2. ProActive : Migration of active objects

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

Can be initiated from outside through any public method

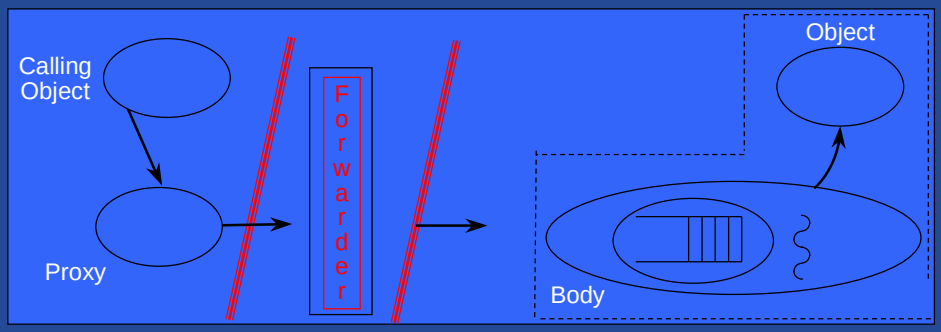
The active object migrates with:

- all pending requests
- all its passive objects
- all its future objects

Automatic and transparent forwarding of:

- requests (remote references remain valid)
- replies (its previous queries will be fullfilled)

ProActive : Migration of active objects



Migration is initiated by the active object through a request

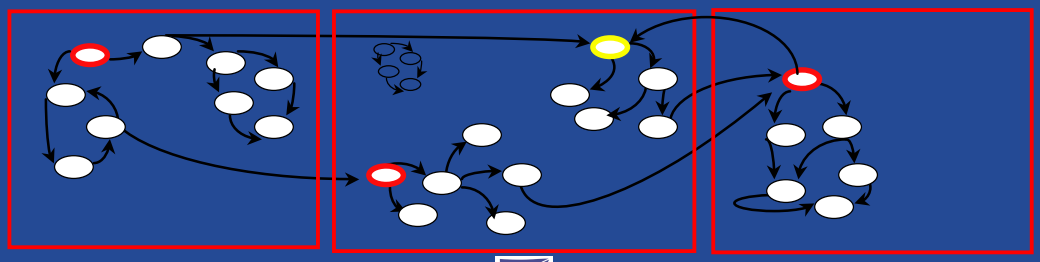
The active object migrates with

- its passive objects - the queue of pending requests - its future objects

2 Techniques : Forwarders or Centralized server

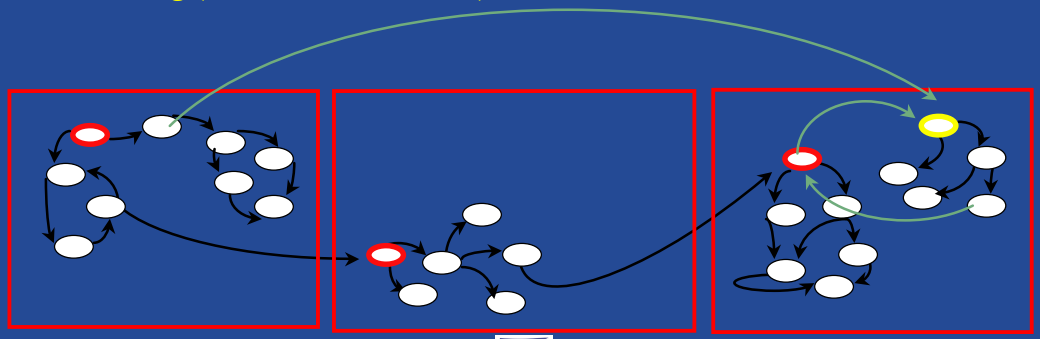
Principles

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



Principles

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



ProActive : API for Mobile Agents

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

ProActive : API for Mobile Agents

- Mobile agents (active objects) that communicate

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

ProActive : API for Mobile Agents

- Mobile agents that communicate
- Primitive to automatically execute action upon migration
  - public static void onArrival (String r)

    // Automatically executes the routine r upon arrival

    // in a new VM after migration
  - public static void onDeparture (String r)

    // Automatically executes the routine r upon migration

    // to a new VM, guaranted safe arrival
  - public static void beforeDeparture (String r)

    // Automatically executes the routine r before trying a migration

    // to a new VM

ProActive : API for Mobile Agents Itinerary abstraction

Itinerary : VMs to visit

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

API:

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

Still communicating, serving requests:

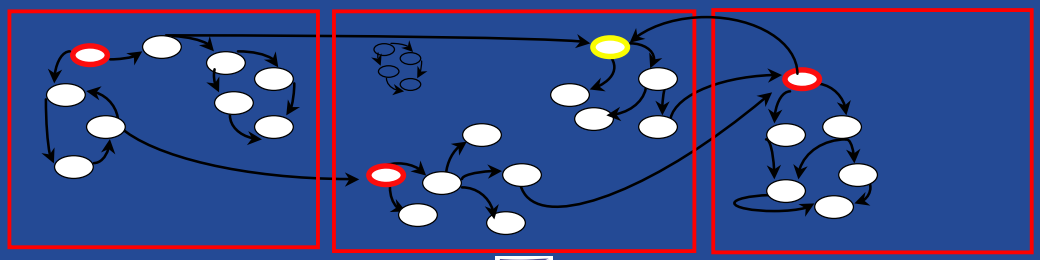
- itineraryMigrationFirst ();

  // Do all migration first, then services, Default behavior
- itineraryRequestFirst ();

  // Serving the pending requests upon arrival before migrating again

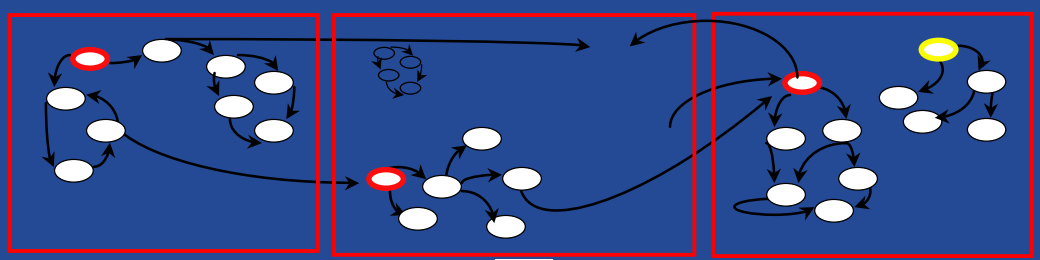
Characteristics and optimizations

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



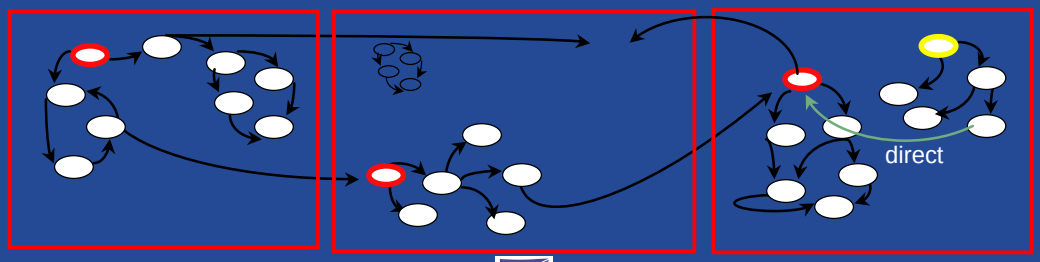
Characteristics and optimizations

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



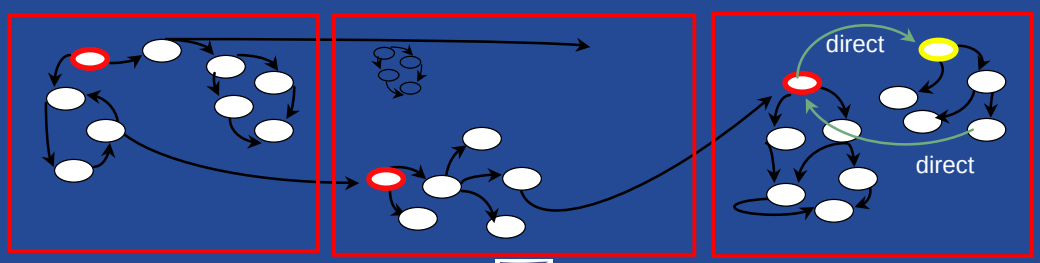
Characteristics and optimizations

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



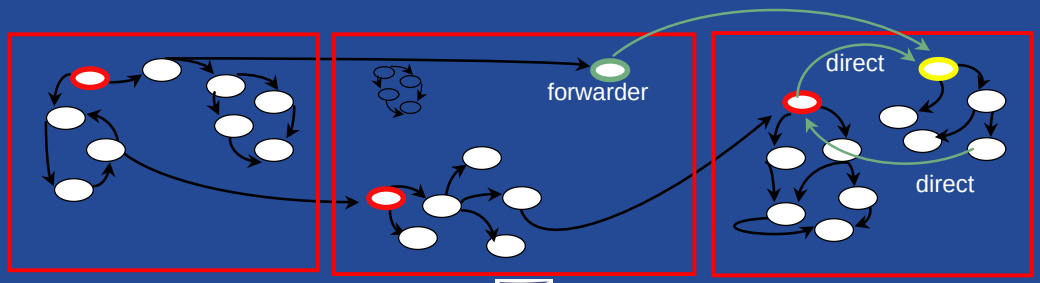
Characteristics and optimizations

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



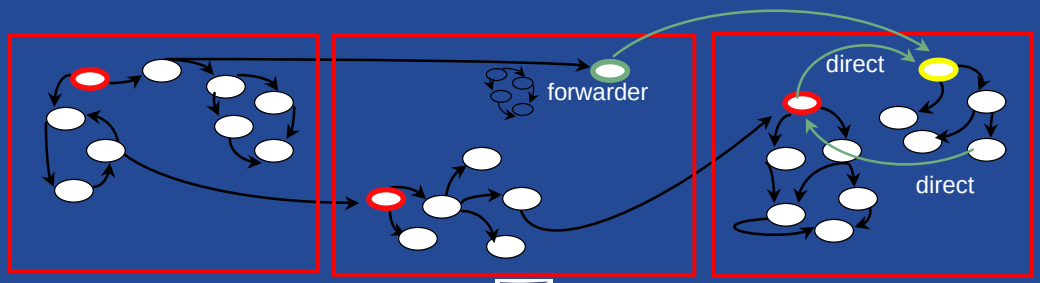
Characteristics and optimizations

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



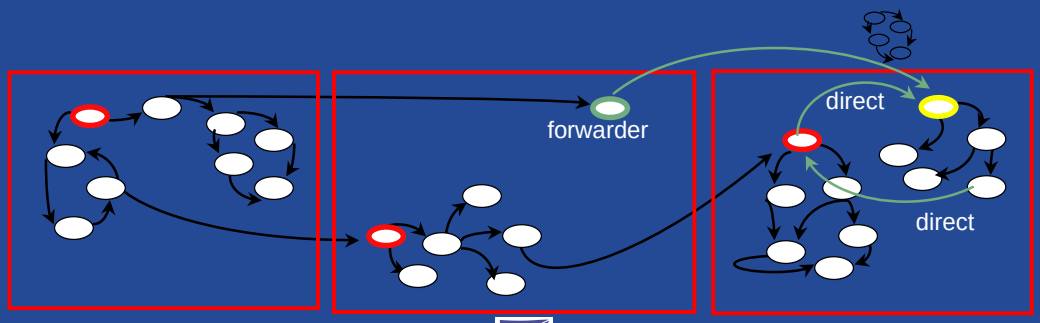
Characteristics and optimizations

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



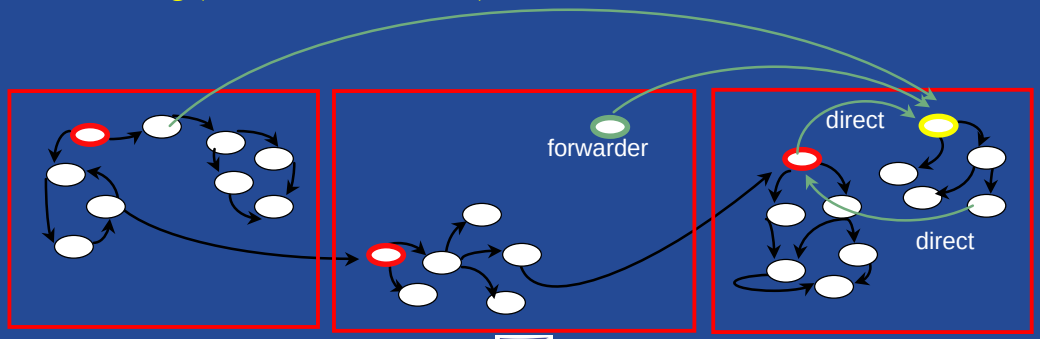
Characteristics and optimizations

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



Characteristics and optimizations

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



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-

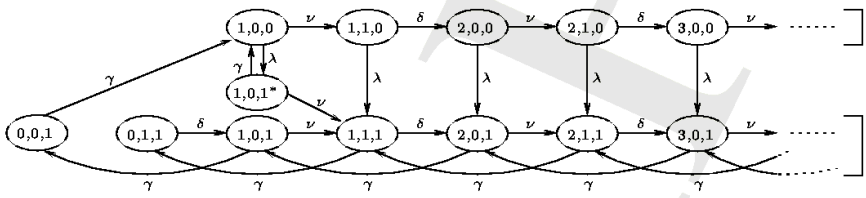


FIG. 4.2 – *États et transitions du mécanisme des répéteurs*

3 -

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

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

4 -

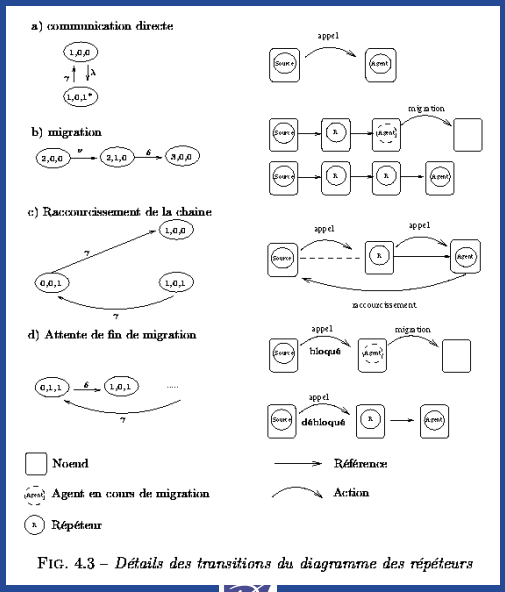


FIG. 4.3 – *Détails des transitions du diagramme des répéteurs*

5 -

Analyse Markovienne du serveur centralisé

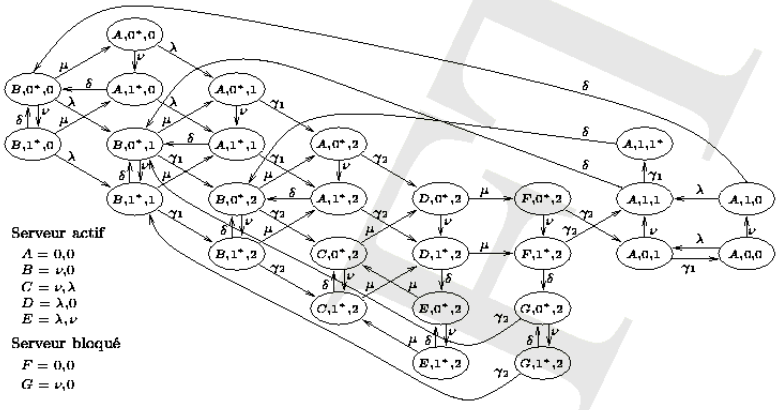


FIG. 4.6 – *État du système et taux de transitions dans le mécanisme du serveur.*

6 -

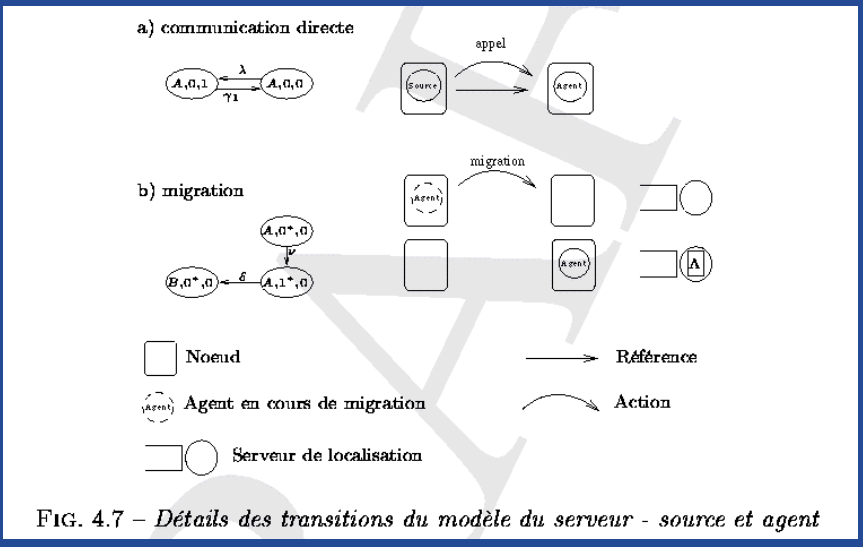


FIG. 4.7 – *Détails des transitions du modèle du serveur - source et agent*

7 -

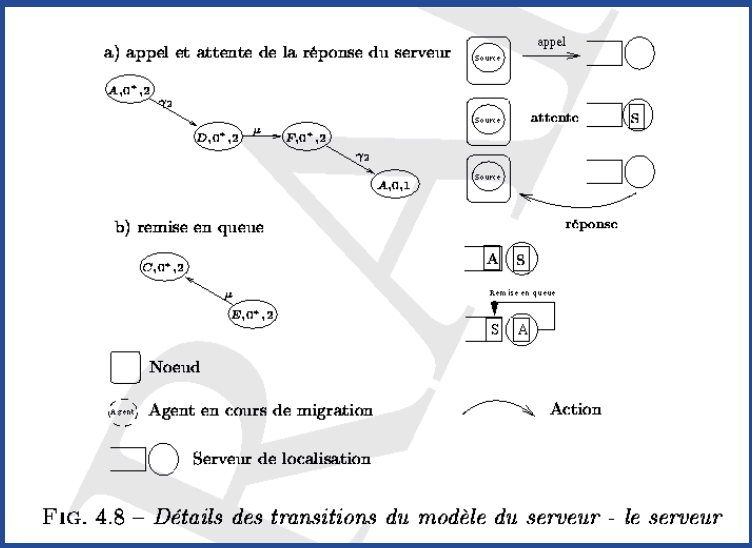


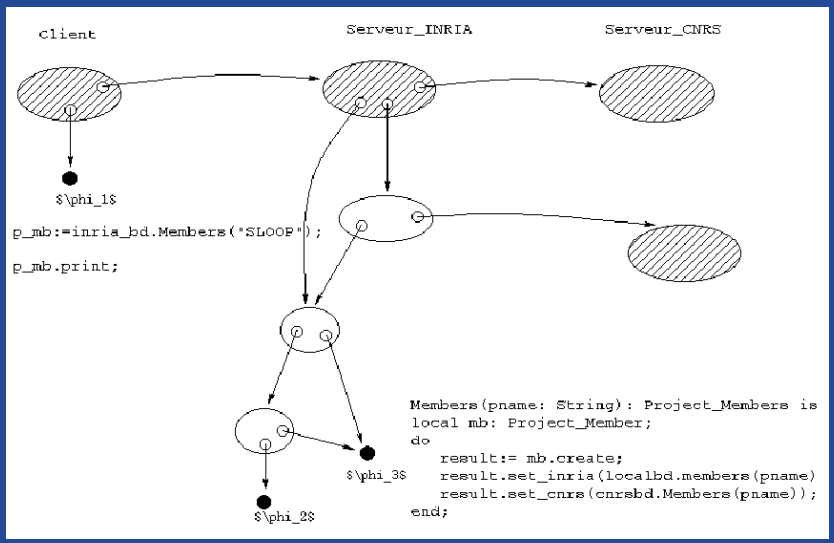
FIG. 4.8 – *Détails des transitions du modèle du serveur - le serveur*

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)

ProActive Non Functional Properties

Currently in ProActive:

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

Others:

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

Some Perspectives

Non-functional Exceptions:

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

ProActive Components:

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

Checkpointing:

- Communication induced checkpoints
- Message logging

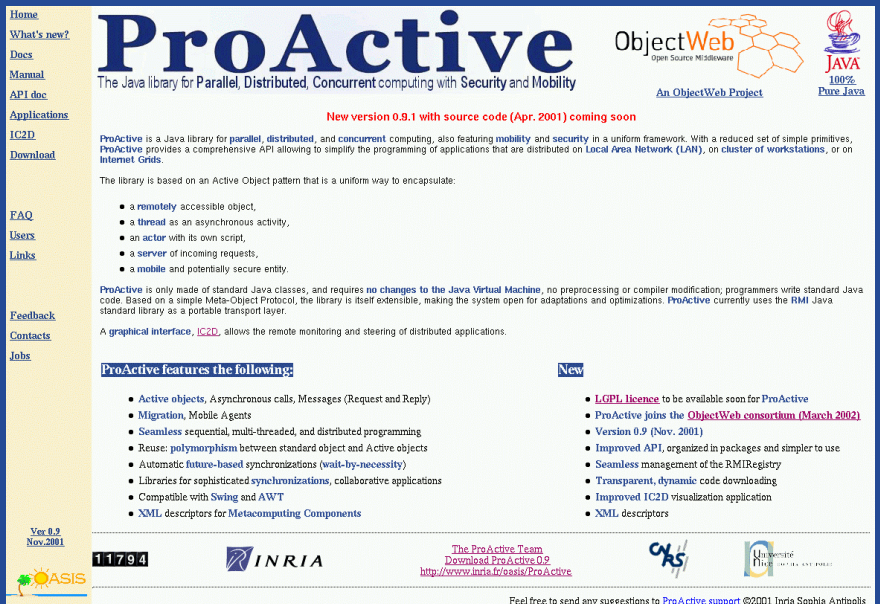
--> Components and Deployment Descriptors integration

Conclusion

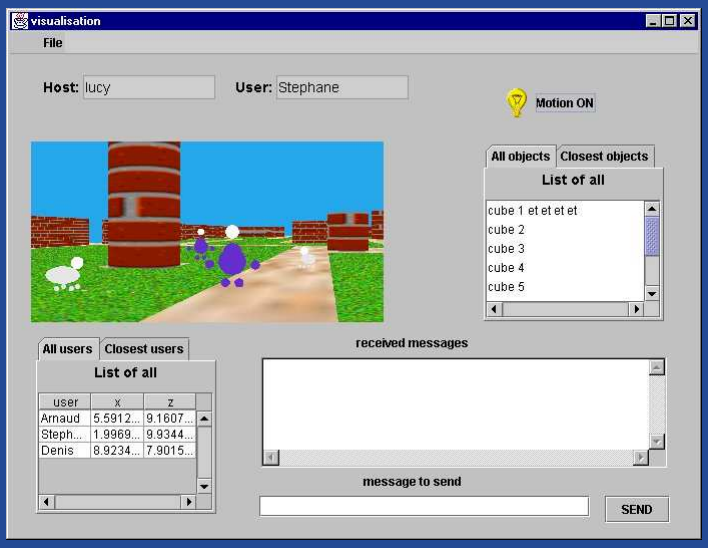
- A library: *ProActive* 100% Java
- Parallelism, Distribution, Synchronization (CSCW), Group and Mobility
- Reuse -- seamless
  - Polymorphism with existing class types
  - Asynchrony -- Wait-by-necessity
- An interactive tool towards GRID: **IC2D**
- A calculus: ASP: Asynchronous Sequential Processes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

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

www.inria.fr/oasis/ProActive



DIVA: Distributed Int. Virtual World in Java



Extra Material

An Object-Oriented Application for 3D Electromagnetism

Together with:

Francoise Baude, Roland Bertuli, Christian Delbe (OASIS),

Said El Kasmi, Stéphane Lanteri (CAIMAN)

3D Electromagnetism Applications:

- Visualize the radar reflection of a plane

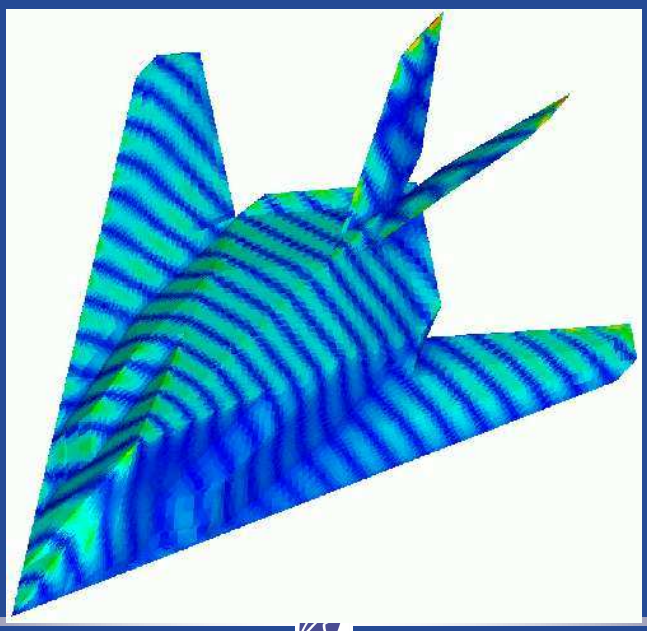
Goals:

- Sequential object-oriented design, modular and extensible
- Designed to allow both Element and Volume type methods

  --> currently 3D Maxwell equations, towards CFD
- Valid on structured, unstructured, or hybrid meshes.
- Sequential version can be smoothly distributed,

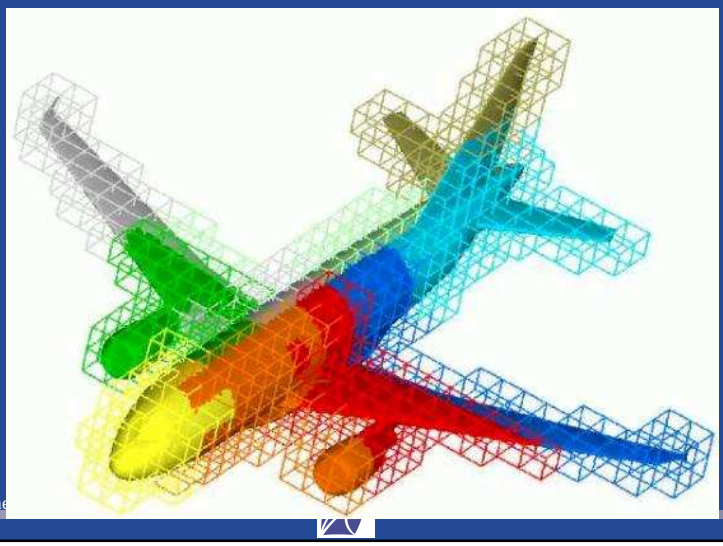
  --> keeping structuring and object abstractions

Avion Furtif F117



Airbus A318

Meshing, 1 color par processor, 9 here



Geometry definition

Meshes: Vertices, and Elements

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

- a very general method

Computation of a flux balance through the boundary of Control Volume

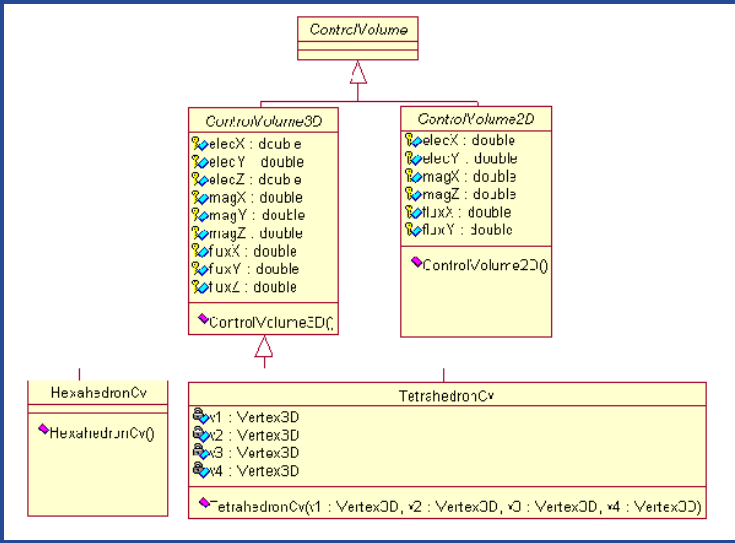
- the calculation support of FVM (vs. Vertices of the mesh)

Example, and experimentation:

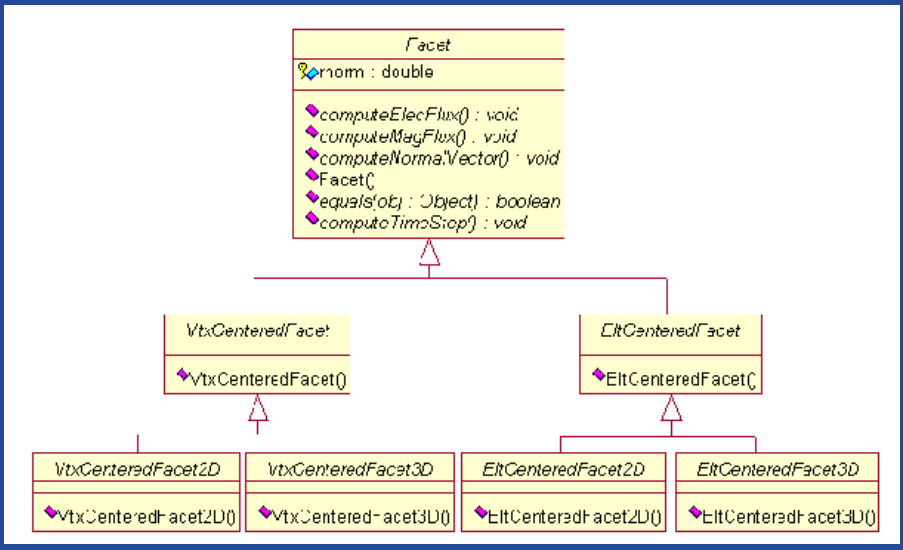
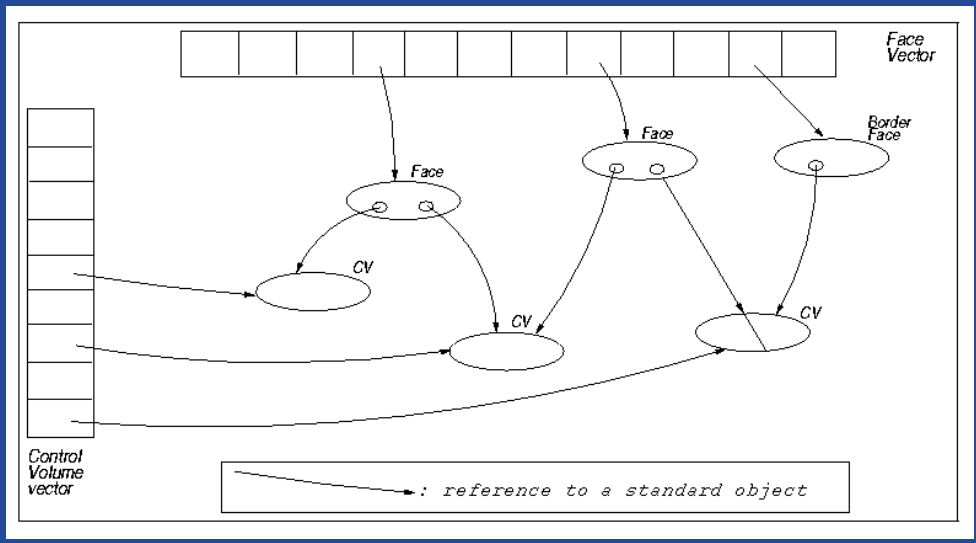
- Control Volume = Tetrahedron
- Facet = Triangle

----> Picture

Control Volume in 2D and 3D



Facets in 2D and 3D



Architecture of the sequential version