ProActive
Programming, Composing, Deploying on the Grid

Open Source Middleware for the GRID

Denis Caromel, et al.
www.inria.fr/oasis/ProActive
OASIS Team
INRIA -- CNRS - I3S -- Univ. of Nice Sophia-Antipolis, IUF
Barcelona Lectures, UPC, 27-29 Sept. 2005, Spain

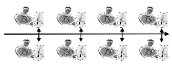
- Remote Objects, Asynchronous Method Calls, Futures
- Groups, OO SPMD, Mobile Objects, Component
- Graphical Interface (IC2D), XML Deployment, Security
- Interfaced with Globus, rsh, ssh, LSF, PBS, RMIregistry, Jini

Université Nice SOPHIA ANTIPOLIS — CNRS CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE — INRIA — ObjectWeb Open Source Middleware

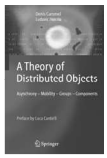
Barcelona Agenda

ProActive Middleware

Practical Sessions



Theory: ASP Calculus



To be precisely defined
according to students knowledge and main interests

ProActive:

A Java API + Tools for Parallel, Distributed Computing

A uniform framework: **An Active Object pattern**
A formal model behind: **Determinism (POPL'04)**

Programming Model:
- **Remote Objects**
- **Asynchronous Communications, Wait-By-Necessity**
- **Groups, Mobility, Components, Security, Fault-Tolerance: Checkpoints**

Environment:
- **XML Deployment Descriptors**
- **Interfaced with: `rsh, ssh, LSF, PBS, Globus, Jini, SUN Grid Engine`**
- **Graphical Visualization and monitoring: IC2D**

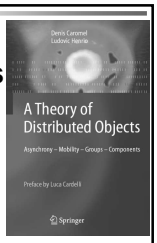
In the www. ObjectWeb .org Consortium (Open Source **LGPL**)

ObjectWeb Open Source Middleware

THEORY

A Theory of Distributed Objects

D. Caromel, L. Henrio,
Springer 2005, Monograph



A Calculus:

ASP: Asynchronous Sequential Processes
- **Based on Sigma-Calculus (Abadi-Cardelli)**
- **Formal Proofs of determinism**
- **Releases a few important implementation constraints**

Table of Contents (1)

Table of Contents (2)

Table of Contents (3)

ProActive
Programming, Composing, Deploying on the Grid

Programming

W r a p p i n g

Composing

Deploying

Figures: Web Page Hits: ~ 4 000 / month,
Downloads: 150-300 / month, Users: ?? .us, .mx, .br, .cl, .ch, .it, .fr, ...

ProActive:

A Java API + Tools for Parallel, Distributed Computing

A uniform framework: **An Active Object pattern**
A formal model behind: **Determinism (POPL'04)**

Programming Model:
- **Remote Objects**
- **Asynchronous Communications, Wait-By-Necessity**
- **Groups, Mobility, Components, Security, Fault-Tolerance: Checkpoints**

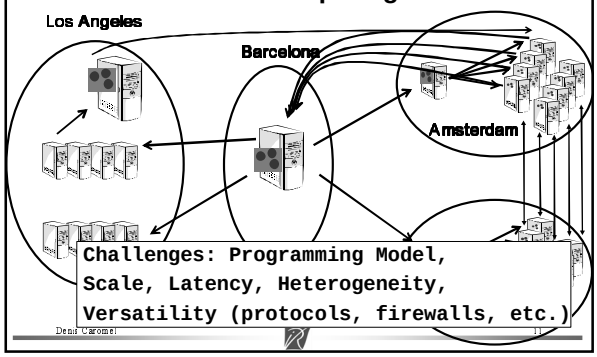
Environment:
- **XML Deployment Descriptors**
- **Interfaced with: `rsh, ssh, LSF, PBS, Globus, Jini, SUN Grid Engine`**
- **Graphical Visualization and monitoring: IC2D**

In the www. ObjectWeb .org Consortium (Open Source **LGPL**)

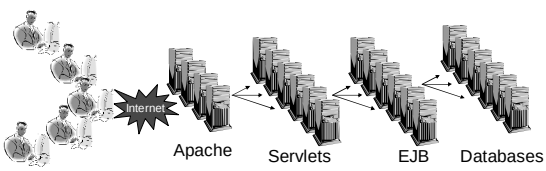
ObjectWeb
Open Source Middleware

GRIDs

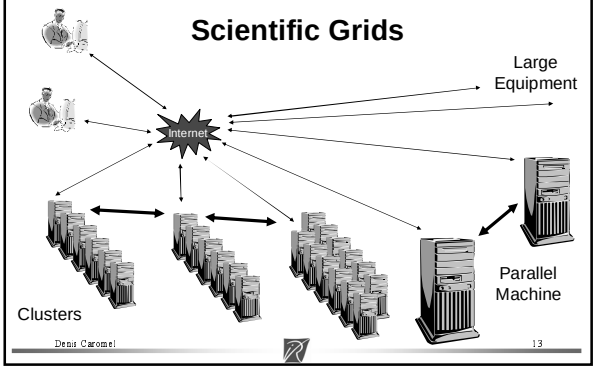
Grid Computing



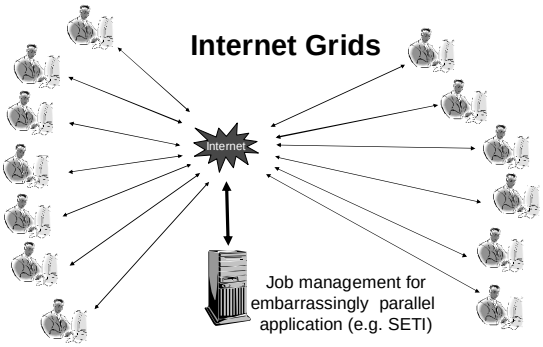
Enterprise Grids



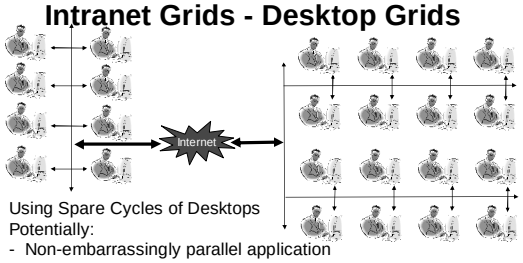
Scientific Grids



Internet Grids



Intranet Grids - Desktop Grids



Using Spare Cycles of Desktops
Potentially:
- Non-embarrassingly parallel application
- Across several sites

How to Program ?
How to Deploy ?

The multiple GRIDs

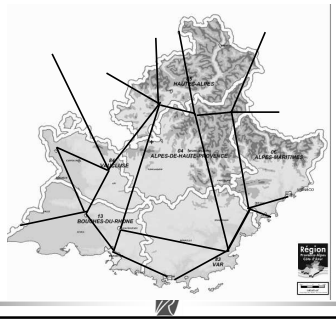
- Scientific Grids
- Enterprise Grids
- Internet Grids
- Intranet Desktop Grids

Strong convergence in process!

At least at the infrastructure level, i.e. WS

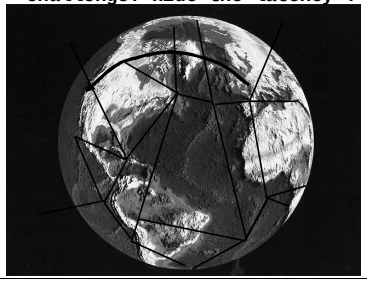
Grid: from enterprise ... to regional

Very hard deployment problems ... right from the beginning

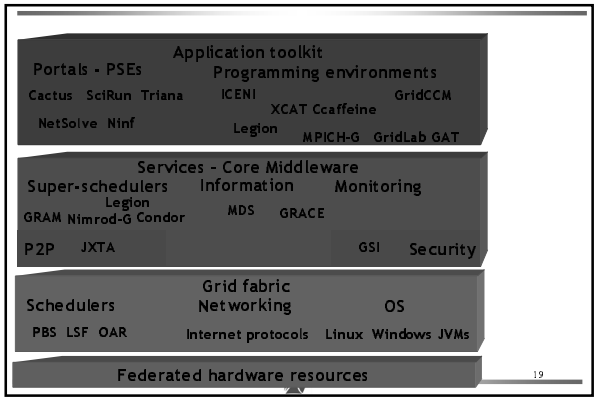


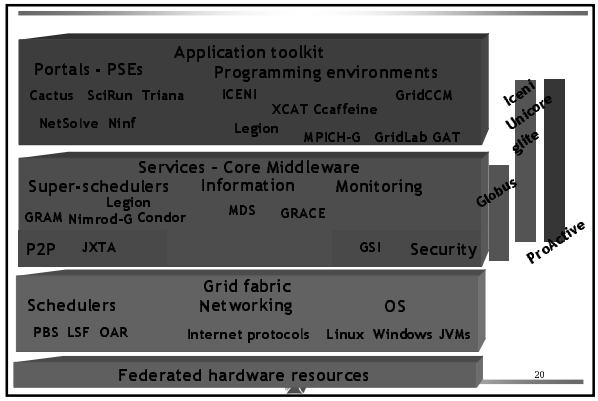
Grid: from regional ... to worldwide

Communication Nice-Los Angeles: 70 ms Light Speed
Challenge: Hide the latency !



Define adequate programming model





1. Distributed Objects

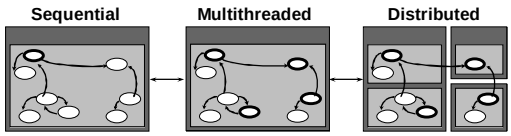
ProActive

Programming

ObjectWeb
Open Source Middleware

ProActive
Objectives and Rationale

Seamless

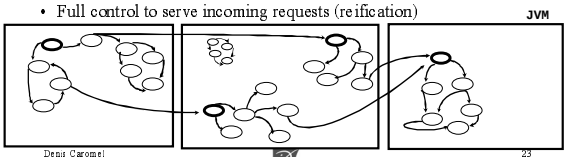


- Most of the time, activities and distribution are not known at the beginning, and change over time
- Seamless implies reuse, smooth and incremental transitions

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)



ProActive model (2)

Java RMI (Remote Method Invocation = Object RPC = o.foo(p))

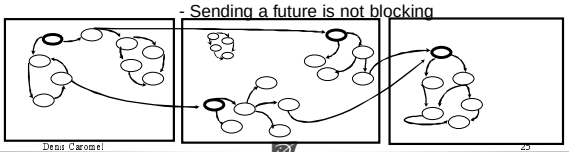
plus a few important features:

- Sequential Object: a single thread with FIFO service
- Asynchronous Method calls towards Active Objects:
  Implicit Futures as method results
- Wait-By-Necessity:
  - Automatic wait upon a strict operation on an unknown future
  - First-Class Futures:
    - Futures can be passed to other activities
    - Sending a future to another machines is not blocking

ProActive model (3)

Java RMI (Remote Method Invocation = Object RPC = o.foo(p))
plus a few important features:

- Asynchronous Method calls towards Active Objects:
  Implicit Futures as RMI results
- Wait-By-Necessity:
  - Automatic wait upon the use of an implicit future
  - First-Class Futures:
    - Futures passed to other activities
    - Sending a future is not blocking



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):**

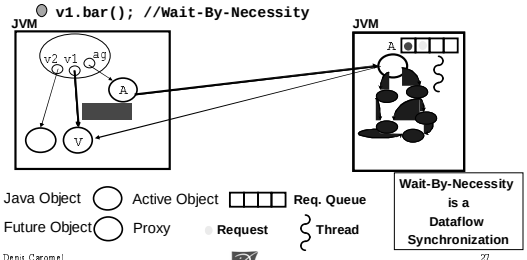
```
class pA extends A implements RunActive { … }
```

- **Object-based:**

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


ProActive : Active objects

```
A ag = newActive ("A", […], VirtualNode)
V v1 = ag.foo (param);
V v2 = ag.bar (param);
...
v1.bar(); //Wait-By-Necessity
```



Java Object, Active Object, Req. Queue, Future Object, Proxy, Request, Thread

Wait-By-Necessity is a Dataflow Synchronization

Wait by necessity

- A call on an active object consists in 2 steps
  - A query : name of the method, parameters…
  - A Reply : the result of the method call
- A query returns a Future object which is a placeholder for the result
- The callee will update the Future when the result is available
- The caller can continue its execution event if the Future has not been updated

```
foo ()
{
  Result r = a.process();
  //do other things
  ...
  r.toString();
}
```

`will block if not available`

```
Result process()
{
  //perform long
  //calculation
  ...
  return result;
}
```


ProActive : Explicit Synchronizations

```
A ag = newActive ("A", […], VirtualNode)
V v = ag.foo(param);
...
v.bar(); //Wait-by-necessity
```

Single Future Synchronization:

- `ProActive.isAwaited (v);`
- `.waitFor (v);`

Vectors of Futures:

- `.waitForAll (Vector); // getOne`
- `.waitForAny (Vector);`

Group Synchronization:

- `ProActiveGroup.waitOne(groupB); // getOne`
- `.waitAll(groupB);`

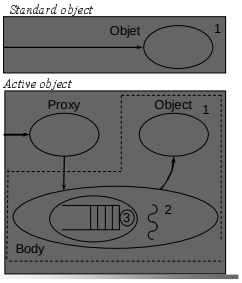
Group Predicates:

- `noneArrived, kArrived(i), allAwaited, ...`

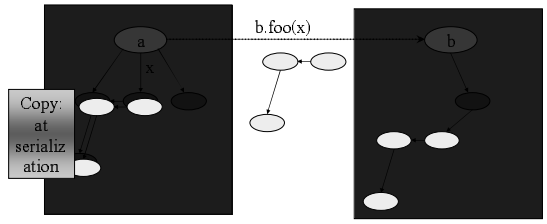
ProActive : Active object

An active object is composed of several objects :

- The object being activated: Active Object (1)
- A set of standard Java objects
- A single thread (2)
- The queue of pending requests (3)



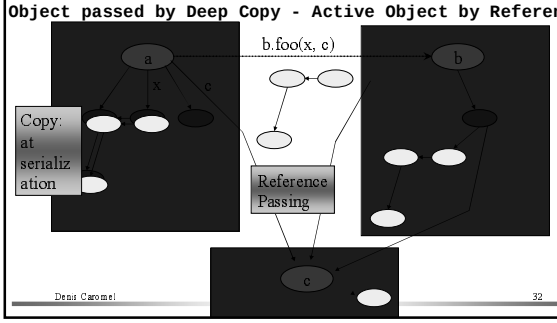
Call between Objects: Parameter passing: Copy of Java Objects



(Deep) Copies evolve independently -- No consistency

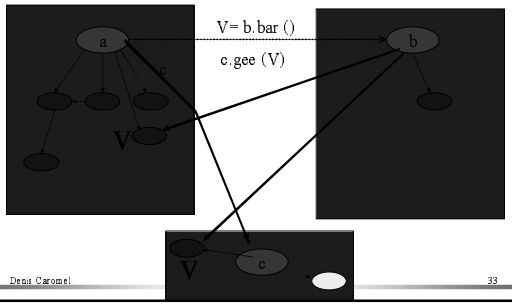
Call between Objects: Parameter Passing: Active Objects

Object passed by Deep Copy - Active Object by Reference



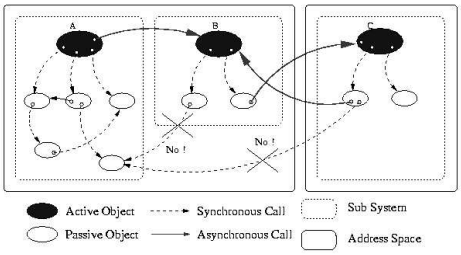
Wait-By-Necessity: First Class Futures

Futures are Global Single-Assignment Variables



Standard system at Runtime: Dynamic Topology, Asynchrony, WbN, ... but no sharing

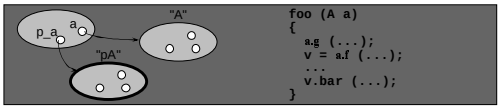
Proofs of Determinism



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)



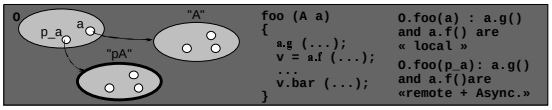
- Wait-by-necessity: inter-object synchronization
  - Systematic, implicit and transparent futures

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)



- 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:

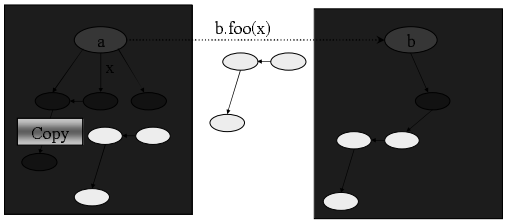
- Non-blocking services,...
  - `serveOldest ();`
  - `serveOldest (f);`
- Blocking services, timed, etc.
  - `serveOldestBl ();`
  - `serveOldestTm (ms);`
- Waiting primitives
  - `waitARequest();`
  - `etc.`

```
class BoundedBuffer extends FixedBuffer
                implements RunActive {

    // Programming Non FIFO behavior
 runActivity (ExplicitBody myBody) {
   while (...) {
     if (this.isFull())
          serveOldest("get");
     else if (this.isEmpty())
          serveOldest ("put");
     else serveOldest ();

     // Non-active wait
     waitArequest ();
   }
 }}
```

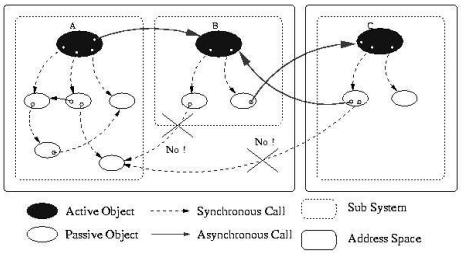
Implicit (declarative) control: library classes

```
e.g. : Blocking Condition Abstraction for concurrency control:
          doNotServe ("put", "isFull");
```


Optimization: SharedOnRead
Call between Objects



Standard system at Runtime



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

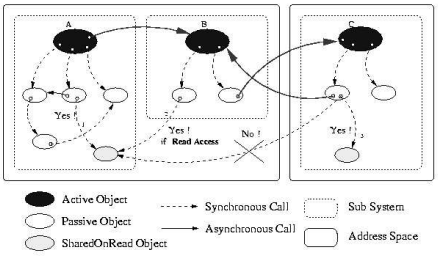
SharedOnRead (2)

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)



Adaptive Feature: Multi-transports layer RMI, RMI-ssh, …, Ibis, HTTP XML, ...

Adaptive choice of transport layer between:

- RMI
- ssh/RMI

Also available with static configuration:

- Ibis (TCP, Myrinet, etc.)
- HTTP
- ... ssh/HTTP

Short Term Perspective:

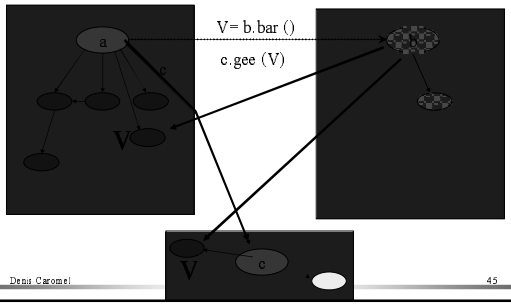
Fully Adaptive Choice between all transports

First-Class Futures

Update

Wait-By-Necessity: First Class Futures

Futures are Global Single-Assignment Variables



Future update strategies

No partial replies and requests:

- No passing of futures between activities, more deadlocks

Eager strategies: as soon as a future is computed

- Forward-based:
  - Each activity is responsible for updating the values of futures it has forwarded
- Message-based:
  - Each forwarding of future generates a message sent to the computing activity
  - The computing activity is responsible for sending the value to all

Mixed strategy:

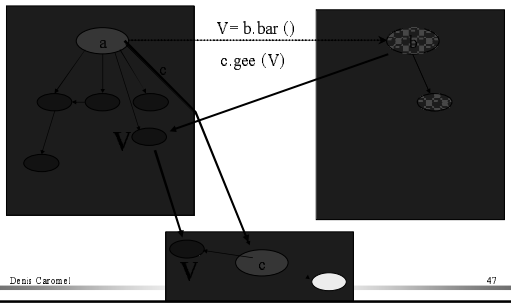
- Futures update any time between future computation and WbN

Lazy strategy:

- On demand, only when the value of the future is needed (WbN on it)

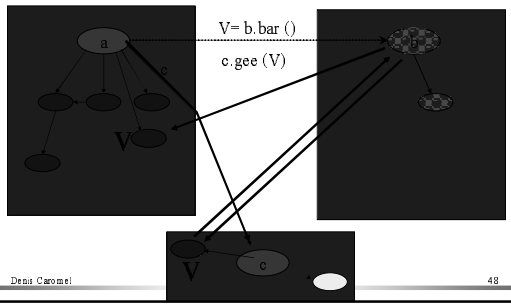
Wait-By-Necessity: Eager Forward Based

AO forwarding a future: will have to forward its value



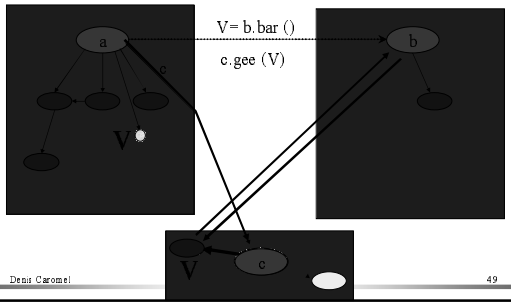
Wait-By-Necessity: Eager Message Based

AO forwarding a future: send a message



Wait-By-Necessity: Lazy Strategy

An Active Object requests a Future Value when needed



TYPED

ASYNCHRONOUS

GROUPS

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**
- **Based on the ProActive communication mechanism**
  - Replication of N ' single ' communications
  - Parallel calls within a group (latency hiding)
  - preserve the « rendez-vous »

Groups: Analysis of Related Works

3 projects close to our work:

- JGroups
  - Centered on low layers of communication
  - Socket programming style
- Object Group Service
  - CORBA focused
  - Group communication return only one result (by default)
- Group Method Invocation
  - Group members have to implement and extend a class and an interface

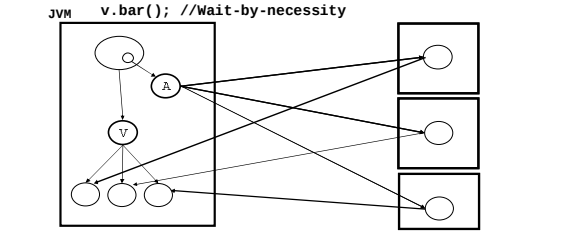
Groups: Analysis of Related Works

- Lack of transparency:
  - Specific interface for functional calls
  - Inheritance from specific classes and interfaces
- Sometimes too application / domain-specific

ProActive Groups address those problems !

Creating AO and Groups

```
A ag = newActiveGroup ("A", […], VirtualNode)
V v = ag.foo(param);
...
v.bar(); //Wait-by-necessity
```



Group, Type, and Asynchrony are crucial for Cpt. and GRID

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 STANDARD OBJECTS « A »
A agS = (A)ProActiveGroup.newGroup("A",{…})
V v = ag.foo(param); // foo(param) invoked
                    // on each member
// A group v of result of type V is created
```

```
// Build a group of ACTIVE OBJECTS « A »
A ag = (A) newActiveGroup("A",{…}, Nodes)
V v=ag.foo(param);//invoked // on each member
```

```
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 property: order of group member = order of reply origin

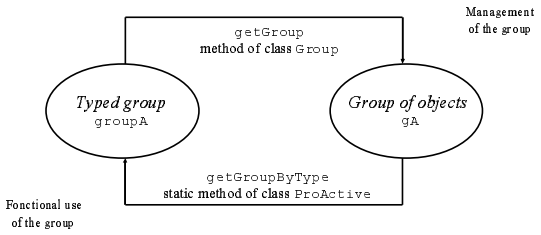
Synchronization primitive

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

Predicates:

```
noneArrived, kArrived, allArrived, ...
```


Two Representation Scheme



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

Broadcast is the default behavior

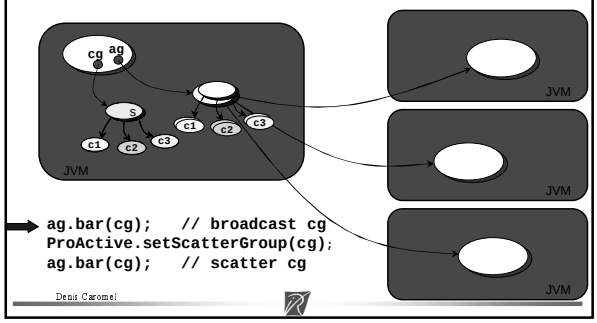
Scatter is also possible

- Scatter uses a group as parameter
- Distribution relies on rank

```
ag.bar(cg, dg); // broadcast cg and dg
ProActive.setScatterGroup(cg);
ag.bar(cg, dg); // scatter cg, still broadcast dg
```

One call can both **scatter** and **broadcast**

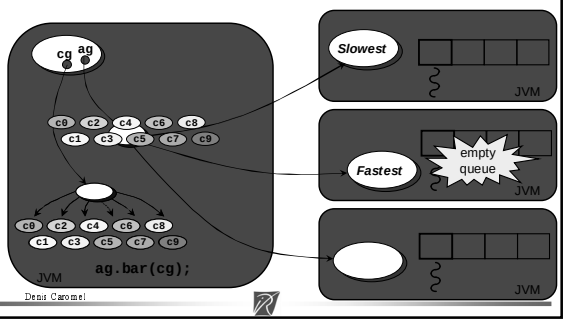
Broadcast and Scatter




```
ag.bar(cg);   // broadcast cg
ProActive.setScatterGroup(cg);
ag.bar(cg);   // scatter cg
```

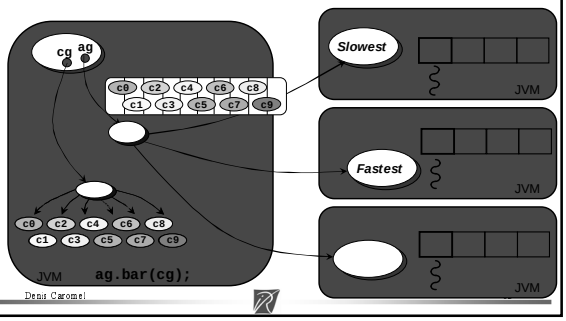
Denis Caromel

Static Dispatch Group



Denis Caromel

Dynamic Dispatch Group



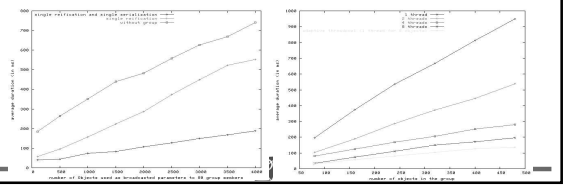
Denis Caromel

Performances and Optimizations

Common operations

- Single reification
- Single serialization

Parallel calls (adaptive thread pool)



Handling Group Failures (1)

Using Java exceptions

- Common way to express failure in Java

A Group may produce N exceptions during one Group Com.

Result Group is used to store exceptions

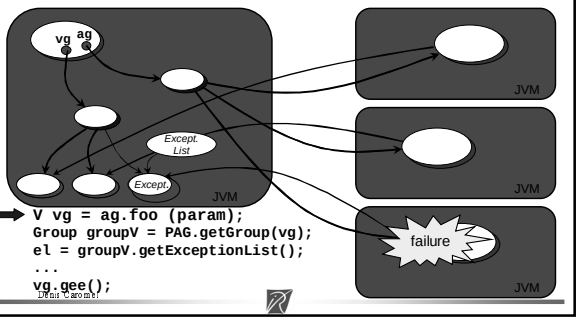
To get them all at once:

```
E.L. el = group.GetExceptionList()
```

`ExceptionList:` an exception that contains all raised exceptions + a reference to the Group that produced them

Denis Caromel

Handling Group Failures (2)




```
V vg = ag.foo (param);
Group groupV = PAG.getGroup(vg);
el = groupV.getExceptionList();
...
vg.gee();
```

Denis Caromel

Active and Hierarchical Groups

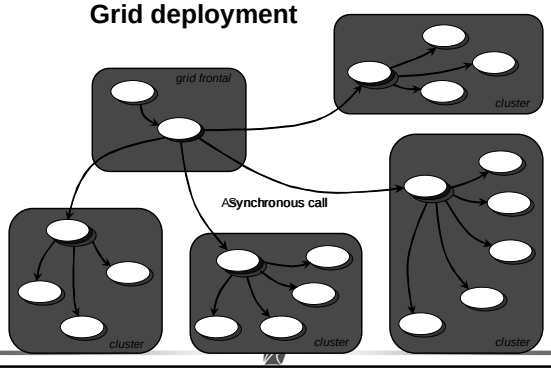
Active Group: a group being a (Remote) Active Object

- A group becomes remotely accessible
- Multiple coherent view of a single Group from several JVMs
- Modifiable service policy

Hierarchical group

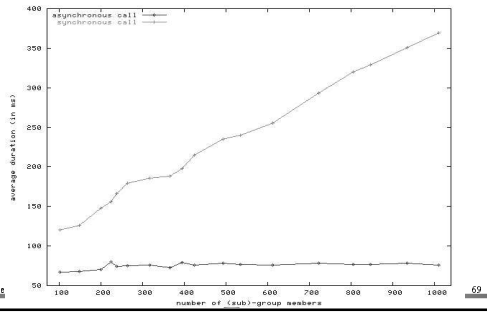
- A group of groups
- Dynamically built
- Achieves scalability

Grid deployment



Grid Group Calls on a 1000 CPU at once

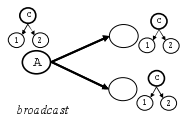
Grid'5000

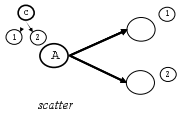


Broadcast or Scatter for Parameters

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

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





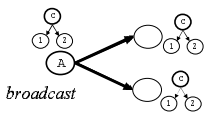
Parameters: Broadcast or Scatter

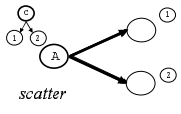
Broadcast is the default behavior

Scatter is also possible

- use a group as parameter
- Scattered depends on rankings

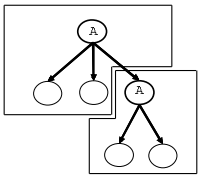
```
gA.bar(gC, p);   // broadcast gC
ProActive.setScatterGroup(gC, p);
ga.bar(gC, p);   // scatter gC
```





Hierarchical Groups

- Add a typed group into a typed group
  vs. Adding an element in a group
- Scalability of groups
- Can match the network topology



Implementation

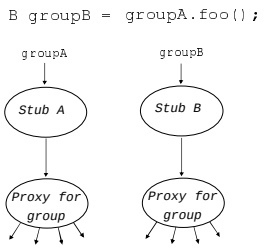
MOP generates Stub

Stub inherits from the class of object

Stub connects a proxy

special proxy for group

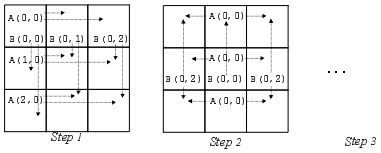
result is stub+proxy



Example : Matrix multiplication

Matrix code : Broadcast to Broadcast approach

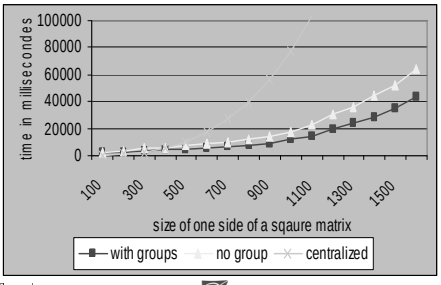
- more than 20 lines of Java code



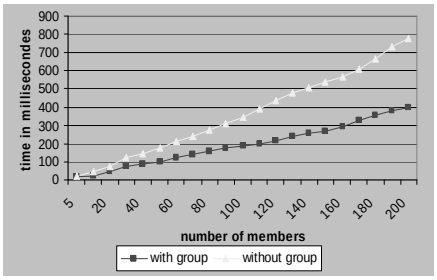
- 2 lines with ProActive groups

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


Measurement : Matrix Multiplication



Measurement : Method Call



Features of 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

Flexibility: Active Group

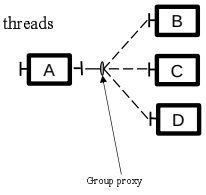
- A group becomes remotely accessible so: updateable and consistent
  ---> Dynamic changes
  ---> Migration of a group is possible

Perspective: using network multicast

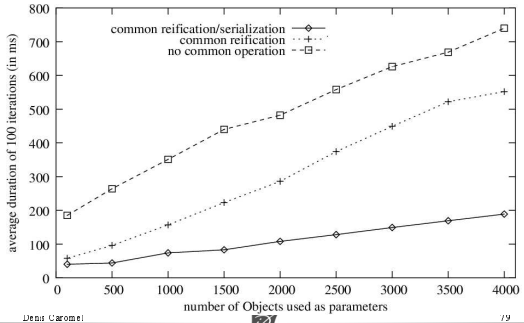
Adaptive Feature: Parallel Group Communications

Adaptive strategy to manage the number of threads

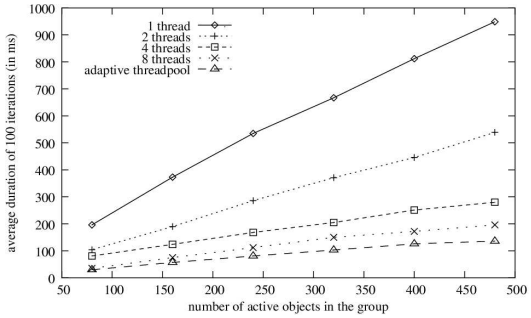
Execution of N calls in //



Different Optimizations in Groups



Thread Pool for sending Requests in //



Object-Oriented SPMD

Motivation

- Cluster / GRID computing
- SPMD programming notoriety

Able to express most of MPI's collective communications

- broadcast
- scatter
- gather
- reduce
- allscatter
- allgather

Main MPI problems for the Grid

Too static in design

Too complex interface (API)

- More than 200 primitives and 80 constants

Too many specific primitives to be adaptive

- `Send`, `Bsend`, `Rsend`, `Ssend`, `Ibsend`, etc.

Typeless (message passing)

Manual management of complex data structures

Object-Oriented SPMD

(Single Program Multiple Data)

Motivation

- Cluster / Grid programming
- SPMD programming widely used

Already able to express most of the MPI's collective communications

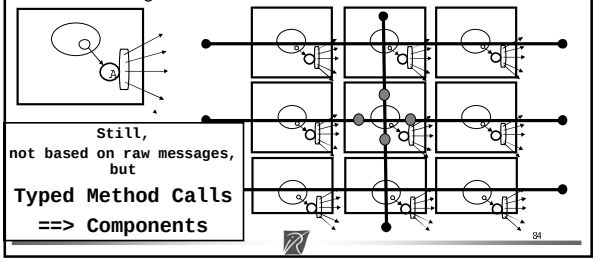
- Broadcast
- Scatter
- Gather
- Reduce
- All scatter
- All gather

Different barriers

Topologies

OO SPMD

```
A ag = newSPMDGroup ("A", […], VirtualNode)
        // In each member
        myGroup.barrier ("2D"); // Global Barrier
        myGroup.barrier ("vertical"); // Any Barrier
        myGroup.barrier ("north","south","east","west");
```



Still,
not based on raw messages,
but

Typed Method Calls
==> Components

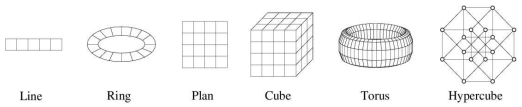
API

Topologies
- Table, Ring, Plan, Torus, Cube, ...
- Open API
- Neighborhood

Barrier
- Global
- Neighbor-based
- Method-based

Topologies

Topologies are typed groups
Open API
Neighborhood
Creation by extraction



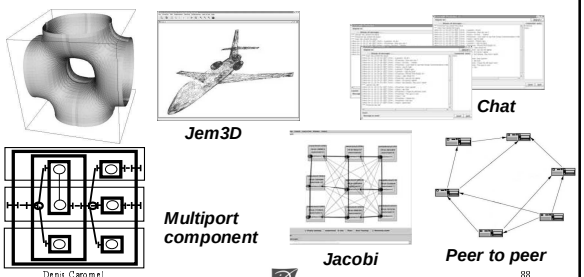

```
Plan plan = new Plan(groupA, Dimensions);
Line line = plan.getLine(0);
```


ProActive OO SPMD

A simple communication model
- Small API
- No "Receive" but data flow synchronization
- No message passing but RPC
- User defined data structure (Object)
- SPMD groups are dynamic
- Efficient and dedicated barrier

Group communication is now a key feature of ProActive

Used in many applications and other features



MPI Communication primitives

For some (historical) reasons, MPI has many com. Primitives:

MPI_Send	Std	MPI_Recv	Receive
MPI_Ssend	Synchronous	MPI_Irecv	Immediate
MPI_Bsend	Buffer	... (any) source, (any) tag,	
MPI_Rsend	Ready		
MPI_Isend	Immediate, async/future		
MPI_Ibsend, ...			

I'd rather put the burden on the implementation, not the Programmers !

How to do adaptive implementation in that context ?

Not talking about:
- the combinatory that occurs between send and receive
- the semantic problems that occur in distributed implementations

Is Recv at all needed ? First adaptive feature: Dynamic Control Flow of Mess.

Main MPI problems for the GRID

Too static in design

Too complex in Interface (API)

Too many specific primitives to be adaptive

Type Less

Adaptive GRID

The need for adaptive middleware is now acknowledged,
with dynamic strategies at various points in containers, proxies, etc.

Can we afford adaptive GRID ?

with dynamic strategies at various points
(communications, groups, checkpointing, reconfiguration, ...)
for various conditions (LAN, WAN, network, P2P, ...)

HPC vs. HPC
High Performance Components vs. High Productivity Components

Group behavior manager

In collaboration with University of Sannio – Benevento (Italy)

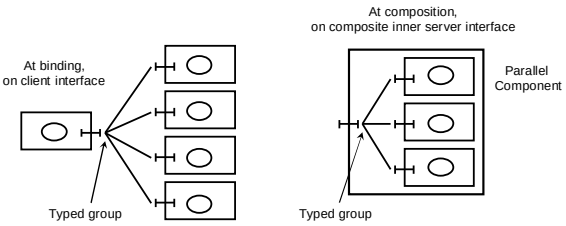
Definition of a group behavior
- Request mapping
- Parameters distribution
- Results gathering
- Synchronization semantic

Dynamic configuration and binding

IP Multicast library
- Bindings to multicast transport protocol
- TRAM, included in JRMS 1.1

Groups in Components

Implementation of the Fractal component model
Collective ports
Composite components



Sum up: MPI vs. ProActive OO SPMD

A simple communication model, with simple communication primitive(s):
- No RECEIVE but data flow synchronization
- Adaptive implementations are possible for:
  - // machines, Cluster, Desktop, etc.,
  - Physical network, LAN, WAN, and network conditions
  - Application behavior

Typed Method Calls:
--> Towards Components

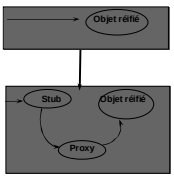
Reuse and composition:
- No main loop, but asynchronous calls to myself:
  ==> See details this afternoon in the Hands-On Session

1.3. ProActive architecture: a simple MOP

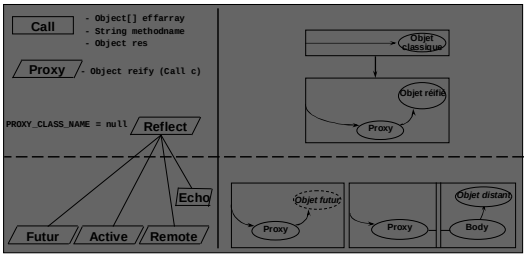
- MOP (Meta-Object Protocol)
  - Runtime reflection (for dynamicity)
  - New semantics for method and constructor calls
  - Same technique for Future implementation
  - Uses the Java Reflection API
- ProActive
  - Implemented on top of the MOP
  - Other models can be built on top of ProActive or on top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



The MOP - principle



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* :

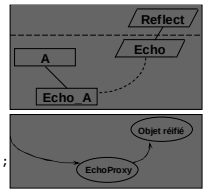
```
Vector v = (Vector) MOP.newInstance ( <name of class standard>,
                                      <class proxy name to use>,
                                      <parameters given to proxy>,
                                      <parameters of reified object constructor > );
```

- Or *object per object* :

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

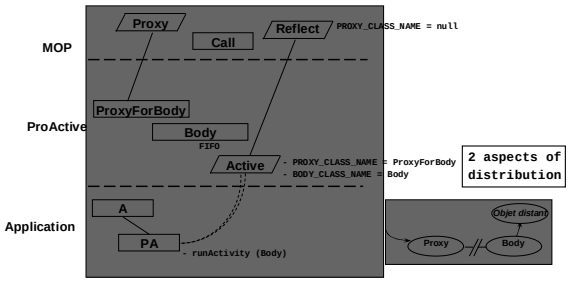

Example : EchoProxy

```
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"; }
```



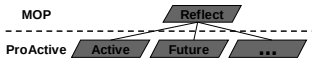

```
A a =(A)newInstance ("Echo_A",null,null);
A a =(A)newInstance ("A","EchoP.",null,null);
A a =(A)turnReified ( a, "EchoP.", null);
```


ProActive : implementation principle

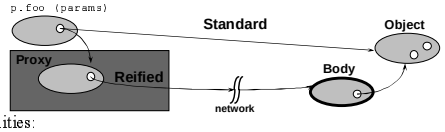


Proxy and Body

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



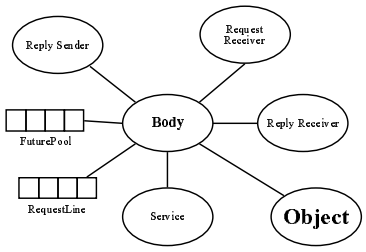
A proxy / body model



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

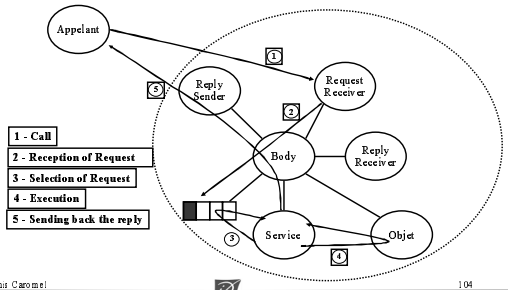


Composition d'un objet actif

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



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

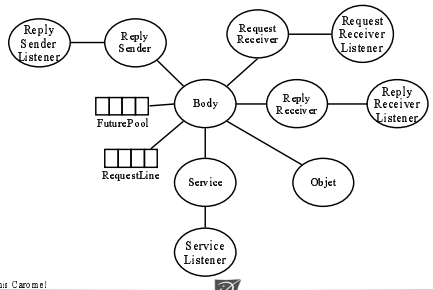
Listener - Modifier

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.

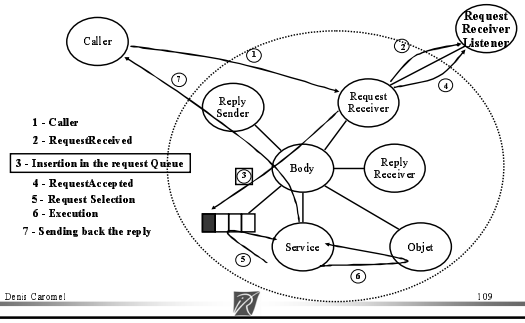
Application: debugging, monitoring, interactif Control of execution

Localization of listeners



Request Reception with a Listener

Used for IC2D implementation (see below)



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

2. ProActive : Migration of active objects

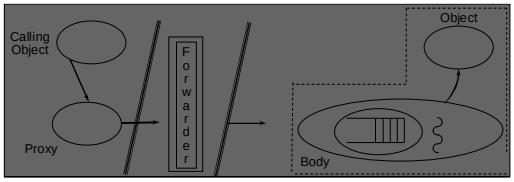
Migration is initiated by the active object itself through a primitive: migrateTo

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



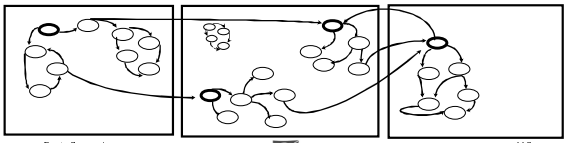
Migration is initiated by the active object through a request

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

2 Techniques : Forwarders or Centralized server

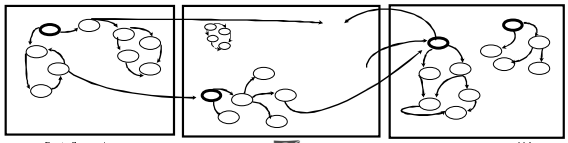
Principles and optimizations

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)



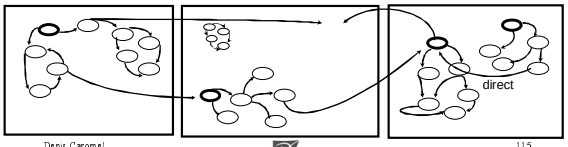
Principles and optimizations

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)



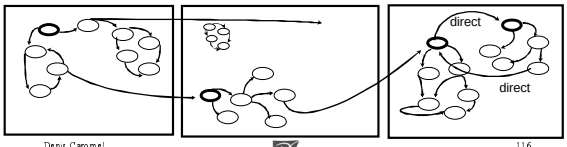
Principles and optimizations

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)



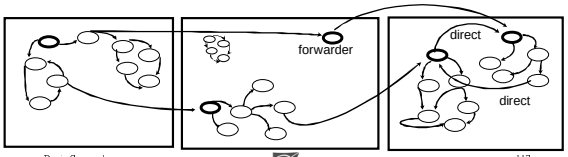
Principles and optimizations

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)



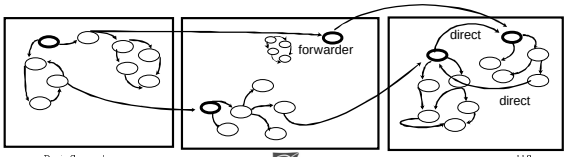
Principles and optimizations

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)



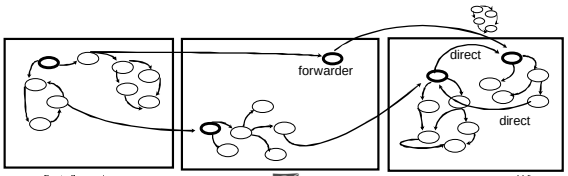
Principles and optimizations

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)



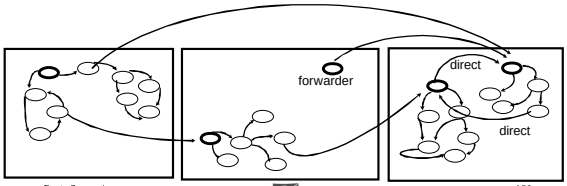
Principles and optimizations

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)



Principles 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)



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 runActive, 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 runActivity (Body myBody){
      while (… not end of itinerary …){
            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

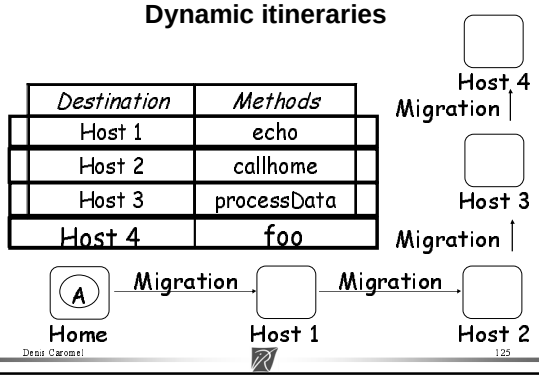
Itinerary : VMs to visit
- specification of an itinerary as a list of (site, method)
- automatic migration from one to another
- dynamic itinerary management (start, pause, resume, stop, modification, …)

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

Dynamic itineraries

Destination	Methods
Host 1	echo
Host 2	callhome
Host 3	processData
Host 4	foo



Communicating with mobile objects

Ensuring communication in presence of migration

Should be transparent (i.e. nothing in the application code)

Impact on performance should be limited or well known

ProActive provides 2 solutions to choose from at object creation

- Location Server
- Forwarders

also, it is easy to add new ones!

Forwarders

Migrating object leaves forwarder on current site

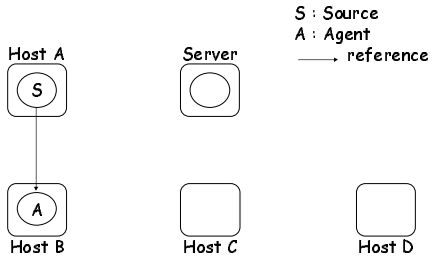
Forwarder is linked to object on remote site

- Possibly the mobile object
- Possibly another forwarder => a *forwarding chain* is built

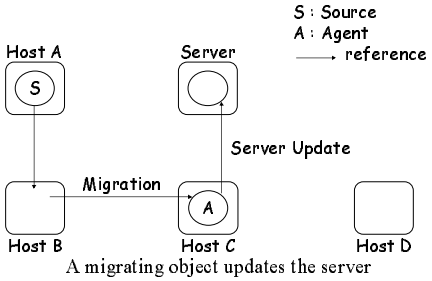
When receiving message, forwarder sends it to next hop

Upon successful communication, a tensioning takes place

Other Strategy: Centralized (location Server)

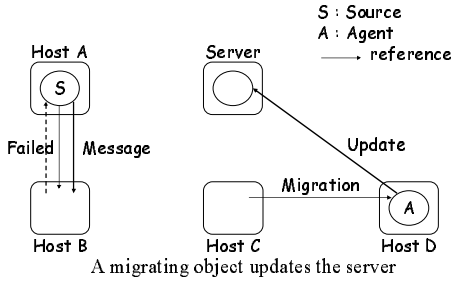


Centralized Strategy (2)



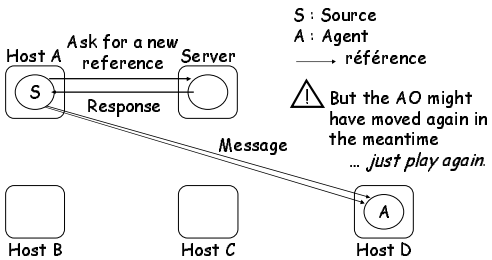
A migrating object updates the server

Centralized Strategy (3)



A migrating object updates the server

Centralized Strategy (4)



The source get a new reference from the server

Location Server vs Forwarder

Server

- No fault tolerance if single server
- Scaling is not straightforward
- Added work for the mobile object
- The agent can run away from messages

Forwarders

- Use resources even if not needed
- The forwarding chain is not fault tolerant
- An agent can be lost

What about the performance?

Impact on performance

Simple test:

- Measure the time for a simple call on a mobile object
- no parameters, no return value

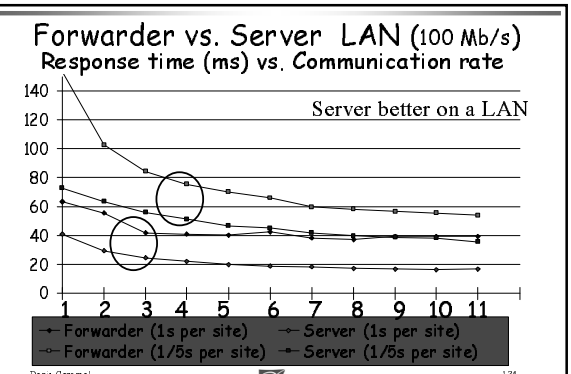
Application made of 2 objects: The source and the Agent

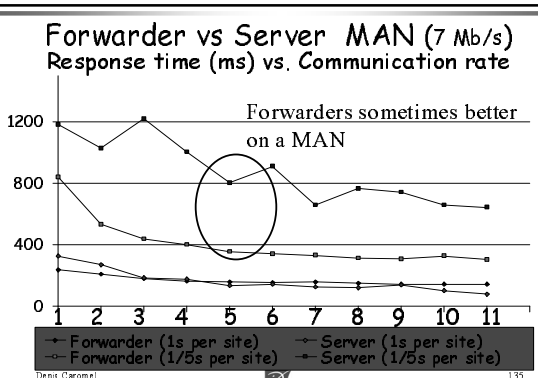
Source

- Does not migrate
- Wait an average time before calling the Agent (1/communication rate)

Agent

- Wait an average time on a site before migrating (1/migration rate)





On the cost of the communication

Server:

- The agent must call the server => the migration is longer
- Cost for the source:
  - Call to site where the agent was
  - Call to the server and wait for the reply
  - Call to the (maybe) correct location of the agent

Forwarder:

- The agent must create a forwarder (< to calling server)
- Cost for the source:
  - Follow the forwarding chain
  - Cost of the tensioning (1 communication)

Conclusion

Weak Migration of any active object

Communications using two schemes: server and forwarders

Current applications:

- Network Administration
- Desktop to Laptop

Perspective: Taking the best of the forwarders and the server

- Forwarder with limited lifetime
- Server as a backup solution

TTL-TTU mixed parameterized protocol

TTL: Time To Live + Updating Forwarder: **5 s.**

- After TTL, a forwarder is subject to self destruction
- Before terminating, it updates server(s) with last agent known location

TTU: Time To Update mobile AO:

- After TTU, AO will inform a localization server(s) of its current location

Dual TTU: first of two events:

- maxMigrationNb: the number of migrations without server update **10**
- maxTimeOnSite: the time already spent on the current site **5 s.**

Conclusion on Mobile Active Objects

AO = a good unit of Computational Mobility

Weak Migration OK (even for Load Balancing)

Both Actors and Servers

Ensuring communications: several implementation to choose from:

- Location Server
- Forwarders
- Mixed: based on TTL-TTU

Primitive + Higher-Level abstractions:

- migrateTo (location)
- onArrival, onDeparture
- Itinerary, etc.

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-

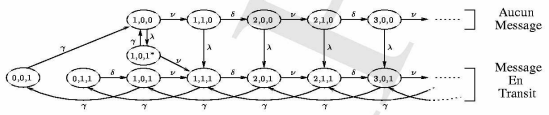


FIG. 4.2 – *États et transitions du mécanisme des répéteurs*

3 -

Départ	Transition	Arrivée	Description
(1,0,0)	ν	(1,1,0)	Début de migration
	λ	(1,0,1*)	Nouveau message généré
$(i,0,0)$ avec: $i \geq 2$	ν	$(i,1,0)$	Début de migration
	λ	$(i,0,1)$	Nouveau message généré
$(i,1,0)$ avec: $i \geq 1$	δ	$(i+1,0,0)$	Fin de migration
	λ	$(i,1,1)$	Nouveau message généré
(1,0,1*)	ν	(1,1,1)	Début de migration
	γ	(1,0,0)	Message a atteint l'agent
$(i,0,1)$ avec: $i \geq 1$	ν	$(i,1,1)$	Début de migration
	γ	$(i-1,0,1)$	Message a effectué un saut
$(i,1,1)$ avec: $i \geq 1$	δ	$(i+1,0,1)$	Fin de migration
	γ	$(i-1,1,1)$	Message a effectué un saut
(0,1,1)	δ	(1,0,1)	Fin de migration
(0,0,1)	γ	(1,0,0)	Message a atteint la source

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

4 -

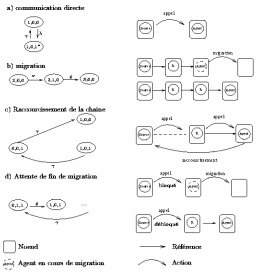


FIG. 4.3 – *Détails des transitions du diagramme des répéteurs*

5 -

Analyse Markovienne du serveur centralisé

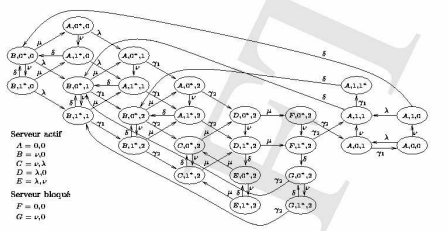


FIG. 4.6 – *État du système et taux de transitions dans le mécanisme du serveur.*

6 -

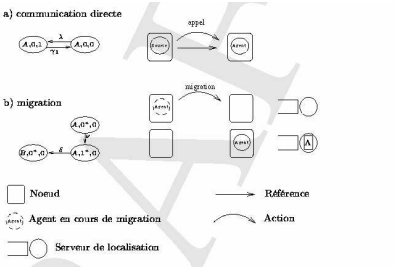


FIG. 4.7 – *Détails des transitions du modèle du serveur - source et agent*

7 -

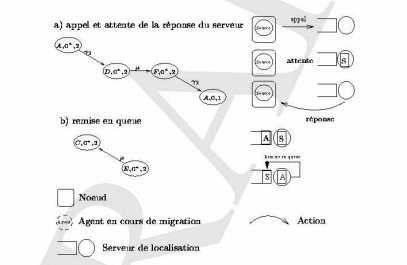


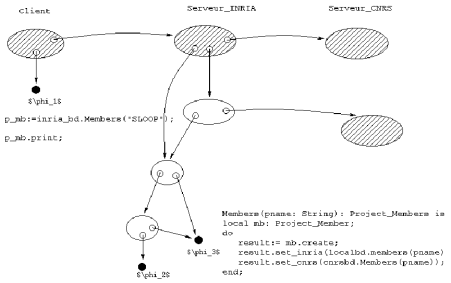
FIG. 4.8 – *Détails des transitions du modèle du serveur - le serveur*

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



Transparent Future transmissions (Request,Reply)

Parallel, Distributed, Hierarchical

3. Components

for the Grid

Composing

3.1 Component Introduction

Component - What is it ?

A Component = a unit of Composition and Deployment

From Objects (Classes) to Components:

- Objects:
  - Programming in the small
  - Composition by programming (Inheritance, Instantiation/Aggregation)
- Components:
  - Building software in the large
  - Tools for assembling and configuring the execution

Component = a module (80s!) but subject to:

- Configuration (variation on Non Functional Properties)
- Instantiation, life Cycle management

To be deployed on various platforms (some portability)

Characteristics -- How ?

How it works --- Common characteristics

- A standardized way to describe a component:
  - a specification of what a component does:
    - Provide (Interfaces, Properties to be configured)
    - Require (services, etc.)
    - Accept as parameterization
- Usually dynamic discovery and use of components:
  - Auto-description (Explicit information: text or XML, reflection, etc.)
- Usually components come to life through several classes, objects
- Legacy code: OO code wrapper to build components from C, Fortran, etc.

My Definition of Software Components

A component in a given infrastructure is:

- a software module,
- with a standardized description of what it needs and provides,
- to be manipulated by tools for Composition and Deployment

A primitive example: JavaBeans
Graphical components in Java

Quite simple :

- a Java class (or several)
- a naming convention to identify **properties:**
  - method: `public T getX ()`
  - method: `public void setX ()`
  - an attribute: `private T X = <default value>;`
- a communication pattern: **Events, Source, Listeners**

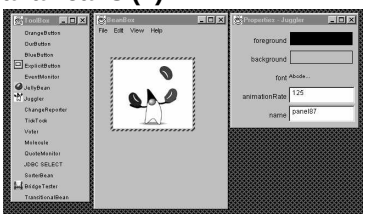
and ... a class is turned into a graphical component !

The Java introspection allows to discover dynamically the properties, and to configure them, assemble JB interactively

JavaBeans (2)
The BeanBox

So for JavaBeans:

Software module =
Java Class
Standardized description =
getX, setX, X,
listeners
Tools:
Composition = BeanBox
Deployment = JVM



Nothing very new (cf. NeXTStep Interface Builder),
but life made a bit easier with byte code and introspection

Deploying and Executing Components

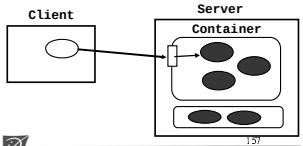
Components have to be configured on their Non-Functional Properties:

- Functional Properties, Calls (Def.):
  - Application level services a component provides (e.g. Balance, Saxpy)
- Non-Functional Properties, Calls (Def.):
  - The rest, mainly infrastructure services:
    - Transaction, Security, Persistence, Remote/Asynchronous Com., Migration, …
    - Start, Stop, Reconfiguration (location, bindings), etc.

so, Typical Infrastructure : Container for Isolation

Allows to manage and implement:

- the non-functional properties
- Life cycle of components



Enterprise Java Beans

A 3 tiers architecture (Interface, Treatment, DB), in Java

- Objectives: ease development + deployment + execution
- Java portability

A few concepts and definitions:

- EJB Home object:
  - management of life cycle: creation, lookup, destruction, etc.
- EJB Remote object:
  - Client view and remote reference
- EJB object (or Bean):
  - the component itself (Session Stateless or Statefull, Entity Bean)
- Functional Properties = Business Methods

Summary: Enterprise Java Beans

… for EJB:

Software module =

- Java Classes and Interfaces (Home, Remote, Beans, …)

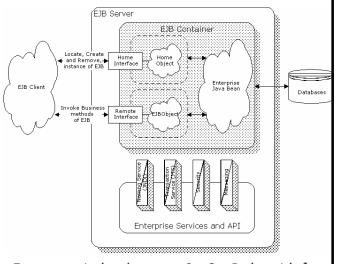
Only Provides (server), no Uses

Standardized description =

- a file with a standard format

Tools

- Composition = ? EJBrew ?
- Deployment = JVM + RMI, JTS, + Generators + EJB Servers



From www.tripod.com , G. S. Raj article

Components in Windows .Net

.Net basics:

- A VM designed for several languages (C, C++, VB, + others)
  - CLR (Common Language Runtime)
  - CIL (Common Intermediate Language, MSIL) wider than ByteCode
    - Boxing/Unboxing (value type <–> object), etc.
- A new language: C#
- An interactive tool (Visual Studio) to manipulate the "components"

A key choice: Extraction of description from program code

- C# introduces language constructions for component information:
  - Properties
  - Attributes
  - XML tags in source code (in Attributes)

Components in Windows .Net (2)

Example of Attributes, and Properties in C#:

```
[HelpUrl ("http://someUrl/Docs/SomeClass")]
class SomeClass {
private string caption;
public string Caption {
   get { return caption; }
   set { caption = value;
         Repaint (); }
}
```

An attribute: HelpUrl

It is actually a user define class (derive from Syst.Att.)

Attribute exists at RT.

A Property: Caption

JavaBeans in a language

Also: Indexed properties

Components for Web program. : WSDL (Web Services Description Lang.)

- WSDL (Def. of Web callable methods) + Directories +
- SOAP as wire format + Classes with Attributes and properties,

Components in Windows .Net

Components characteristics:

Software module =

- Classes and Interfaces in various languages, but focus on C#

Standardized description =

- Still the COM, DCOM interfaces
- Extraction of Attributes, Properties from source code!
- WSDL

Tools

- Composition = Visual Studio, etc.
- Deployment = Windows .NET CLR

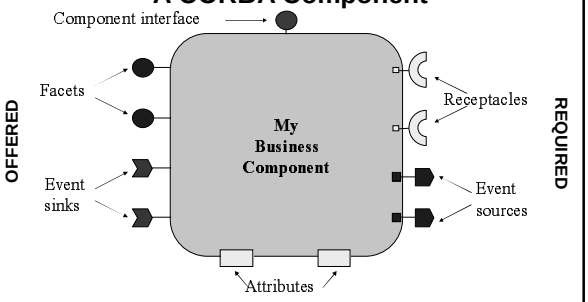
A Web Service: the instance of a component, … running...

Assembly of Components Corba 3 and CCM

CCM: Corba Components Model =

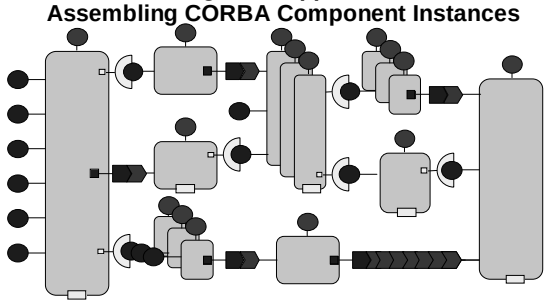
- EJB + a few things :
  - More types of Beans defined:
    - **Service**, Session, **Process**, Entity, …
  - Not bound to Java (Corba IDL)
  - Provides but also Uses :
    - Specification of the component needs, dependencies
    - "Client Interfaces"
  - A deployment model (ongoing at OMG)

A CORBA Component



Courtesy of Philippe Merle, Lille, OpenCCM platform

Building CCM Applications = Assembling CORBA Component Instances



Provide + Use, but flat assembly

Towards GRID Components

Parallel and Distributed:

--> Group Communications

Plus specificity :

- High performance
- Communication :
  - Important Bandwidth
  - Very High Latency
- Deployment complexity: **--> Abstractions**
  - Various remote execution tools (rsh, ssh, Globus, Web Services)
  - Various registry and lookup protocols (LDAP, RMI, WS, etc.)
  - Large variations in nodes being used (1 to 5000, … 200 000)
- Debugging, Monitoring, and Reconfiguring
  - Across the world ??

`High-Performance a specificity ? Not sure: an EJB component handling 1 000 000 of requests already needs High-Performance!`

`Networks grow faster than Procs`

`Techniques for hiding it`

3.2 ProActive Components

Component Orientedness

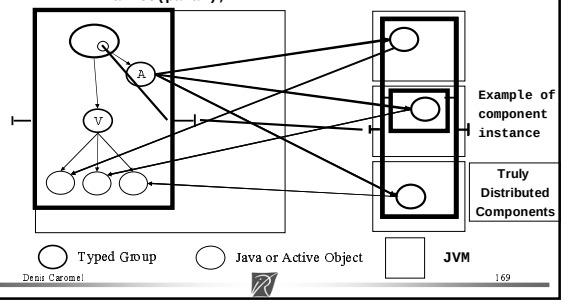
- Level 1: Instantiate - Deploy - Configure
  - Simple Pattern
  - Meta-information (file, XML, etc.) — JavaBeans, EJB
- Level 2: Assembly (flat)
  - Server and client interfaces — CCM
- Level 3: Hierarchic
  - Composite — Fractal, ProActive, …
- Level 4: Distributed + Reconfiguration
  - Binding, Inclusion, Location — ProActive + On going work

ProActive

Interactions / Communications:
Functional Calls: service, event, stream
Non-Functional: instantiate, deploy, start/stop, inner/outer, re-bind

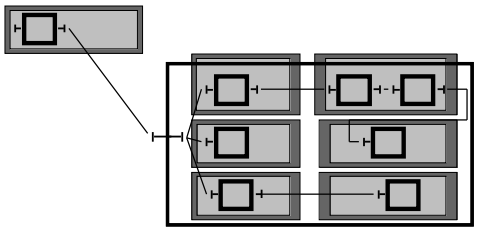
Objects to Distributed Components (1)

```
ComponentIdentity Cpt = newActiveComponent (params);
A a = Cpt … .getFcInterface ("interfaceName");
V v = a.foo(param);
```



Distributed Components (2)

1 component can be distributed over several hosts

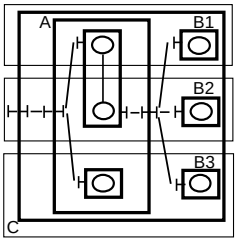


Components vs. Activity and JVMs

Activity **JVM** **Component**

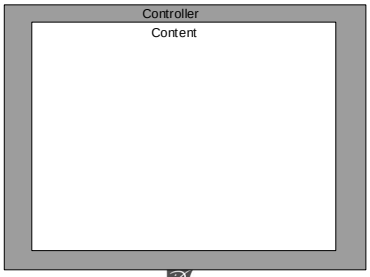
Cp. are rather orthogonal to activities and JVMs: contain activities, span across several JVMs

Components are a way to globally manipulate distributed, and running activities



The Fractal model: Hierarchical Component

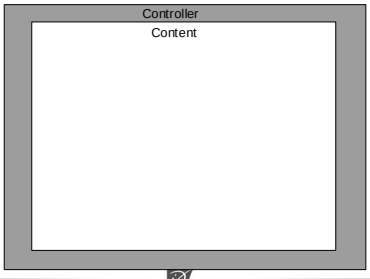
Defined by E. Bruneton, T. Coupaye, J.B. Stefani, INRIA & FT



ObjectWeb

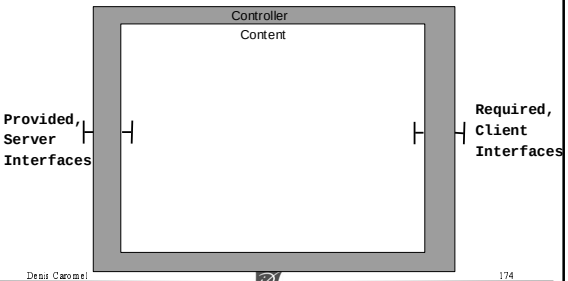
The Fractal model: Hierarchical Components

Common component model of the ObjectWeb consortium

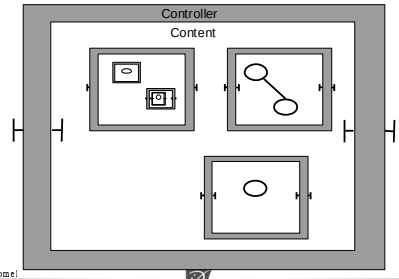


ObjectWeb

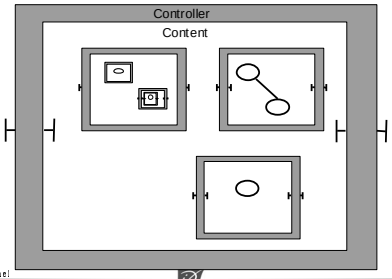
Interfaces = Provided and Required



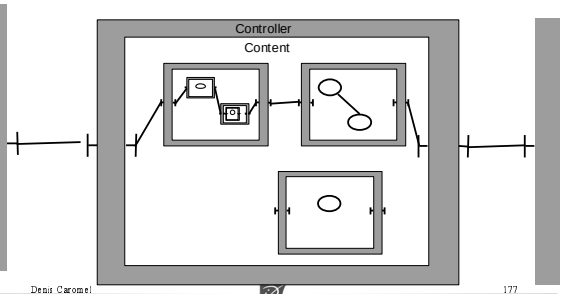
Hierarchical model : Composites encapsulate Primitives, Primitives encapsulate Code



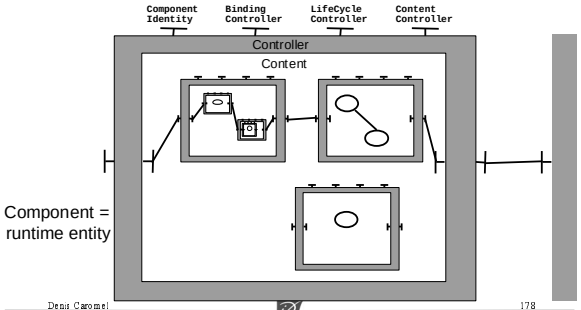
Binding = in an external file (XML ADL), Not in programs



Binding = in an external file (XML ADL), Not in programs



Controllers : non-functional properties

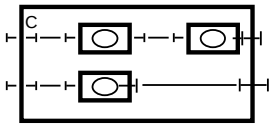


Component = runtime entity

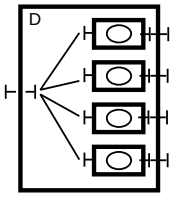
ProActive **Components for the GRID**

An activity, a process, … potentially in its own JVM

1. Primitive component



2. Composite component



3. Parallel and composite component

Composite: Hierarchical, and Distributed over machines

Parallel: Composite + Broadcast (group)

ProActive Component Definition

A component is:

- Formed from one (or several) Active Object
- Executing on one (or several) JVM
- Provides a set of server ports: Java Interfaces
- Uses a set of client ports: Java Attributes
- Point-to-point or Group communication between components

Hierarchical:

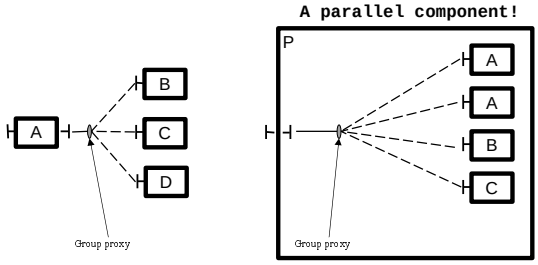
- Primitive component: define with Java code and a descriptor
- Composite component: composition of primitive + composite
- Parallel component: multicast of calls in composites

Descriptor:

- XML definition of primitive and composite (ADL)
- Virtual nodes capture the deployment capacities and needs

Virtual Node is a very important abstraction for GRID components

Groups in Components (1)



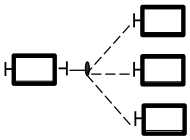
Broadcast at binding, on client interface

At composition, on composite inner server interface

Groups in Components (2)

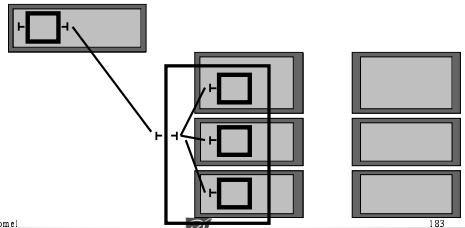
Typed group communications as the implementation of collective interfaces

2 modes : broadcast or scatter



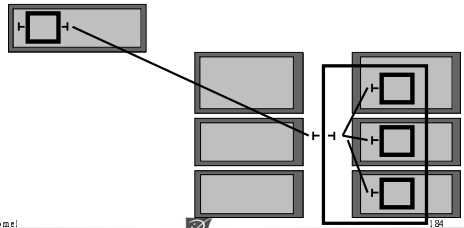
Migration Capability of composites

Migrate sets of components, including composites



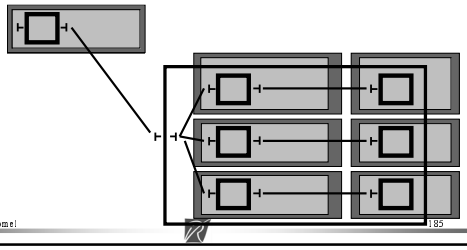
Migration Capability of composites

Migrate sets of components, including composites



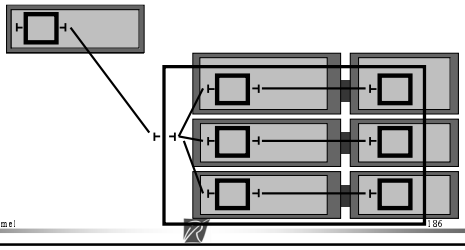
Co-allocation, Re-distribution

e.g. upon communication intensive phase



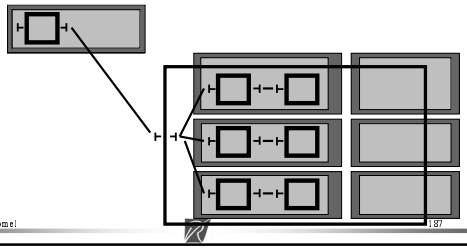
Co-allocation, Re-distribution

e.g. upon communication intensive phase



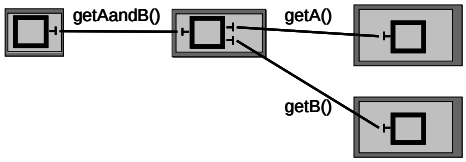
Co-allocation, Re-distribution

e.g. upon communication intensive phase



Functionalities : Without First Class Futures

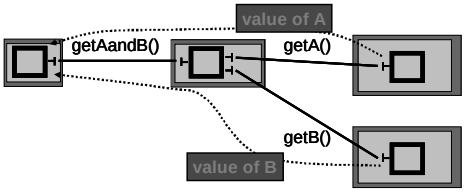
Or in the case of **Synchronous** method calls



Functionalities : With First Class Futures

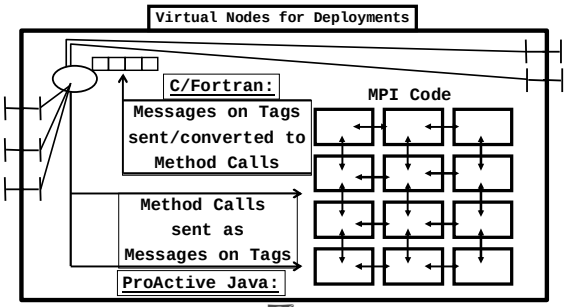
Non-blocking method calls

Example 2 : **Asynchronous** method calls with full-fledge **Wait-By-Necessity**



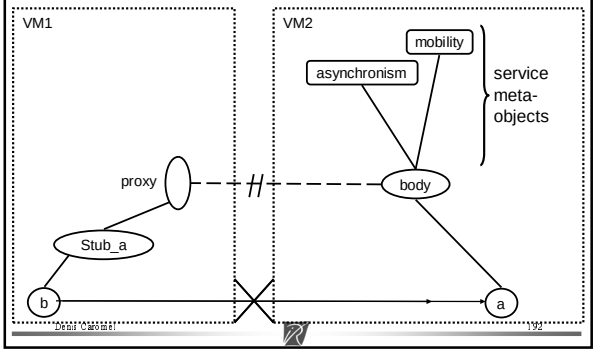
Assemblage are not blocked with Asynchrony + WbN

Wrapping Legacy MPI Components

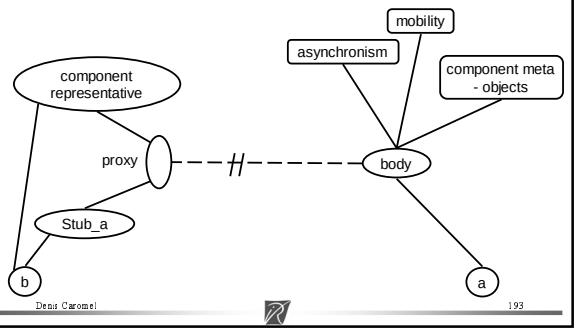


3.3 Component Implementation

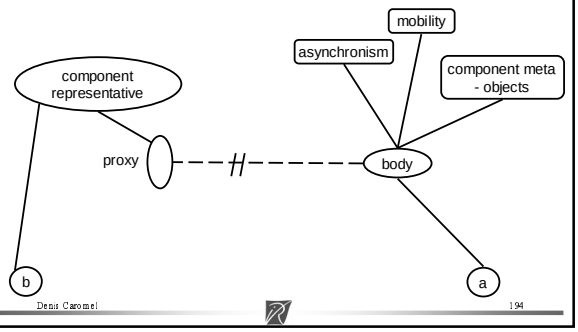
Architecture : based on the MOP



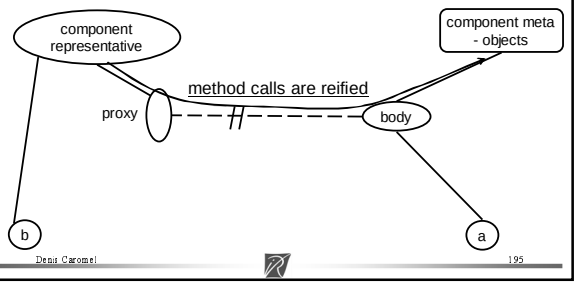
Architecture : component stubs



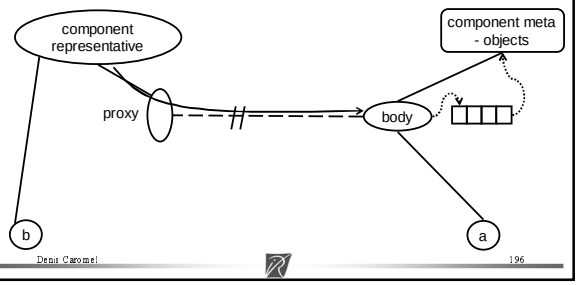
Architecture : component stubs



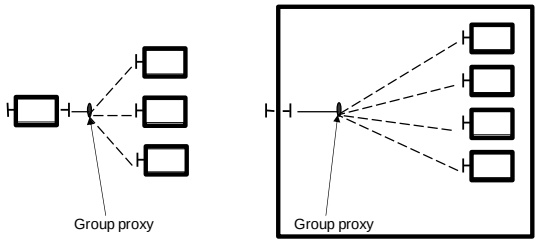
Architecture : communications



Architecture : request queue



Architecture : collective interfaces



➔ typed groups API
➔ Management of Collective Client Interface
➔ broadcast or scattering

Architecture : sum-up

Design
- component meta-object
- component stub (representative)
- Fractal interfaces / functional interface references
- 1 component = 1+ active object(s)

Communications
- reification
- request queue
- collective interfaces :
  - typed groups API

Using the ProActive implementation

Code

- Standard Fractal code

```
fractal.provider =
  org.objectweb.proactive.core.component.Fractive
```

- Implementation-specific parameters for instantiation

```
Component c = componentFactory.newFcInstance(
  Type type,
  ControllerDescription controllerDesc,
  ContentDescription contentDesc);
```

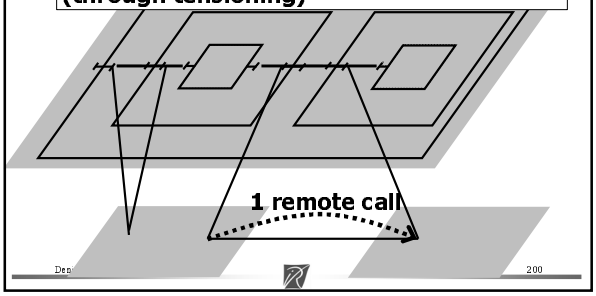
- Collective interfaces

```
I i1 = Fractive.createCollectiveClientInterface(
 String itfName, String itfSignature);
ProActiveGroup.getGroup(i1).add(serverItf);
i1.foo(toto); // scattered or broadcasted
```

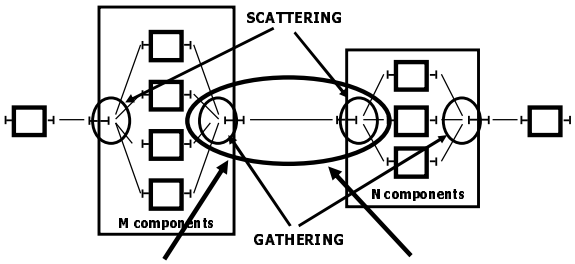
- no templates

On-going work : optimizations

Dynamic shortcuts for distributed bindings (through tensioning)



On-going : MxN communications
Control at binding points

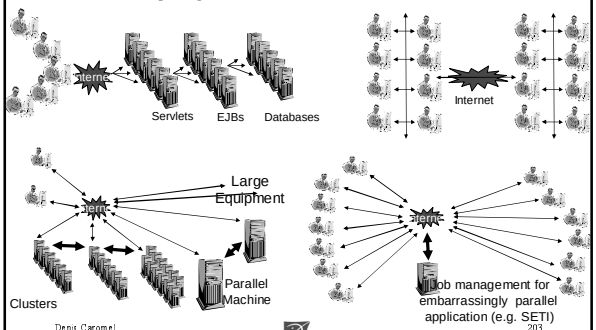


REDISTRIBUTION from M to N → also, Functional Code

4. Environment

Deploying

How to deploy on the Various Kinds of Grid



4.1 Abstract Deployment Model

Problem:

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

A key principle: Virtual Node (VN) + XML deployment file

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

Protocols and infrastructures:

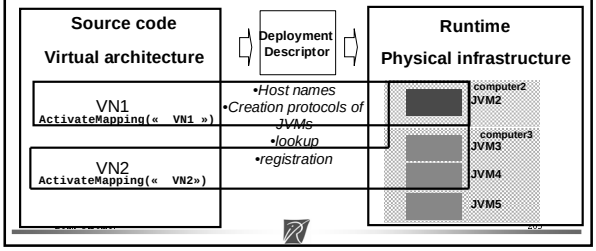
- Globus, ssh, rsh, LSF, PBS, ... Web Services, WSRF, ...

Abstract deployment model

Separates design from deployment infrastructure

Virtual nodes

Dynamic enactement of the deployment, from the application



Context

> **Grid**
- **Scalable**
- **Heterogeneous resources**
  - OS, CPU, Memory
  - Security policies: firewall, NAT, private IP addresses,…
  - Model,...

> **Painless deployment**
- **Based on well-known technologies**
  - Java, XML
- **Abstract Deployment model**

Abstract Deployment Model

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

A key principle: Virtual Node (VN) in XML deployment file
- **Abstract Away from source code:**
  - Machines names
  - Creation/Connection Protocols
  - Lookup and Registry Protocols
- **Interface with various protocols and infrastructures:**
  - Cluster: LSF, PBS, SGE , OAR and PRUN(custom protocols)
  - Intranet P2P, LAN: intranet protocols: rsh, rlogin, ssh
  - Grid: Globus, Web services, ssh, gsissh

XML Deployment files

> **Virtual Node (VN):**
- **Identified as a string name**
- **Used in program source**
- **Configured (mapped) in the 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**
- **Security**

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

Runtime structured entities: 1 VN --> n Nodes in m JVMs on k Hosts

Descriptors: Virtual Nodes in Programs

```
Descriptor pad = ProActive.getDescriptor("file:DeploymentDescriptor.xml");
VirtualNode vn = pad.activateMapping ("Dispatcher");
// Triggers the JVMs and Nodes creation
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

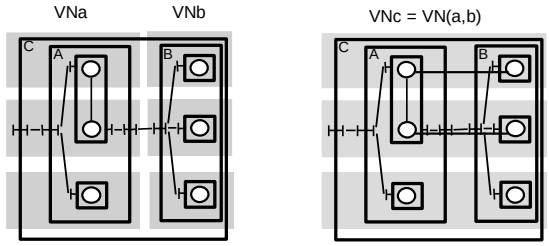
```
Descriptor pad = ProActive.getDescriptor("file:DeploymentDescriptor.xml");
VirtualNode vn = pad.activateMapping ("Dispatcher");
// Triggers the JVMs and Nodes creation
Node node = vn.getNode();
C3D c3d = ProActive.newActive("C3D", param, node);
log ( ... "created at: " + node.name() + node.JVM()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

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 >
...
```


XML Deployment (Not in source)



Separate or Co-allocation

Model and Tools for Grid Deployment

Abstract away machines, creation, registry, lookup protocols

- Use only **Virtual Nodes** in the source code
- Describe mapping of virtual nodes in XML Deployment Descriptors
- Interfaced with various protocols for creation/lookup: **rsh, ssh, Jini, LSF, PBS, Globus, OGSA...**

Unification of various deployment systems through ProActive runtimes

```
VirtualNodes:
  Dispatcher  <RegisterIn RMIregistry, Globus, Grid Service, ... >
  RendererSet
Mapping:
  Dispatcher --> DispatcherJVM
  RendererSet --> JVMset
JVMs:
  DispatcherJVM = <javapath ... /> <classpath.../>
                  sshProcess
  JVMset = <javapath ... /> <classpath.../>
                  GlobusProcess
Processes:
  sshProcess = <hostname dLunice.fr/> <login../>
  GlobusProcess = <hostname cluster.inria.fr/>
                  <gram port 2119/> <node 10/>
```

Working on: acquire ProActive runtimes from voluntary PCs (P2P computing) running a ProActive P2P infrastructure

Mapping Virtual Nodes: example (1)

Definitions and mapping

```
<virtualNodesDefinition>                          } Definition of
  <virtualNode name="Dispatcher"/>                } Virtual Nodes
</virtualNodesDefinition>

<map virtualNode="Dispatcher">                    }
  <jvmSet>                                        }
    <vmName value="Jvm1"/>                        } Mapping of
  </jvmSet>                                       } Virtual Nodes
</map>                                            }
<jvm name="Jvm1">                                 }
  <acquisition method="rmi"/>                     }
  <creation>                                      }
    <processReference refid="linuxJVM"/>          }
  </creation>                                     }
</jvm>
```


Mapping Virtual Nodes: example (2)

Definitions and mapping

```
<virtualNodesDefinition>                          } Definition of
  <virtualNode name="Jem3DNode"/>                 } Virtual Nodes
</virtualNodesDefinition>

<map virtualNode="Jem3DNode">                     }
  <jvmSet>                                        }
    <vmName value="clusterJvm"/>                  }
  </jvmSet>                                       }
</map>                                            } Mapping of
                                                  } Virtual Nodes
<jvm name="clusterJvm">                           }
  <acquisition method="rmi"/>                     }
  <creation>                                      }
    <processReference refid="clusterProcess"/>    }
  </creation>                                     }
</jvm>
```


Mapping Virtual Nodes: example (3)

Infrastructure informations

```
<processDefinition id="linuxJVM">
  <jvmProcess class="org.objectweb.proactive.core.process.JVMNodeProcess"/>
</processDefinition>

                    JVM on the current Host

<processDefinition id="rshProcess">
  <rshProcess class="org.objectweb.proactive.core.process.rsh.RSHJVMProcess"
              hostname="sea.inria.fr">
    <processReference refid="linuxJVM"/>
  </rshProcess>
</processDefinition>

                    JVM started using RSH
```

Mapping Virtual Nodes: example (4)

Infrastructure informations

```
<processDefinition id="singleJVM">
    <jvmProcess class="org.objectweb.proactive.core.process.JVMNodeProcess"/>
</processDefinition>

<processDefinition id="clusterProcess">
  <bsubProcess class="org.objectweb.proactive.core.process.lsf.LSFBSubProcess"
               hostname="cluster.inria.fr">
    <processReference refid="singleJVM"/>
    <bsubOption>
        <processor>12</processor>
    </bsubOption>
  </bsubProcess>
</processDefinition>
```

Definition of bsub process

Mapping Virtual Nodes: example (5)

Infrastructure informations

```
<processDefinition id="clusterProcess">
  <bsubProcess class="org.objectweb.proactive.core.process.lsf.LSFBSubProcess"
               hostname="cluster.inria.fr">
    <processReference refid="singleJVM"/>
    <bsubOption>
        <processor>12</processor>
    </bsubOption>
  </bsubProcess>
</processDefinition>
```

Definition of LSF deployment, … Globus

Mapping Virtual Nodes: example (6)

Infrastructure informations

```
<processDefinition id="linuxJVM">
    <jvmProcess class="org.objectweb.proactive.core.process.JVMNodeProcess"/>
</processDefinition>
```

JVM on the current Host

```
<processDefinition id="sshProcess">
    <sshProcess class="org.objectweb.proactive.core.process.ssh.SSHJVMProcess"
                hostname="sea.inria.fr">
        <processReference refid="linuxJVM"/>
    </sshProcess>
</processDefinition>
```

JVM started using SSH

Mapping Virtual Nodes: example (7)

Infrastructure informations

```
<processDefinition id="plugtestProcess">
  <sgeProcess class="org.objectweb.proactive.core.process.sge.SGEProcess"
              hostname="frontal">
      <processReference refid="localJVMprocess"/>
    <sgeOptions>
        <processor>12</processor>
    </sgeOptions>
  </sgeProcess>
</processDefinition>
```

Definition of SGE process

```
<processDefinition id="localJVMprocess">
    <jvmProcess class="org.objectweb.proactive.core.process.JVMNodeProcess"/>
</processDefinition>
```

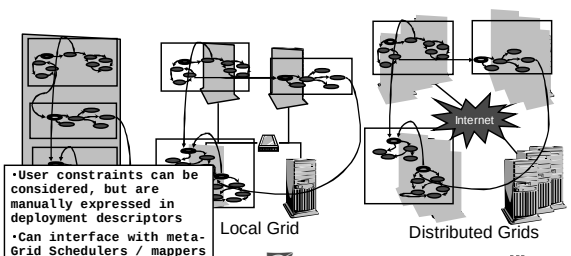
JVM created on the remote Hosts

Mapping Virtual Nodes (8): Mixed Protocol

Infrastructure informations

```
<processDefinition id="plugtestProcess">
    <sshProcess class="org.objectweb.proactive.core.process.ssh.SSHProcess"
                hostname="sea.inria.fr">
        <processReference refid="clusterProcess"/>
    </sshProcess>
</processDefinition>

<processDefinition id="clusterProcess">
  <pbsProcess class="org.objectweb.proactive.core.process.pbs.PBSProcess"
    <processReference refid="localJVMprocess"/>

  </pbsProcess>
</processDefinition>

<processDefinition id="localJVMprocess">
    <jvmProcess class="org.objectweb.proactive.core.process.JVMNodeProcess"/>
</processDefinition>
```


Same application, many deployments



- User constraints can be considered, but are manually expressed in deployment descriptors
- Can interface with meta-Grid Schedulers / mappers

IC2D

Interactive Control & Debug for Distribution

GUI for the GRID

4.2 IC2D

Interactive Control & Debug for Distribution

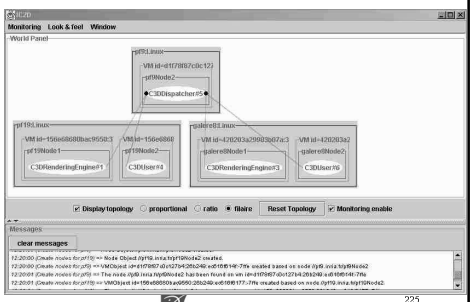
Features:

- 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: 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.)

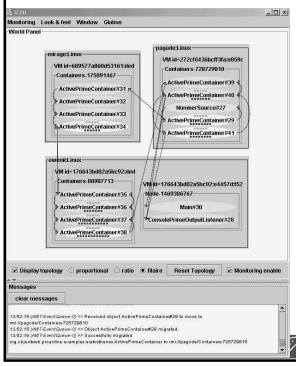
Control and Monitoring:
- Drag and Drop migration of executing tasks
- Creation of additional JVMs and nodes

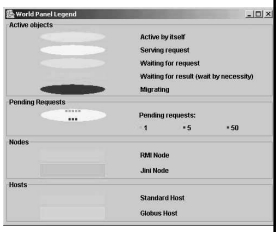
Job Management:
- JVM, AO per Job ID
- Textual visualisation, control (kill all, etc.)

IC2D: Basic features

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

IC2D: Interactive Control and Debugging of Distribution





With any ProActive application

Features:

Graphical and Textual visualization

Monitoring and Control

IC2D: Basic features

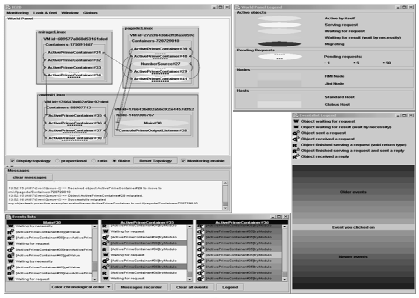
Graphical Visualization:

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

Textual Visualization:

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

IC2D: Logical Clocks



IC2D: Related Events

Events:

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

IC2D: Basic features

Graphical Visualization:

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

Textual Visualization:

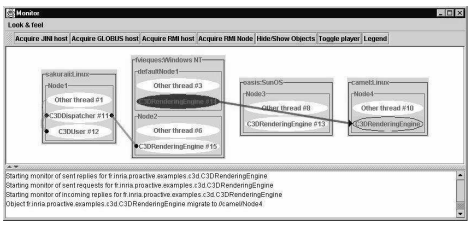
- Ordered list of messages, Logical Clock
- Status: waiting for a request or for a data
- Causal dependencies between messages
- Related events (corresponding send and receive, etc.)

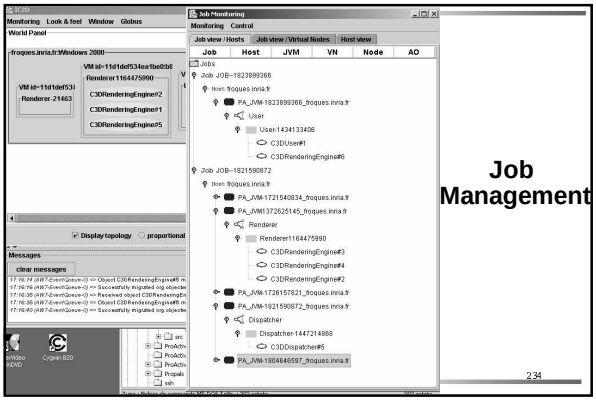
Control and Monitoring:

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

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

Drag-n-Drop tasks around the world



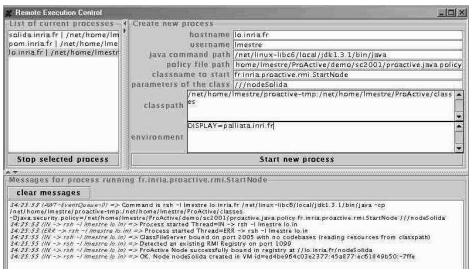


Job Management

IC2D: Dynamic change of Deployment New JVMs

Creation, Acquisition of new JVMs, and Nodes

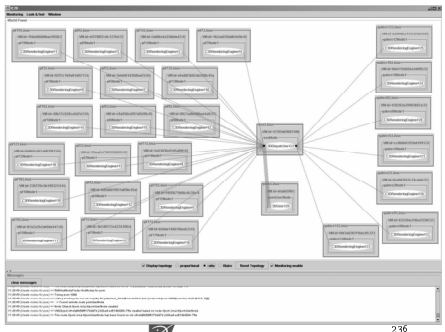
Protocols: rsh, ssh Globus, LSF



IC2D: Cluster Visualization

Visualization of 2 clusters (1Gbits links)

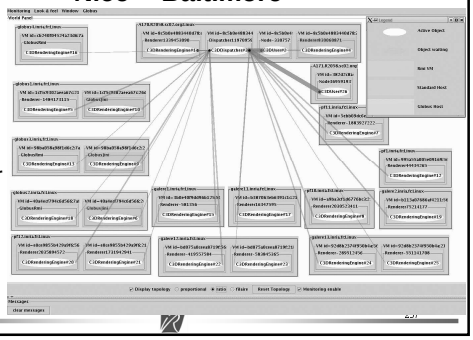
Featuring the current communications (proportional)



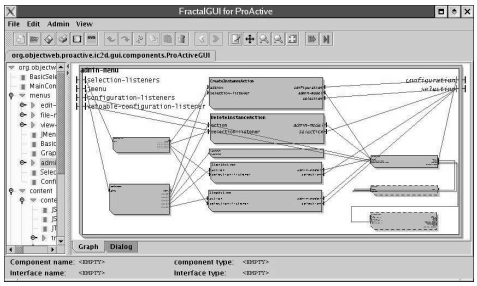
Monitoring of RMI, Globus, Jini, LSF cluster Nice -- Baltimore

ProActive IC2D:

Width of links proportional to the number of com-munications



On-going work : GUI for Components



5. ProActive Security

A key principle:

Specify security policies in the XML deployment, NOT IN SOURCE!

Security Models

Confidentiality models

- control on data access: Discr (HRU + Take-Grant) Mandat (BLP)
- control on data flow

Integrity models

- commercial domain : Biba

Availability models

- resource allocation problem

Access Control Models

Focus on access data control

Users are properly connected:

- Identification + Authentication OK
- no user can steal rights from another user.

Formal Rules to state if a given subject can access to a given object.

Access Control Models

Discretionary Access Control

- control is deduced from subject/group identity
- a subject can give his own rights to some body else: confidence in users !?

Mandatory Access Control

- control is deduced from security level of objects and subjects
- security levels can not be changed nor transfered by a subject, they are managed by a third party

HRU Model (Discr.)

Harrison, Ruzzo & Ullman

Access Control Matrix

- S in rows, O + S in columns

Rights:

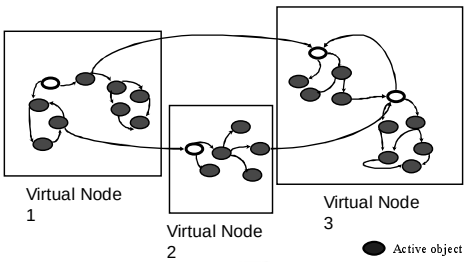
- own , read, write, add, execute, control

	O1	O2	O3	O4	O5	S1	S2	S3
S1	rw		r		w		x	
S2	rw	w			r			
S3	rw		w		rw	x		

Security : Objectives

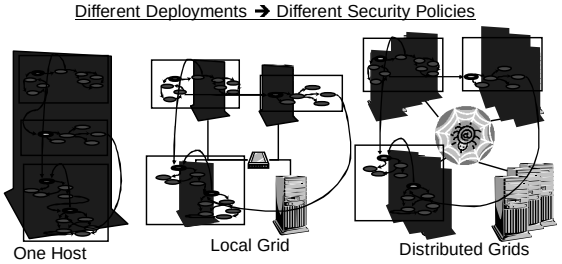
- Access control, communication privacy and integrity
- Security at user- and administrator-level
- Non functional security
- Declarative security language
- Dynamic when needed

A ProActive Application



Multiple Deployments Issues

Different Deployments → Different Security Policies



Security Issues, in Grid

Authentication of Computers, Users, and Applications

Creation, connection to, and monitoring of activities

Authentication, Integrity, Confidentiality (AIC) of communications

Various security policies (physical and logical organizations)

0 - SSH Tunneling

Static authentication and encryption

SSH Tunneling

A fact : overprotected clusters

- Firewalls prevent incoming connections
- Use of private addresses
- NAT, IP Address filtering, …

A consequence :

- Multi clustering is a NIGHTMARE

SSH Tunneling

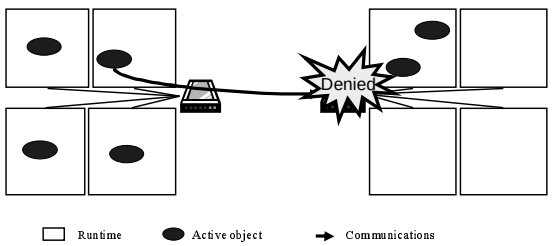
Context :

- SSH protocol : encrypt network traffic
- Administrators accept to open SSH port
- SSH provides encryption

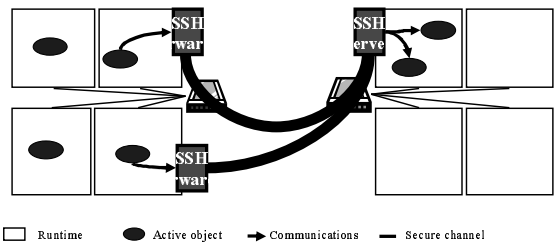
So :

- Let us do SSH tunneling !

Without SSH Tunneling



With SSH Tunneling



Abstract

Pros :
- VPN like
  - Static authentication
  - Static encryption of communication

Cons :
- No notion of security at application level
- Static feature

5.1 Issues for (Dynamic) Grid Security

- Authentication of Computers, Users, and Applications
- Creation, connection to, and monitoring of activities
- Authentication, Integrity and Confidentiality (AIC)
- Hierarchical domains
- Security Policies: Domain, User, Application
- Variation in Grid connectivity: LAN, Wireless, VPN, Internet
- Variation in deployment

Objectives

Goals :
- Authentication of Computers, Users, and Applications
- Communication authentication, privacy and integrity
- Security defined at user and administrator level
- Easy and adaptable configuration
- Support for current middlewares features : deployment, migration, group communication, components

Ways :
- Ubiquitous Security (Meta Object Protocol)
- Logical Security Architecture / Abstract Deployment
- Declarative Security Language

Public Key Infrastructure

- Elements of PKI
  - Certificate Authorities (CA)
  - Public/Private Key Pairs - Key management
  - X.509 Identity Certificates - Certificate management

Certificate Authority

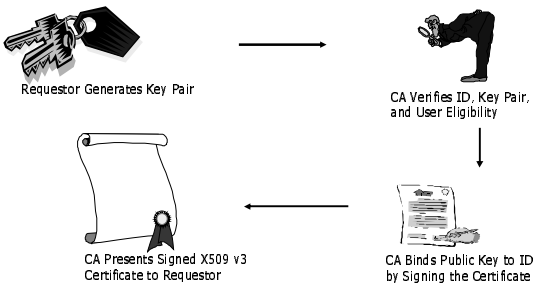
A trusted third party - must be a secure server

Signs and publishes X.509 Identity certificates

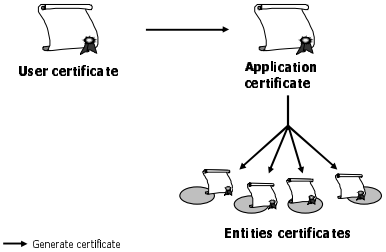
Revokes certificates and publishes a Certification Revocation List (CRL)

Many vendors
- OpenSSL - open source, very simple
- Netscape - free for limited number of certificates
- Entrust - Can be run by enterprise or by Entrust
- Verisign - Run by Verisign under contract to enterprise
- RSA Security - Keon servers

Authentication : X509 Certificate



Application authentication



X.509 Identity Certificates

Distinguished Name of user
- C=FR, O=INRIA , OU=Sophia, CN=Arnaud Contes

DN of Issuer
- C=FR, O=INRIA, CN=SOPHIA-CA

Validity dates:
- Not before <date>, Not after <date>

User's public key

V3- extensions

Signed by CA

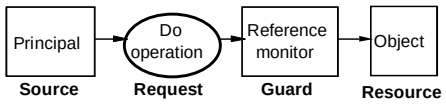
Authorization with ACLs

Access control lists (ACLs)
- An object *O* has an ACL that says: principal *P* may access *O*.
  - Lampson may read and write *O*
  - MSR may append to *O*

ACLs must use names for principals
- so that people can read them.

The Access Control Model

Guards control access to valued resources.



Application (1)

Composed of active objects

each active object has :
- an owner
- depends on an application

- Each application has a different identity even if launched by the same person, on the same machine, only certificate are important

- Active objects belongs to an application which belongs to an user which belongs to a domain.

Application (2)

1. User certificate ==> Application certificate
   - user private key used only for generating application certificate

2. Application certificate ==> active object certificate

5.2 ProActive Security Principles

Non-functional security
Hierarchical security domains
Dynamic policy negotiation
Certification chain to identify users, JVMs, objects
Application security policies set by deployment descriptors

ProActive Security: Key Features

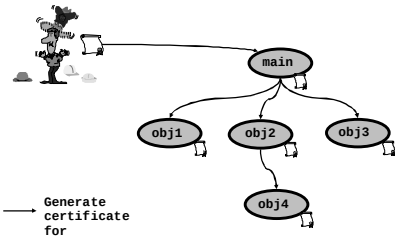
ProActive Security Features

- Authentication of users and applications (PKI X 509 certificates)
- Authentication, Integrity and Confidentiality of communications [A,I,C]
- In XML deployment files, Not In Source
- Mobility Aware
- Dynamically negotiated policies

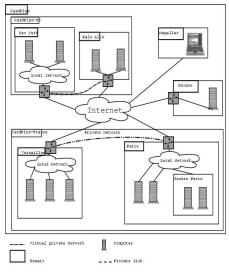
What's a secure ProActive application

- Composed of 'classic' active objects, no change in sources.
- Each application is different, even if launched by the same person, on the same machine.
- PKI Certification chain to identify users,applications,objects
  - User certificate => Application certificate =>active object certificate
  - user private key used only once for generating application certificate
- Security policy gets from deployment descriptors.

Certification Chain



Domains



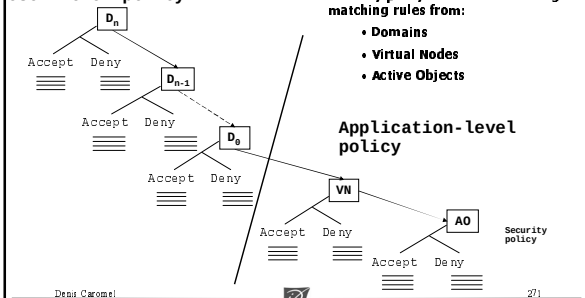
- Logical way to group many entities that have the same security needs.
- Domains are hierarchical.
- Sub-domains inherits parent's security policies.
- Default : Sub-domains cannot weaken parent's security policies.
- 'Can override' : a domain authorizes an entity to override its policies
- Find the first common domain if exists
- A domain has a policy server + a certification authority
- Dynamically configurable via SSL connections

Hierarchical Domains

Logical way to group many entities that have the same security needs.
Domains are hierarchical.
Sub-domains inherits parent's security policies.
Default : Sub-domains cannot weaken parent's security policies.
'Can override' : a domain authorizes an entity to override its policies
Find the first common domain if exists
Dynamically configurable via SSL connections

Multi-level Policies

Administrator + User-level policy



Security policy is defined according all matching rules from:

- Domains
- Virtual Nodes
- Active Objects

Application-level policy

Security Rule

```
Entities -> Entities : Interactions # Security Attributes
```

Entities :
- Domain
- User
- Virtual Node
- Object

Interactions :
- JVMCreation
- NodeCreation
- CodeLoading
- ObjectCreation
- ObjectMigration
- Request
- Reply
- Listing

Attributes :
- Authentication
- Integrity
- Confidentiality

Each attribute can be :
- Allowed
- Optional
- Disallowed

Combining Policies

Search for the most specific rule in each domain.

Retrieve all matching rules in the Domain hierarchy, the Virtual Node and the Active Object.

Compute policies according to security attributes.

Sender \ Receiver	Required (+)	Optional (?)	Disallowed (-)
Required (+)	+	+	invalid
Optional (?)	+	?	-
Disallowed (-)	invalid	-	-

Descriptor Security Model

A key principle:
- Specify security policies in the XML deployment, NOT IN SOURCE!

In program source:
- Virtual Node (VN, a string name):

In XML descriptors:
- List of policy rules
- Trusted Certification Authorities

Descriptors: Security

```
VirtualNodes
    vn1, vn2
SECURITY:
  VN [vn1] -> VN [vn2] : Q,P  #  [+A,?I,+C]
  VN [vn1] -> VN [vn2] : M  #  Forbidden
  VN [vn2] -> VN [vn1] : Q,P  #  [?A,?I,?C]
  VN [vn2] -> VN [vn1] : M  # Forbidden
Mapping:
  vn1 --> GridAComputers, GridBComputers
  vn2 --> GridAComputers
JVMs:
  /…/
```


ProActive Security Manager

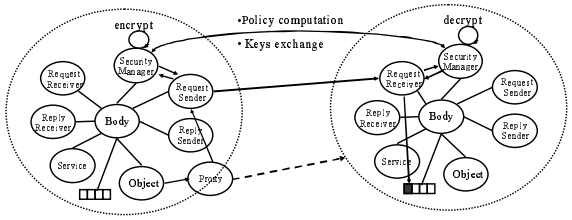
In charge of security for an active object

Retrieve, combine and negotiate policies

Negotiate session key,

Encrypt/decrypt messages

Request to an Active Object



5.3 Security Example

2 domaines GridA & gridB with security policies

- Domain [GridA] -> Domain [GridB] : Q,P,M # [+A,+I,+C]
- Domain [GridB] -> Domain [GridA] : Q,P,M # [+A,+I,+C]

Application :

- 2 Virtual Nodes (vn1,vn2)
- 2 Active objects

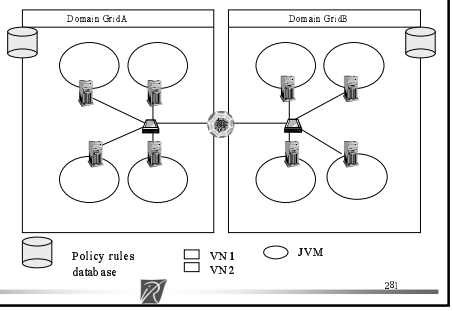
Descriptor with Security

VirtualNodes
vn1, vn2
SECURITY:
VN [vn1] -> VN [vn2] : Q,P # [+A,?I,+C]
VN [vn1] -> VN [vn2] : M # Forbidden
VN [vn2] -> VN [vn1] : Q,P # [?A,?I,?C]
VN [vn2] -> VN [vn1] : M # Forbidden
Mapping:
vn1 --> GridAComputers, GridBComputers
vn2 --> GridAComputers
JVMs:
/…/

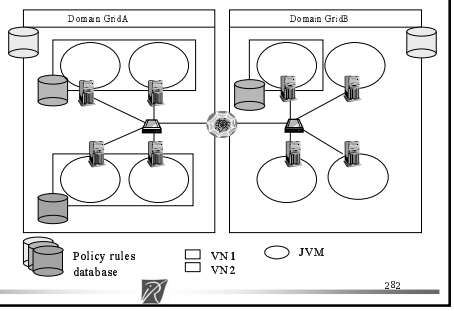
Example: std. code, no security

```
/…/
proActiveDescriptor.activateMappings();
vn1 = proActiveDescriptor.getVirtualNode("vm1");
vn2 = proActiveDescriptor.getVirtualNode("vm2");
/…/
Flower rose = (Flower) ProActive.newActive(Flower.class,new
                        Object[]{« Rose »}, vn1.getNode()};
Flower daliah = (Flower) ProActive.newActive(Flower.class,new
                        Object[]{« Daliah »}, vn2.getNode()};
/* next VN1 node inside the same domain */
rose.migrateTo(vn1);
/* communication inside the same domain */
rose.sayHelloTo(daliah);
/* other virtual node, forbidden */
rose.migrateTo(vn2);
/* next VN1 Node, other domain */
rose.migrateTo(vn1);
/* communication with another domain */
rose.sayHelloTo(daliah);
```

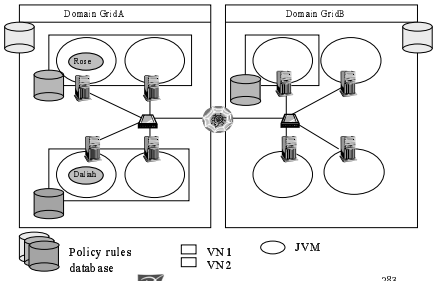

Example



Example

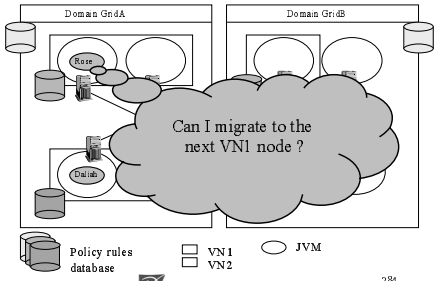


Example



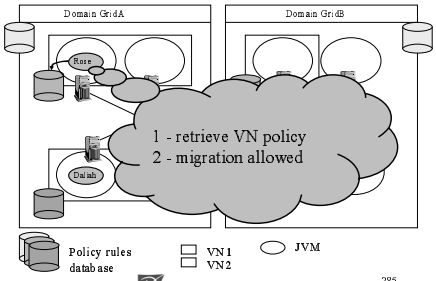
Example

Migration : - same VN - same domain



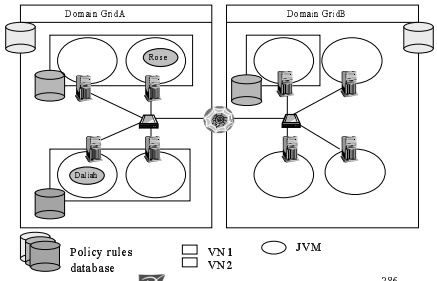
Example

Migration : - same VN - same domain



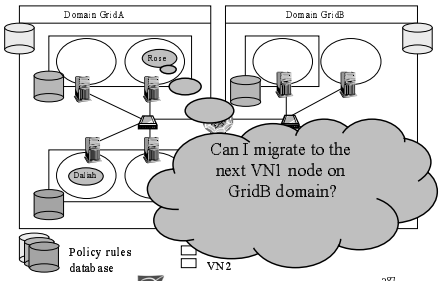
Example

Migration : - same VN - same domain



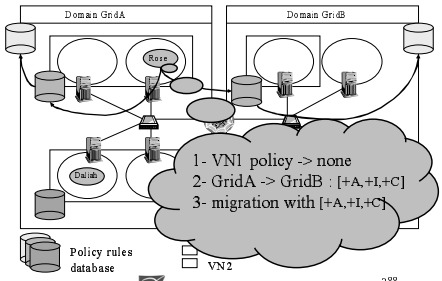
Example

Migration : - same VN - other domain



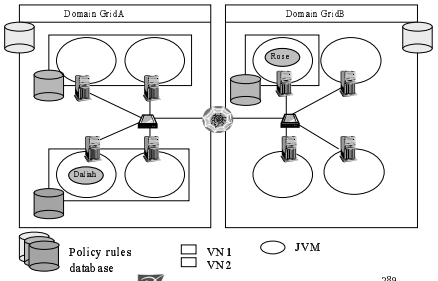
Example

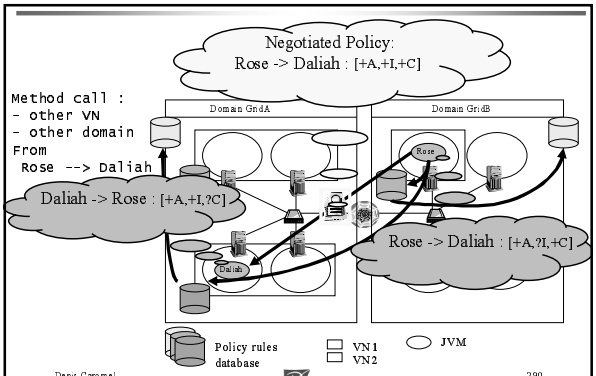
Migration : - same VN - other domain



Example

Migration :
- same VN
- other domain





Adaptive Security

Dynamic setting of the security attributes

Dynamic negotiation between different:
- Domain and Sub-domain
- Virtual Nodes
- Active Objects

on JVMs on different Machines

Security summary

ProActive Security Features
- Authentication of users and applications (PKI X 509 certificate)
- Authentication, Integrity and Confidentiality of communications
- In XML deployment files, Not In Source
- Security model for mobile applications
- Dynamically negotiated policies, non-functional security
- Logical security representation : security is easily adaptable to the deployment

Perspectives:
- Group communication

6. ProActive Applications

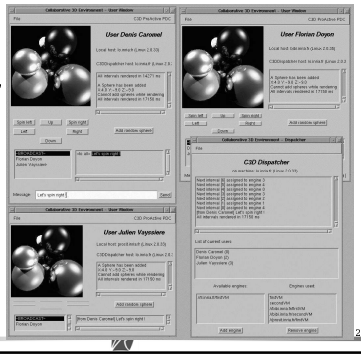
DEMO: Applis with the IC2D monitor

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

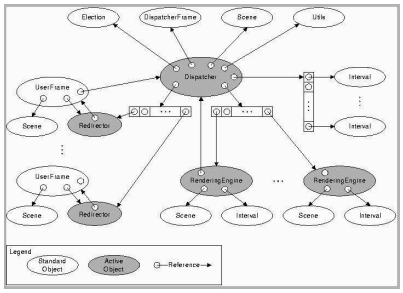
IC2D: Interactive Control & Debug for Distribution
work with any ProActive application
Features:
Graphical and Textual visualization
Monitoring and Control

C3D: distributed-//-collaborative

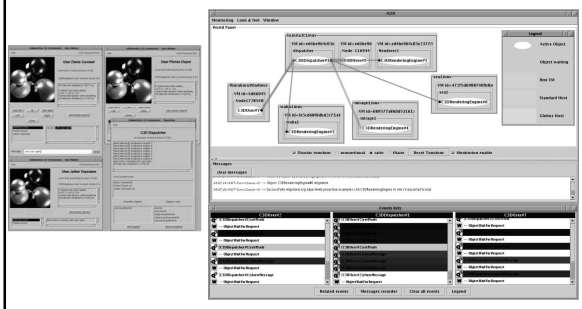
Collaborative 3D, Rendering in //, Application mobility



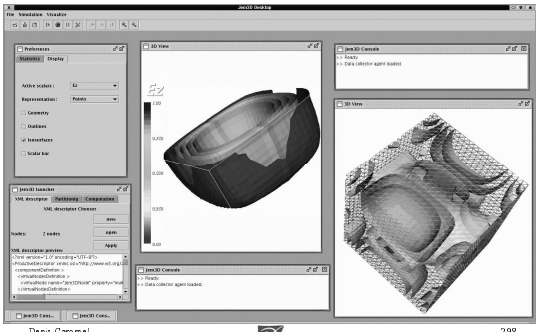
C3D: Object Diagram



C3D Monitoring: graphical and textual com.



Jem3D



JEM 3D : Java 3D Electromagnetism

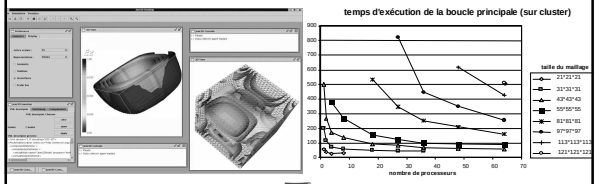
together with Said El Kasmi, Stéphane Lanteri (caiman)

Maxwell 3D equation solver, Finite Volume Method (FVM)

Pre-existing Fortran MPI version: EM3D (CAIMAN team @ INRIA)

Up to 294 machines at the same time (Intranet and cluster)

Large data sets: 150x150x150 (100 million facets)



Jem3D: Overall Objectives

3D Electromagnetic Application:

- Visualize the radar reflection of a plane, medicine on head, etc.
- Pre-existing Fortran MPI version: EM3D

Jem3D:

- Sequential object-oriented design, modular and extensible (in Java)
- Sequential version can be smoothly distributed:
  --> keeping structuring and object abstractions
- Efficient distributed version, large domains, Grid env.

Jem3D: Geometry definition

A Generic Numerical Method: Finite Volume Method (FVM)
(vs. Finite Element Methods)

- Calculation of unknowns as average of Control Volume
  (vs. Vertices of the mesh for FEM)
- Valid on structured, unstructured, or hybrid meshes

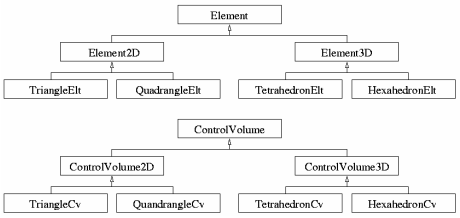
Computation:

- a flux balance through the boundary of Control Volume (M, E, EM, loop)

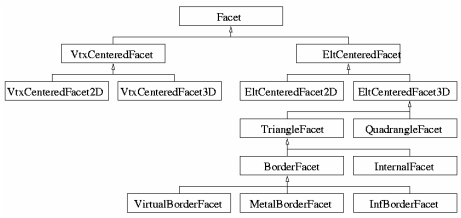
Example, and benchmarks here:

- Control Volume = Tetrahedron
- Facet = Triangle

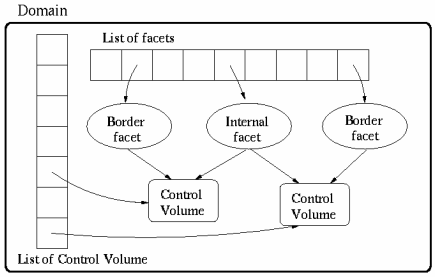
Jem3D: Control Volume in 2D and 3D



Jem3D: Facets in 2D and 3D

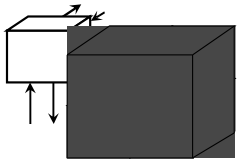


JEM 3D : Architecture of the sequential version

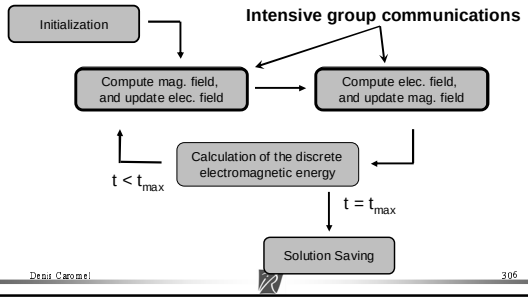


Communication based on groups

Groups allow sub-domains communication

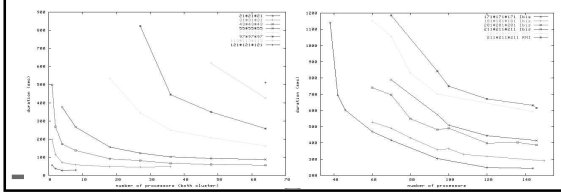


Algorithm of Jem3D

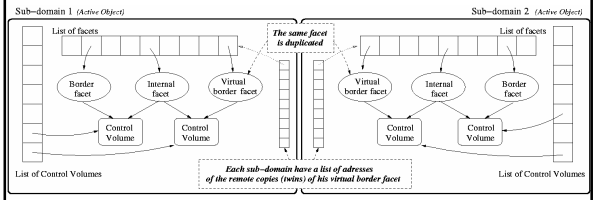


Jem3D

Computation of the propagation of an electromagnetic wave in a cubic metallic cavity (standard test case)

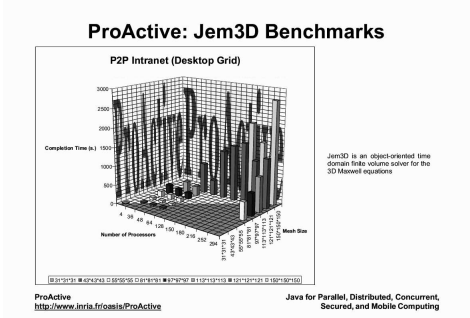


JEM 3D : Architecture of the ProActive distributed version

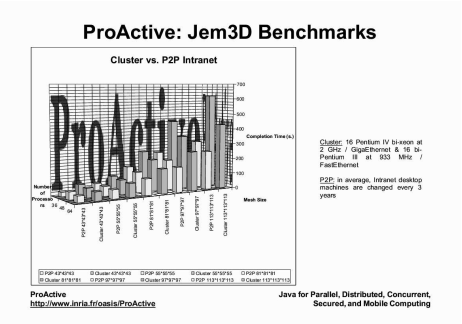


An Object-Oriented SPMD program

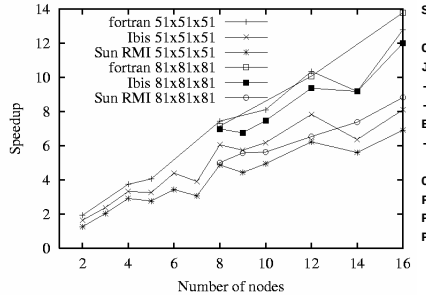
JEM 3D : Benchmarks on desktop machines



JEM 3D : Cluster vs. Intranet P2P



JEM 3D : Recent Benchmarks



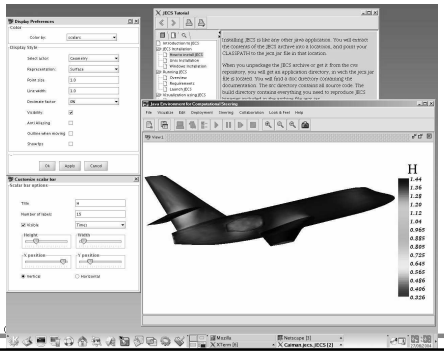
Seq. Java/Fortran: 2

Comparison:
Jem3D over
- ProActive/RMI Sun
- ProActive/RMI Ibis
Em3D in
- Fortran/MPI

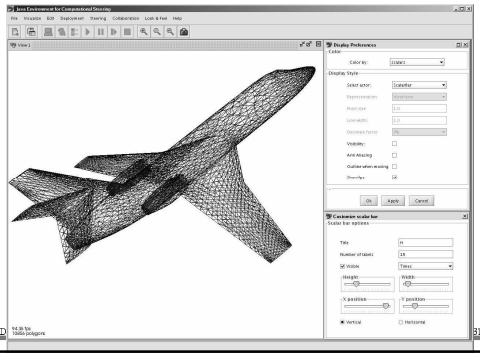
On 16 machines:
Fortran: 13.8
ProActive/Ibis: 12
ProActive/RMI: 8.8

Grid experiment on 5 clusters (DAS 2): Speed up of 100 on 150 machines

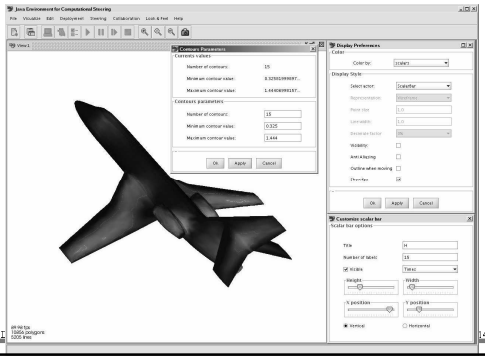
JECS : A Generic Version of Jem3D



JECS : A Generic Version of Jem3D



JECS : A Generic Version of Jem3D



Beating Fortran ?

Current status:

- Sequential Java vs. Fortran code: 2 times slower
- Large data sets in Java ProActive: 150x150x150 (100 million facets)
- Large number of machines: up to 294 machines in Desktop P2P
- Speed up on **16 machines:** - Fortran: 13.8
  - ProActive/Ibis: 12
  - ProActive/RMI: 8.8

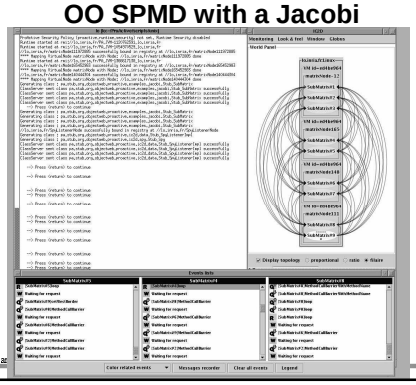
Grid on 5 clusters (DAS 2): Speed up of 100 on 150 machines

Fortran: no more than 40 proc. ...

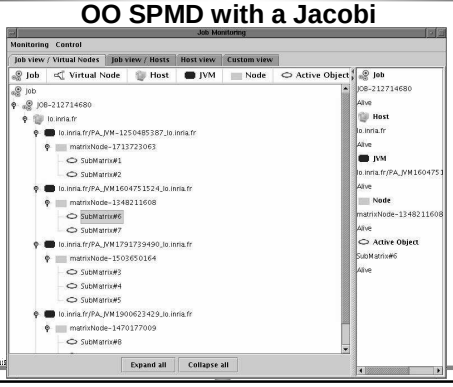
Beating Fortran MPI with Java ProActive? X/40 (14/16) = 2X/ n (100/150)

Yes, starting at 105 machines !

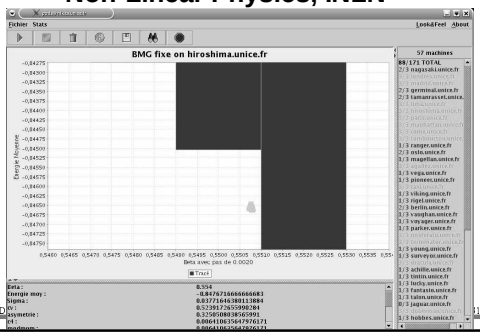
OO SPMD with a Jacobi



OO SPMD with a Jacobi



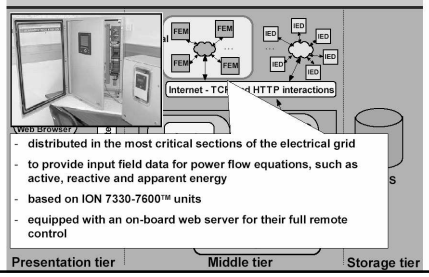
Monte Carlo Simulations, Non-Linear Physics, INLN



Electric Network Planning, E. Zimeo et al., Benevento (Naples), Italy

On-line Power Systems Security Analysis (OPSSA)

A network of field power meters (FEMs)

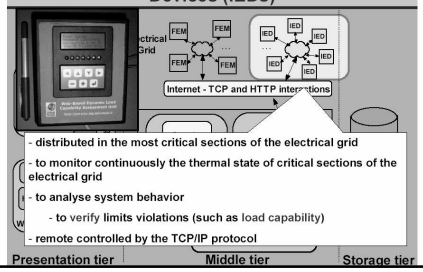


- distributed in the most critical sections of the electrical grid
- to provide input field data for power flow equations, such as active, reactive and apparent energy
- based on ION 7330-7600™ units
- equipped with an on-board web server for their full remote control

Electric Network Planning, E. Zimeo et al., Benevento (Naples), Italy

On-line Power Systems Security Analysis (OPSSA)

A network of distributed Intelligent Electronic Devices (IEDs)



- distributed in the most critical sections of the electrical grid
- to monitor continuously the thermal state of critical sections of the electrical grid
- to analyse system behavior
  - to verify limits violations (such as load capability)
- remote controlled by the TCP/IP protocol

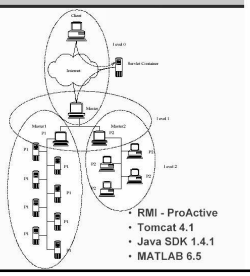
Electric Network Planning, E. Zimeo et al., Benevento (Naples), Italy

On-line Power Systems Security Analysis (OPSSA)

Software platform evaluation on a testbed 1/2

- Standard IEEE 118-nodes test network
  - electrical network description is local to each computational resource
- The experiments refer to 186 contingencies
- A COW with P1 proc.
  - P1 = Pentium II 350 MHz
- A NOW with P2 proc.
  - P2 = Pentium IV 2 GHz

$$\begin{cases} n_1 = n_2 = \ldots = n_6 = 6 & P_1 \\ n_1 = n_2 = \ldots = n_4 = 35 & P_2 \end{cases}$$



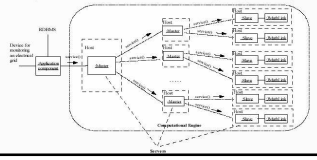
- RMI - ProActive
- Tomcat 4.1
- Java SDK 1.4.1
- MATLAB 6.5

Electric Network Planning, E. Zimeo et al., Benevento (Naples), Italy

On-line Power Systems Security Analysis (OPSSA)

ProActive implementation of HM/S p.

- Each Server
  - is defined as an active object (that can be dynamically created)
  - is allocated on a different computational resource (dynamically)
  - its method `service()` has to be asynchronously and remotely invoked by the client.
  - the asynchronous invocation is directly supported by ProActive

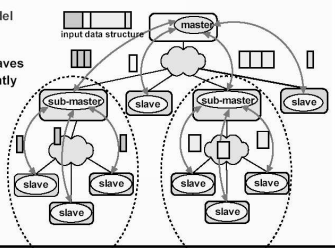


Electric Network Planning, E. Zimeo et al., Benevento (Naples), Italy

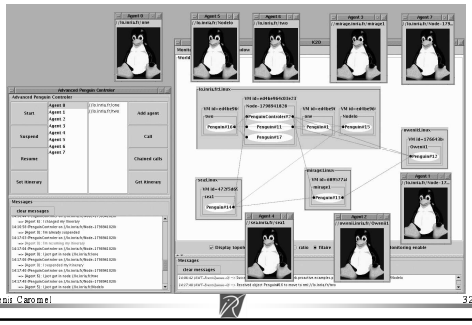
On-line Power Systems Security Analysis (OPSSA)

The hierachical m/s model 1/3

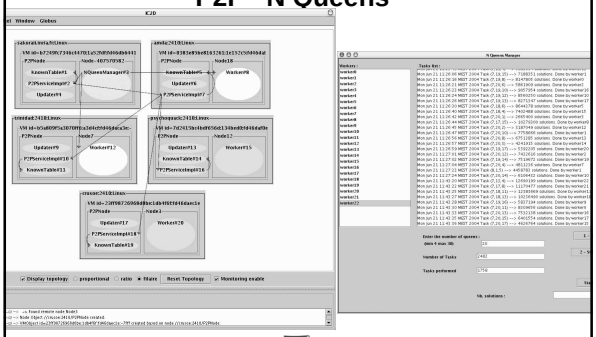
- hierarchical master/slave model
  - object-level parallelism
  - master and slaves are transparently created



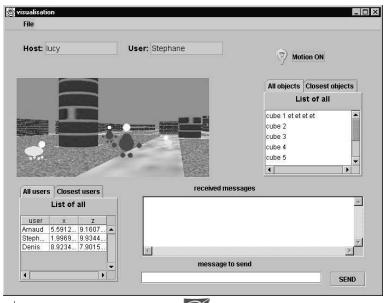
Mobile Application executing on 7 JVMs



P2P N Queens



DIVA: Distributed Interactive Virtual World



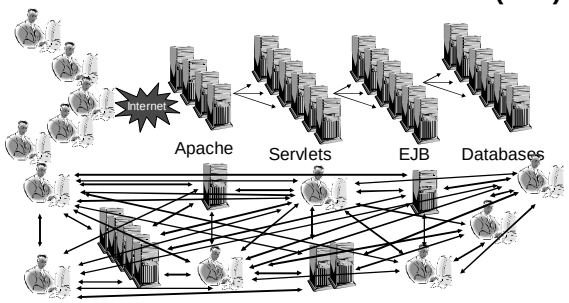
Component Applications

Put here:
C3D version,
Code Coupling EADS

7. Perspective:

Real P2P

Architectures: Server to Peer-To-Peer (P2P)



SOA: Service Oriented Architectures

Pure P2P: Definition

Only PEERs, no above everything, top level, server(s)

Every peer is, somehow, also a server

No master ... No slave !

System get organized dynamically, without static configuration

Coherent, desired behavior, dynamically emerges

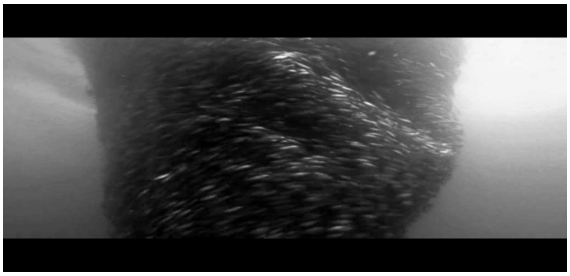
P2P Examples (1)

P2P can be difficult: need to be fault-tolerant, self healing

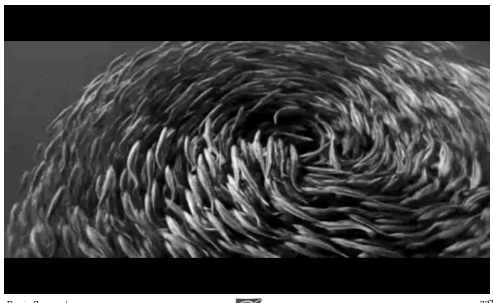
Not a P2P system

Neither a P2P system

P2P Examples (2)

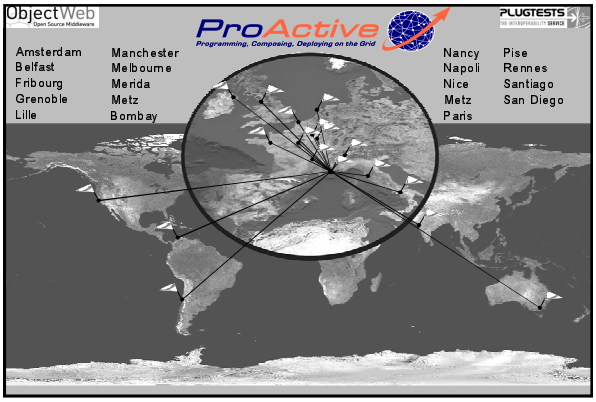


P2P Examples (2bis)



A P2P system at work

8. ProActive User Group Grid Plugtests, October 17-20 2004



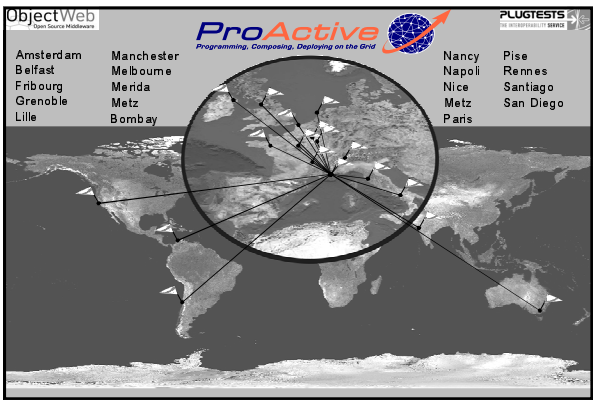
ProActive Grid Plugtests, Oct. 2004

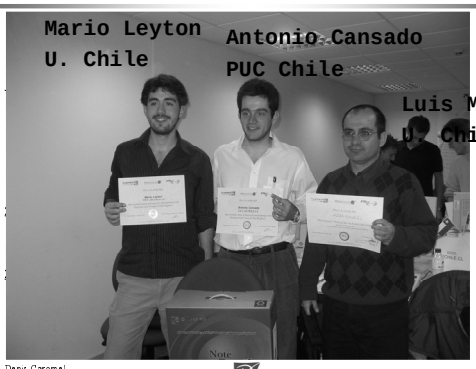
Over 20 sites in the world:

- 800 processors
- Total power: 100 Giga Flops

Highly heterogeneous :

Machines:	IBM, SGI, Sun, Bull, Mac
OS:	Linux, Windows, Solaris, MacOS, SGI Irix
JVMs:	Sun, SGI, BEA
Protocols:	ssh, rsh, sshGSI, Globus Gram
Job Schedulers:	PBS, LSF, Grid Engine, Oar, Prun, Globus





Perspective Adaptive Feature: Future update strategies

No partial replies and requests:

- No passing of futures between activities, more deadlocks

Eager strategies: as soon as a future is computed

- Forward-based:
  - Each activity is responsible for updating the values of futures it has forwarded
- Message-based:
  - Each forwarding of future generates a message sent to the computing activity
  - The computing activity is responsible for sending the value to all

Mixed strategy:

- Futures update any time between future computation and WbN

Lazy strategy:

- On demand, only when the value of the future is needed (WbN on it)

9. New Features Ver. 2.1 Oct. 04 Ver. 2.2 April 05

9.1 HTTP communication layer

Why ?

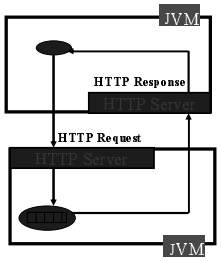
- Facilitate firewalls crossing between ProActive runtimes

How ?

- Encapsulate ProActive Requests into HTTP

Usage :

- Transport configuration in a XML file
  - → Fully transparent !



9.2 Active objects as Web Services

Why ?

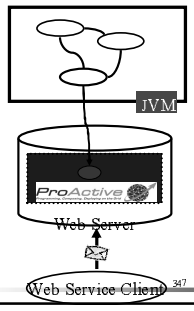
- Make active objects accessible from any foreign language

How ?

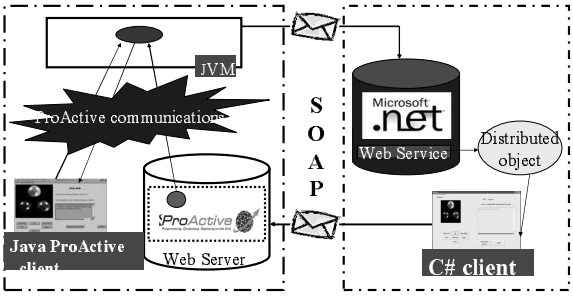
- HTTP Web Server
- Extended SOAP Engine (*Apache SOAP or Axis*)

Usage :

- `ProActive.exposeAsWebService();`
- `ProActive.unExposeAsWebService();`



9.3 .net / C# interoperability



Peer-to-Peer Computing
Model & Infrastructure

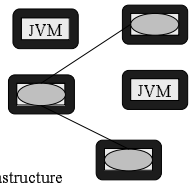
Use Sparse CPU Cycles from Desktop Workstations

- Dynamic Computational Peer-to-Peer (P2P):
  - Intranet & Internet
  - Propose a High Level Model
    - Dynamic JVMs Network (computation nodes)
    - P2P Programming Model for Branch and Bound (B&B) problems
- ProActive Context: no modification of Java language, of JVMs, ...

P2P Infrastructure

Hosts Network ➔ JVMs Network

Dynamic environment :
- Discovery : recording and un-recording
- Resources (JVMs) acquisition



Self-Organizing and Tunable Infrastructure :
- Time To Update (TTU): peer availability
- Number Of Acquaintances (NOA): keep up infrastructure
- Time To Live (TTL): in hop for JVMS depth search, use for NOA

P2P Programming Model Branch & Bound

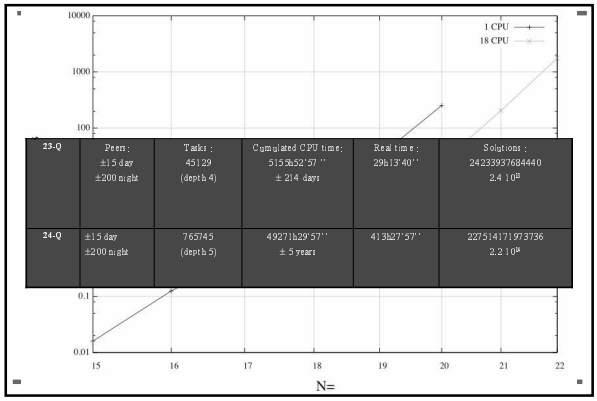
Dynamic P2P Programming for B&B, etc.:
- Tasks, sub-tasks managing?
- Tasks communications: discovery tasks, volatility?

Entities:
- **Worker**: connects model with infrastructure.
- **Solver**: Worker associate.
- **Problem**: Worker associate (Dividing, Merging, Finding)
- **Result**: solution abstraction

Communications between Workers

P2P NQueens with Vincent Cavé

- Master-slave application with Dynamic Workers Acquisition
- Generic Workers (reusable)
- Fault tolerant (Workers availability)



23-Q	Peers : ±15 day ±200 night	Tasks : 45129 (depth 4)	Cumulated CPU time: 5155h52'57 '' ± 214 days	Real time : 29h13'40''	Solutions : 24233937684440 $2.4\ 10^{13}$
24-Q	±15 day ±200 night	765745 (depth 5)	49271h29'57'' ± 5 years	413h27'57''	227514171973736 $2.2\ 10^{14}$

Fault-Tolerance for ProActive

- ✓ Provide a fault-tolerance protocol for ProActive
  - ✓ Maintain portability of ProActive: use only standard Java serialization for checkpointing
  - ✓ Scalable: target Grid applications
- ✓ First Step: applications running on a **single cluster** (homogeneity and low failure rate)

  ➡ ***Communication Induced Checkpointing***
- ✓ Next Step: interconnect clusters using message logging

Implementation

- ✓ This protocol is implemented within ProActive
  - ✓ Single checkpoint and location server
  - ✓ Fault detection (fail-stop) by heartbeat messages
- ✓ Fully transparent use:

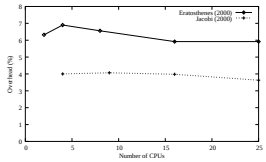
```
~> java org.objectweb.MyAppli
```

```
~> java -Dproactive.ft.server=host.inria.fr org.objectweb.MyAppli
```

- ✓ No need to alter nor recompile code !

Benchmarks

- ✓Master-slaves application: Sieve of Eratosthenes
  - ✓ Overhead less than 7.5 %
- ✓Peer-to-Peer application: Jacobi iteration
  - ✓ Overhead less than 4.5 %
- ✓Scalability



Available in January 2005 Release

10. On-Going + Perpectives

Formal Verification of Behavioral Properties

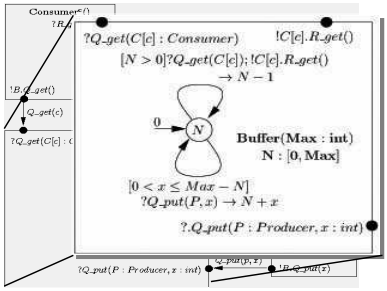
Eric Madelaine

- For Asynchronous Distributed Objects
- Based on Strong Semantic Guaranties
- Application Level Properties

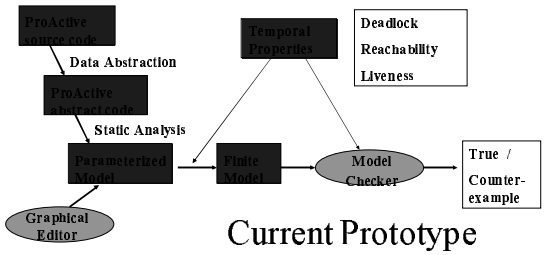
work with

Rabea Boumla, Tomas Barros, Christophe Massol

Models for Distributed Objects

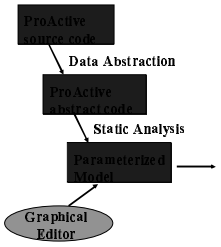


Verification of Properties



Current Prototype

Perspectives



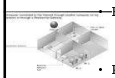
Specification / Implementation

Correct composition of components

ProActive over OSGi

Why targetting OSGi ?

Usage of the OSGi platform is widespread

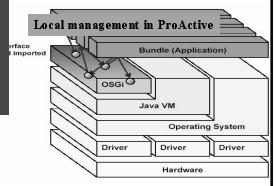


- Residential or Industrial Gateway
  - OSGi enables incremental upgrades and extensions in mission critical « always on » situations

- PDA, smart phones, etc
  - OSGi is a unified mobile platform, with a small footprint and can run disconnectable applications from multiple, independant sources
- Embedded devices (e.g. automotive devices)
  - OSGi provides a viable management solution: enables both end user « pull » and management/operator « push » applications and services

ProActive-enabled OSGi Services : what for and how ?

• A ProActive-based remote management solution of multiple OSGi platforms
  – mgt bundle on each platform
  – a remote monitoring acting in



•Future generation apps built as coordinated OSGi distributed and/or mobile services (e.g. load distribution in a car)

Conclusions

Conclusions and A Few Directions

ProActive A Strong Programming Model + A Strong Deployment

Features not mentioned: Fault Tolerance (CIC), Load-Balancing, P2P

PERSPECTIVES:

- Better Error Management (Exceptions, ...)
- Behavior Specification (Model Checking)
- Tools for Distributed Programming:
  - Code Analysis, Transformation, Integrated IDE

A Grid Alchemy to further develop:

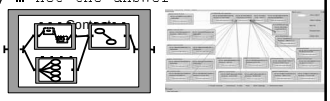
- Asynchrony / Wait By Necessity + Groups + Components

Conclusion (1)

- GRIDs: - Scientific - Enterprise - Internet
  Strong convergence in process (infrastructure): WS, WSRF
- Challenge: adequate and precise programming+composing model
  - Asynchronous distributed objects, Mobility, Components
  - High-level of abstraction, still Open and flexible
  - Modern languages: Java, or others
    not Fortran, MPI, C++, … not the answer

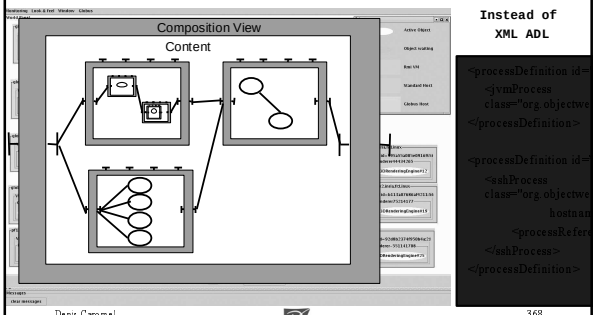
Perspectives:

- Real P2P
- Interactive, Graphical, Deployment and Control:

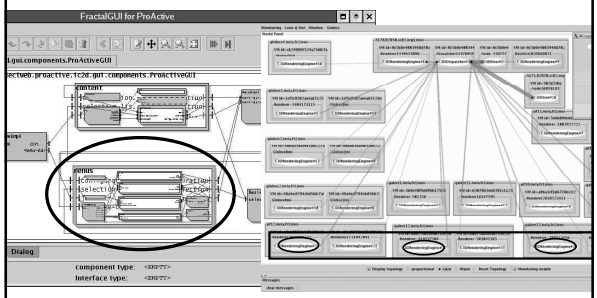


http://ProActive.ObjectWeb.org

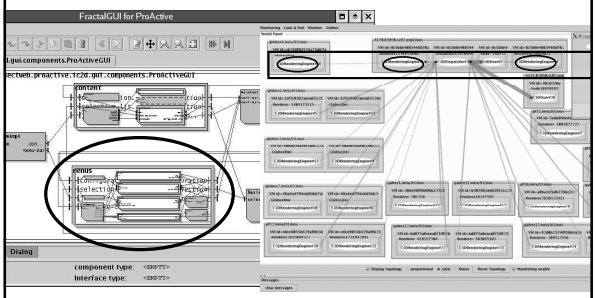
Interactive Composition in IC2D



Perspective for Components - PSE Graphical Composition, Monitoring, Migration



Perspective for Components - PSE Graphical Composition, Monitoring, Migration



Conclusion (2)

Infrastructure and Deployment

- Virtual Nodes and XML
- Protocols: ssh, Globus, LSF, PBS, …
- Transport: RMI, Ibis (Amsterdam)
- Web Services: XML, WSDL, SOAP (ongoing)
- OSGi (future work)

Conclusion (3): Formal Background

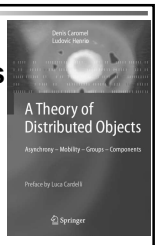
- Performance Evaluation for mobility: Markov Chains
  - Forwarder strategy
  - Server strategy
  - Mixed: TTL-TTU
- A calculus: ASP: Asynchronous Sequential Processes
  - Capture the semantics, and demonstrates the independence of activities
  - Results on Confluence and Determinism [POPL 2004]

THEORY

A Theory of Distributed Objects

D. Caromel, L. Henrio,

Springer 2005, Monograph



A Calculus:

ASP: Asynchronous Sequential Processes

- **Based on Sigma-Calculus (Abadi-Cardelli)**
- **Formal Proofs of determinism**
- **Releases a few important implementation constraints**

ASP theory: Summary and Results

ASP ⇒ Confluence and Determinacy

Proved Properties:

Future updates can occur at any time, in any order

Asynchronous FIFO point-to-point is enough for Requests

Execution characterized by the order of request senders

Applications:

Determinacy of programs based on a dynamic property (DON)

Determinacy of programs communicating over trees,

Deterministic Components, …

Conclusion's Conclusion

Summary:

- **Programming: Distributed Objects, OO SPMD**
- **Composing: Hierarchical Components**
- **Deploying: ssh,Globus,LSF,PBS,…,Web Services**

Application: 3D Electromagnetism on 300 machines at once, Groups of over 1000!
Goal: Towards a few 1000s !

Available in Open Source **ProActive** http://ProActive.ObjectWeb.org **in**

Conclusions and A Few Directions

ProActive A Strong Programming Model **+** Components

FACTS AND FIGURES

5 years of computation in 17 days in Desktop P2P

Deployed at once on 600 CPUs (Plugtests on ssh, Globus, LSF, ...)

(Close to) Beating Fortran on an Electromagnetic Application

PERSPECTIVES FOR COMPONENTS

Safe Reconfiguration

Behavior Specification (SCCC'04, Attali-Barros-Madelaine)

A great alchemy for the Grid:

Asynchrony + Wait By Necessity + Groups + Components

Conclusion

ProActive
Programming, Composing, Deploying on the Grid

Async. Distributed Objects
Groups, Components, Mobility
Grid Deployment, Secu, FT

Current Applications and Benchmarks:

- 5 years of computation in 17 days in Desktop P2P
- A single application on a 1000 processors GRID

Goal: towards a few 1000s!

Looking for collaborations:

- Building reusable Cpts from Numerical Codes
- Generic Techniques for Coupling Codes

CoreGRID

Available in LGPL with **ObjectWeb** http://ProActive.ObjectWeb.org

Some More 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 Non Functional Properties

Currently in ProActive:

- Remotely accessible Objects
  (Classes, not only Interfaces, Dynamic)
- Asynchronous Communications
- First Class Futures: Wait-By-Necessity
- Group Communications, OO SPMD
- Mobility
- Visualization and monitoring (IC2D)
- Fault-Tolerance (checkpointing),
- Security
- Components
- Non-Functional Exceptions: Handler reification (prototype)

ProActive Non Functional Properties

Currently in ProActive:

- Remotely accessible Objects
  (Classes, not only Interfaces, Dynamic)
- Asynchronous Communications
- First Class Futures: Wait-By-Necessity
- Group Communications, OO SPMD
- Migration (Computational Mobility)
- Visualization and monitoring (IC2D)
- Security
- Components
- Non-Functional Exceptions: Handler reification

ProActive and Adaptive

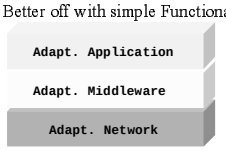
5 Adaptive/Parameterized features in ***ProActive***:

- RMI <--> ssh/RMI ... HTTP - Ibis/TCP - Ibis/Myrinet - ...
- Groups, Localization in Mobility, Security, Future Update

Perspectives -- On-going work:

- Adaptive Components:
  - Tensionning
  - Re-configuration
- Adaptive Checkpointing

Better off with simple Functional RMI / Two-sided MPI Message Passing S



Lets just be careful:
otherwise, we'll just build ...

Maladaptive Adaptive Grids !

TCP is an Adaptive Middleware

Practical CoreGrid Lausanne, Sept. 05

1. ProActive Documentation and Installation
2. Guided Tour, 2.2 Collaborative and Parallel: C3D

ProActive.ObjectWeb.org