: Handler reification for mobility

Others:

- Security (prototype)
- Components (prototype)
- Communications with disconnected mode (exp. Going on)

Some Perspectives

Non-functional Exceptions:
  - Exception handler (object) attached to Future, Proxy, AO, JVM, middleware
  - Dynamic transmission of handler
  - Use to managed Disconnected Mode (e.g. wireless PDA)

ProActive Components:
  - CCM model (component car { provides …; uses …; emits …; attributes …}
  - Fractal (object web model for implementation)
  - Hierarchical components

Checkpointing:
  - Communication induced checkpoints
  - Message logging

--> Components and Deployment Descriptors integration

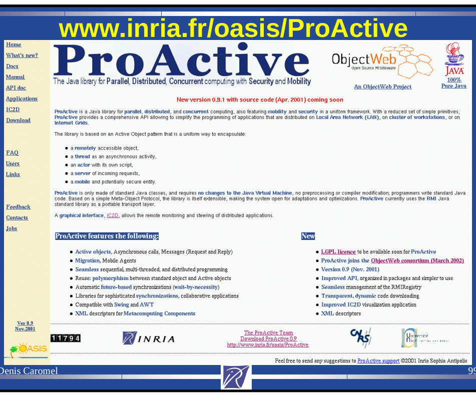
Conclusion

- A library: ProActive 100% Java
- Parallelism, Distribution, Synchronization (CSCW), Group and Mobility
- Reuse -- seamless
  - Polymorphism with existing class types
  - Asynchrony -- Wait-by-necessity
- An interactive tool towards GRID: IC2D
- A calculus: ASP: Asynchronous Sequential Processes
  - Capture the semantics, and demonstrates the independence of activities
  - First results on Confluence and Determinism
  - Mobility to be added

ProActive vs. RMI alone : - 30% of code `www.inria.fr/oasis/ProActive`

---

www.inria.fr/oasis/ProActive



Home
What's new?
Docs
Manual
API doc
Applications
IC2D
Download
FAQ
Users
Links
Feedback
Contacts
Jobs
Ver 0.9
Nov.2001

ProActive

The Java library for Parallel, Distributed, Concurrent computing with Security and Mobility

An ObjectWeb Project

100% Pure Java

New version 0.9.1 with source code (Apr. 2001) coming soon

ProActive is a Java library for parallel, distributed, and concurrent computing, also featuring mobility and security in a uniform framework. With a reduced set of simple primitives, ProActive provides a comprehensive API allowing to simplify the programming of applications that are distributed on Local Area Network (LAN), on cluster of workstations, or on Internet Grids.

The library is based on an Active Object pattern that is a uniform way to encapsulate:

- a remotely accessible object,
- a thread as an asynchronous activity,
- an actor with its own script,
- a server of incoming requests,
- a mobile and potentially secure entity.

ProActive is only made of standard Java classes, and requires no changes to the Java Virtual Machine, no preprocessing or compiler modification; programmers write standard Java code. Based on a simple Meta-Object Protocol, the library is itself extensible, making the system open for adaptations and optimizations. ProActive currently uses the RMI Java standard library as a portable transport layer.

A graphical interface, IC2D, allows the remote monitoring and steering of distributed applications.

ProActive features the following:

- Active objects, Asynchronous calls, Messages (Request and Reply)
- Migration, Mobile Agents
- Seamless sequential, multi-threaded, and distributed programming
- Reuse: polymorphism between standard object and Active objects
- Automatic future-based synchronizations (wait-by-necessity)
- Libraries for sophisticated synchronizations, collaborative applications
- Compatible with Swing and AWT
- XML descriptors for Metacomputing Components

New

- LGPL licence to be available soon for ProActive
- ProActive joins the ObjectWeb consortium (March 2002)
- Version 0.9 (Nov. 2001)
- Improved API, organized in packages and simpler to use
- Seamless management of the RMIRegistry
- Transparent, dynamic code downloading
- Improved IC2D visualization application
- XML descriptors

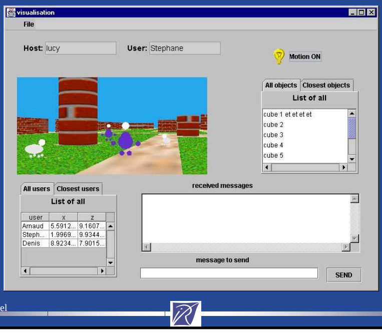
11794

The ProActive Team
Download ProActive 0.9
http://www.inria.fr/oasis/ProActive

Feel free to send any suggestions to ProActive support ©2001 Inria Sophia Antipolis

---

DIVA: Distributed Int. Virtual World in Java



Host: lucy
User: Stephane

cube 1 et et et et
cube 2
cube 3
cube 4
cube 5

user	x	z
Arnaud	5.5912...	9.1607...
Steph...	1.9969...	9.9344...
Denis	8.9234...	7.9015...

Extra Material

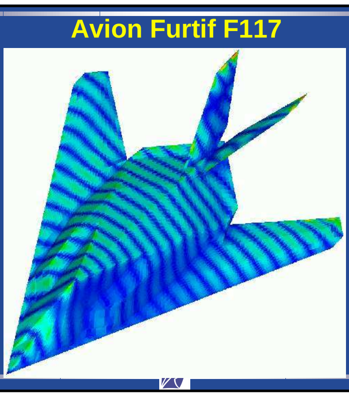
An Object-Oriented Application for 3D Electromagnetism

Together with:
Francoise Baude, Roland Bertuli, Christian Delbe (OASIS),
Said El Kasmi, Stéphane Lanteri (CAIMAN)

3D Electromagnetism Applications:
- Visualize the radar reflection of a plane

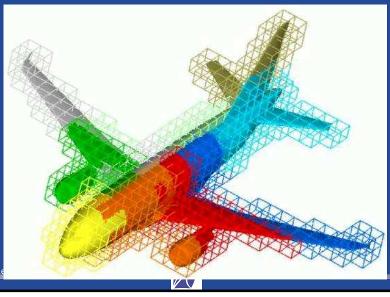
Goals:
- Sequential object-oriented design, modular and extensible
- Designed to allow both Element and Volume type methods
  --> currently 3D Maxwell equations, towards CFD
- Valid on structured, unstructured, or hybrid meshes.
- Sequential version can be smoothly distributed,
  --> keeping structuring and object abstractions

Avion Furtif F117



Airbus A318

Meshing, 1 color par processor, 9 here



Geometry definition

Meshes: Vertices, and Elements

Numerical Method: Finite Volume Methods (vs. Finite Element Methods)

- a very general method

Computation of a flux balance through the boundary of Control Volume

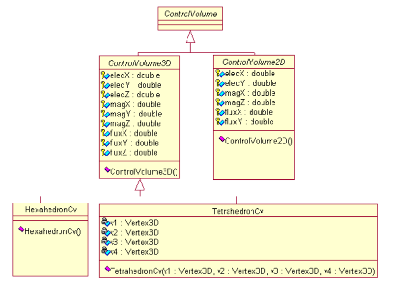
- the calculation support of FVM (vs. Vertices of the mesh)

Example, and experimentation:

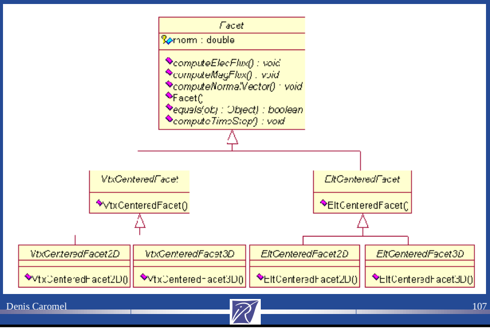
- Control Volume = Tetrahedron
- Facet = Triangle

----> Picture

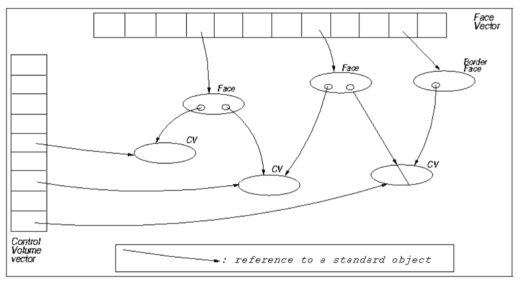
Control Volume in 2D and 3D


ControlVolume
ControlVolume3D
elecX : double
elecY : double
elecZ : double
magX : double
magY : double
magZ : double
fuxX : double
fuxY : double
fuxZ : double
ControlVolume3D()
ControlVolume2D
elecX : double
elecY : double
magX : double
magZ : double
fluxX : double
fluxY : double
ControlVolume2D()
HexahedronCv
HexahedronCv()
TetrahedronCv
v1 : Vertex3D
v2 : Vertex3D
v3 : Vertex3D
v4 : Vertex3D
TetrahedronCv(v1 : Vertex3D, v2 : Vertex3D, v3 : Vertex3D, v4 : Vertex3D)

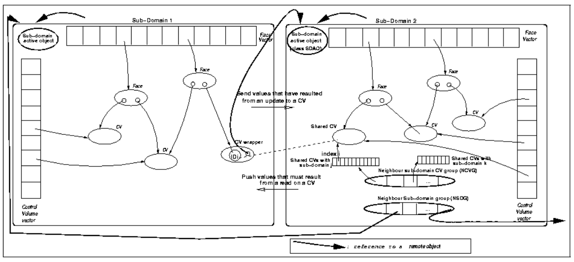
Facets in 2D and 3D


Facet
norm : double
computeElecFlux() : void
computeMagFlux() : void
computeNormalVector() : void
Facet()
equals(obj : Object) : boolean
computeTimeStep() : void
VtxCenteredFacet
VtxCenteredFacet()
EltCenteredFacet
EltCenteredFacet()
VtxCenteredFacet2D
VtxCenteredFacet2D()
VtxCenteredFacet3D
VtxCenteredFacet3D()
EltCenteredFacet2D
EltCenteredFacet2D()
EltCenteredFacet3D
EltCenteredFacet3D()

Architecture of the sequential version


Face Vector
Face
Face
Border Face
CV
CV
CV
Control Volume vector
: reference to a standard object

Architecture of the distributed version


Sub–Domain 1
Sub–domain active object
Face Vector
Face
Face
CV
CV
CV wrapper
IDi
Control Volume vector
Send values that have resulted from an update to a CV
Push values that must result from a read on a CV
Sub–Domain 2
Sub–domain active object (class SDAO)
Face Vector
Face
Face
CV
CV
Shared CV
index i
Shared CVs with sub–domain j
Shared CVs with sub–domain k
Neighbour sub-domain CV group (NCVG)
...
Neighbour Sub–domain group (NSDG)
...
Control Volume vector
: reference to a remote object