ECOLE DES MINES DE PARIS

INRIA

REJO / ROS

Raúl ACOSTA BERMEJO

Outline

- *Motivation*
  - *RAMA: architecture, example*
- *REJO:*
  - *REJO structure*
  - *Execution Model*
  - *Reactive Method Implementation*
- *Mobile Agents*
- *ROS*
  - *Migration*
  - *Agent structure*

RAMA

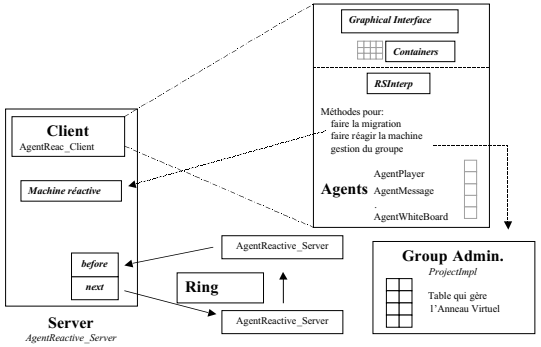
Reactive Autonomous Mobile Agent

A set of ***Java classes*** built by Navid Nikaein
Sep/99, DEA RSD

These classes are used to build ***Mobile Agents***
on top of the ***Reactive Approach***.

Agents are executed on a ***platform*** that offers some services, for instance group service.

Architecture



Example

```
package rama;
import  inria.meije.rc.sugarcubes.*;
import  java.awt.Label;

public class WhiteBoardAgentCode extends AgentPrimitive
{
    public void start(String home)
    {
      try{
          Instruction WhiteBoardAgent=
              new Cube(new JavaStringValue("WhiteBoard_Agent_" +(counter++)+home),
                       new JavaObjectValue(
                           new WhiteBoardAgent("WhiteBoard_Agent_"+(counter-1),home)),
                       new Until("kill", new Seq(
                                         new Await("Migrate"),
                                         new Loop(
                                                   new Until(new OrConfig(
                                                                 new PosConfig("Migrate"),
                                                                 new PosConfig("Return")),
                                                             new Merge(new Seq(
                                                                           new Await("Return"),
                                                                           _migrationToHome()    )
                                                                       ,new Seq(
                                                                           new Await("Migrate"),
                                                                           _migrationCode()      )
                                                                       )
                                                             )
                                                   )
                                         )
                                 ))
                       ,new JavaInstruction(){ public void execute(Link self){
                           ((Agent)self.javaObject()).disposeMessage();
                       }}

                       ...

              );
          AgentReactive_client.currentSite.machine.add(WhiteBoardAgent);

      }catch(Exception e){
          System.out.println("Error when sending the agent:"+e); }
    }
}
```

Nom

Linked Obj

Program

Handlers

Summary

Characteristics:

- Concurrency
- Interactivity
  Broadcast & Reactive Instructions
- Mobility
- Dynamic (agents, platform)
- Multi-platform

Thanks to **SugurCubes** et **Java**

Drawbacks:

- Monolithic Architecture.
- Programming (style) à la SugarCubes.
- Naming problems
- Local broadcast and thus it is necessary to migrate in order to communicate

Advantages:

- Ring Routing
  but this is also a drawback because of big rings & only Algorithm.
- Reactive Agent Model
- Services may be implemented as agents.

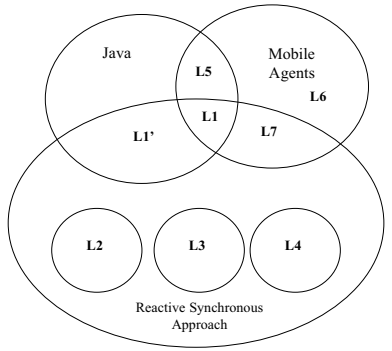
Agent (Cube) = Link + Shell + Freeze

Motivation - 1

We would like to:

- Program reactive Systems à la Reactive Scripts
  but it is not Java, there isn't platform, ...
- Have agent services:
  migration, naming, routing, etc.
- May add and remove modules from the platform to:
  - add a synchronizer, Distributed Reactive Machines (DRM).
  - have the group service.
  - pick the routing algorithm.
  - ...

Motivation - 2



L1 : ***Agent Reactive Languages in Java***
REJO, MARS
L1' : ***Reactive Languages in Java***
SugarCubes, Reactive Scripts, Junior
L2 : ***Imperative Reactive Languages***
Esterel
L3 : ***Declarative Reactive Languages***
Signal, Lustre
L4 : ***High order Reactive Languages***
Library in SML [RP98]
L5 : ***Agent Languages in Java***
Aglets, Concordia, Voyager.
L6 : ***Agent Languages***
Messengers
L7 : ***Agent Reactive Languages***
ROL [GM99]

REJO

REactive Java Object

REJO is an extension of Java that creates ***Reactive Objects***, i.e. objects that have data and a mix of Java instructions and reactive instructions.

The execution model is that of reactive Synchronous Model which executes the Java instructions in an atomic way.

These objects may be considered as ***Mobiles Agents*** because they can migrate using a platform, called ROS, that provides the functionalities they need.

REJO structure

```
import ros.kernel.bin.*;
import java.awt.*;

public class rejo implements Agent
{
 int cont;

      public void Methodes_1()
      {
         body
      }

      public reactive rmain(String[] args)
      {
       int ini=5;

          par{
             cont=ini;
             Methodes_1();
             until("Preemption1" && "Preemption2")
               loop{
                 Methodes_1();
                 stop;}
          ||
             call Methodes_2();
          }
      }

      public reactive Methodes_2()
      {
        body
      }
}
```

REJO = **Data** + **Java Code** + **Reactive Code**

REJO instruction set

- ;
- stop
- atom, *expr*
- par{ b1 || b2 }
- loop
- repeat
- if-else

- gen, generate, !
- wait, ?
- when-else
- until-handler
- local
- control
- freezable

currentValues
previousValues

- call
- inline
- try-catch
- print, println

Link

Translation Examples - 1

```
react method( param )
{
  var loc;
     body
}
```

=>

```
Program method_id()
{
    obj = new _Method();

    return Translation(body) ;
}

class _Method()
{
  var loc;
  param;
}
```

Translation Examples - 2

```
i1;                    =>    Jr.Seq( i1, Jr.Seq(
i2;                            i2, Jr.Seq(
... ;                          iN-1, iN  ) ) )


par                    =>    Jr.Par(
{
   i1;                         i1, Jr.Par(
||
   i2;                         i2, Jr.Par(
||
   ...                         iN-1, iN ) ) )
}
```

Translation Examples - 3

```
loop {                 =>    Jr.Loop ( body )
  body
}


Assign:   i=5;         =>    Jr.Atom(new Action(){
          i= Var Loc ou Glo          public void execute(Environment env)
          Arithm. Exp., etc.         {
Invoke:   meth();                        Instructions;
          obj.meth();                }}
                                   )

atom{
   Expressions;
   Flow Control Structures
       if, for, while.
}
```

Translation Examples - 4

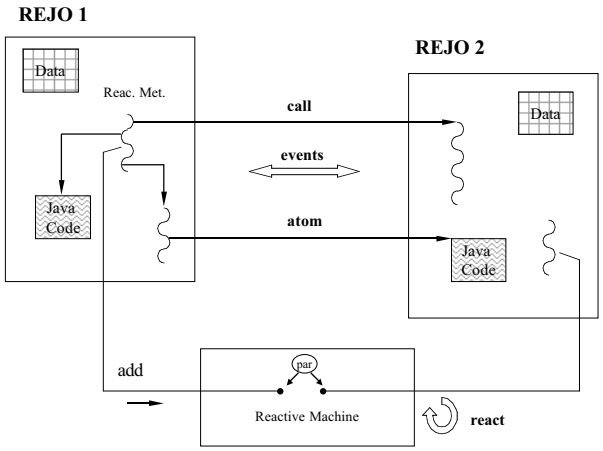
```
repeat( Cte ){      =>      Jr.Repeat( Cte , body )
   body
}

repeat( Var ){      =>      Jr.Repeat (new VarIntegerWrapper (locVar, nAtom),
   body                                   body
}                                      )
```

The same thing for *If* and *When* instructions

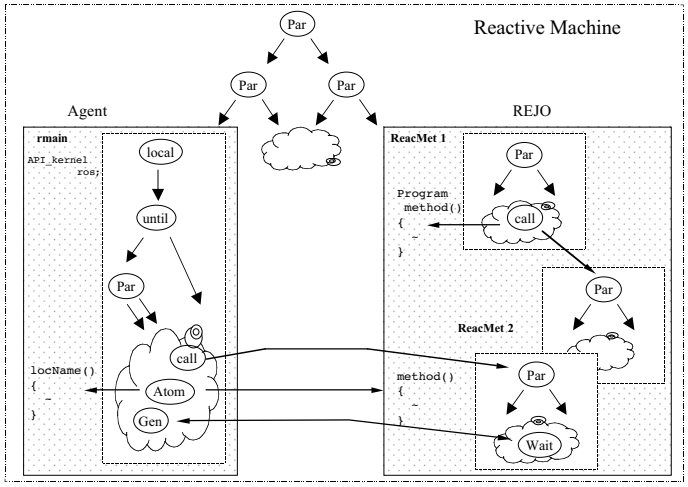
Translation Examples - 5

```
wait  "eve" + var + met()  =>  Jr.Await(
                                    Jr.Presence(new VarIdentifierWrapper(locVar,  nAtom))
                                )
        case nAtom:
                        return "eve" + var + met();


wait  "eve" && var         =>  Jr.Await (
wait  var || met()                  Jr.And( Jr.Presence( … ),Jr.And( … ) )
wait  ! "eve "                      Jr.Or( Jr.Presence( … ),Jr.Or( … ) )
                                    Jr.Not( … )
                                )
```

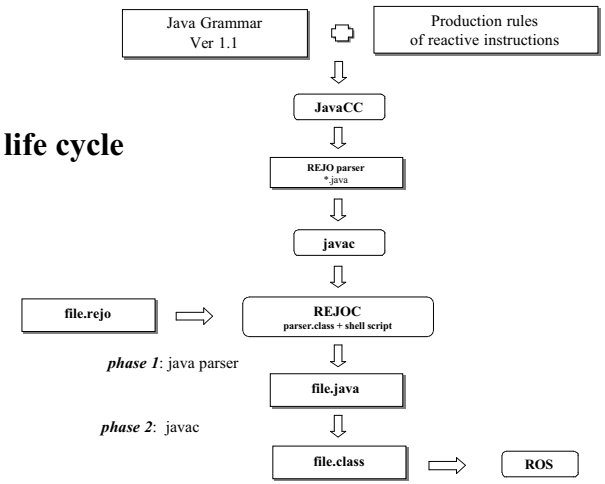
Execution Model - High level

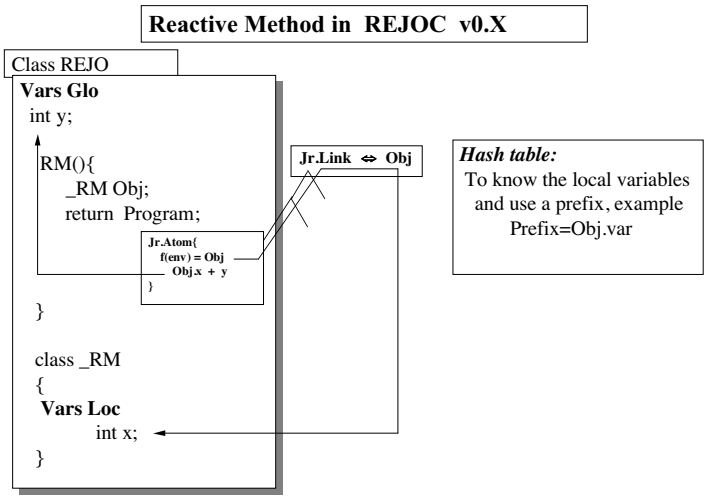


Execution Model - Low level

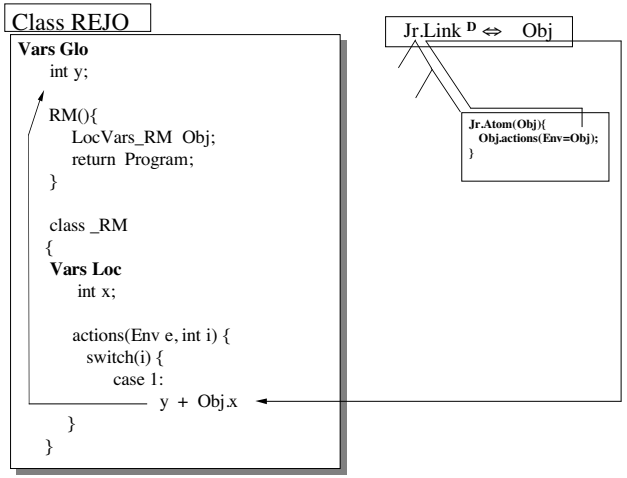


REJO life cycle

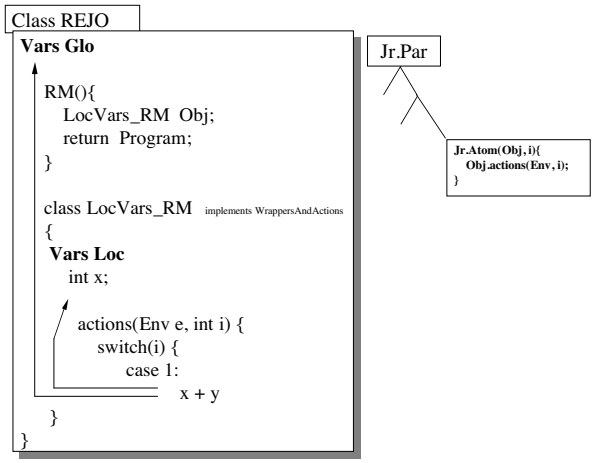




Reactive Method in REJOC v1.1r1



Reactive Method in REJOC v2.0r1



REJO summary

REJO not only makes easier the reactive programming (avoiding the utilization of parentheses) it allows programmers to:

- Mix the reactive model with ***Java variables***
  to use them in conditions (when, wait, …) or values (if, repeat).
  REJO hides the utilization of wrappers.
- Mix the reactive model with ***Java instructions***
  Arithmetic expressions & Java method invocations.
  Atomic execution of Java instructions.
- Use ***Reactive Methods*** to:
  Modular programming.
  Re-use code making invocations to reactive methods.
- Use ***local variables*** inside of reactive methods.

In conclusion, all these elements define a **Reactive Object Model** that doesn't exist in Junior. The other models defined in Java are Rhum and SugarCubes.

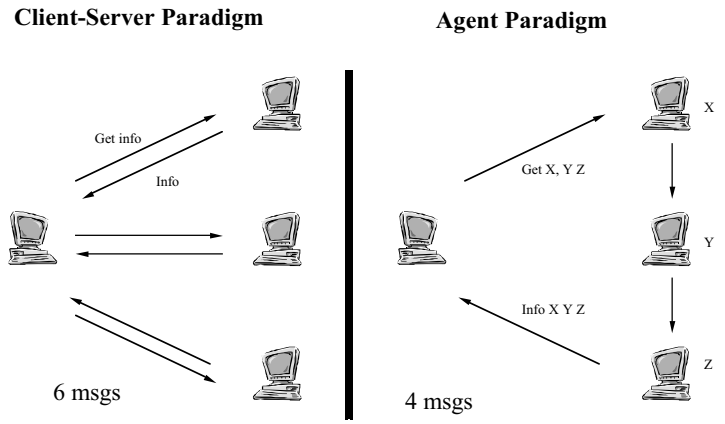
Agents

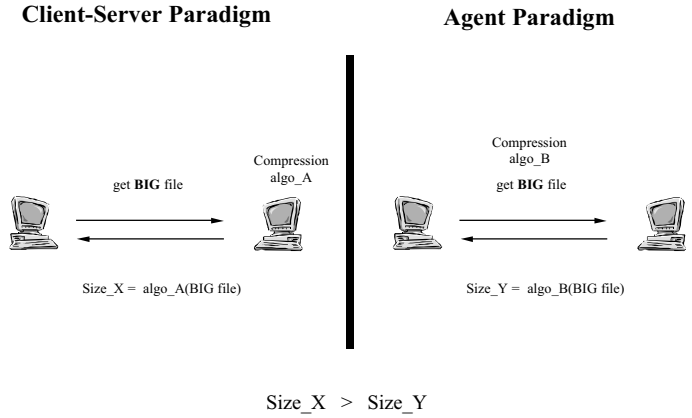
An ***agent*** is a program that is autonomous enough to act independently even when the user or application that launched it is not available to provide guidance and handle errors.

A mobile agent is an agent that can move through a heterogeneous network under its own control, migrating from host to host and interacting with other agents and resources on each, typically returning to its home site when its task is done.

Advantages

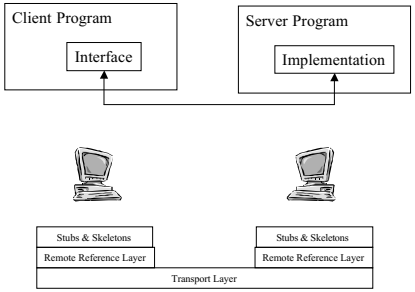
- Bandwidth reduction
- Latency reduction
- Support for disconnected operation (network fails).
- Load balancing
- Development of applications (personalize server behavior).


Client-Server Paradigm
Get info
Info
6 msgs
Agent Paradigm
Get X, Y Z
X
Y
Z
Info X Y Z
4 msgs

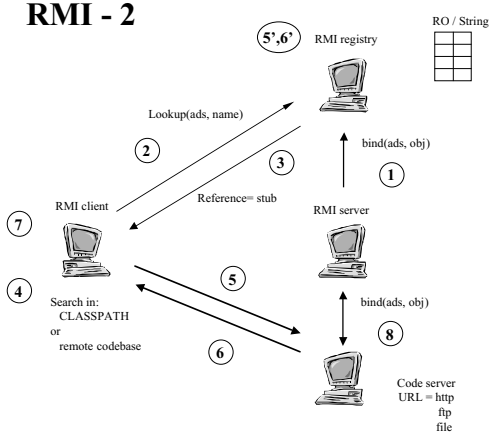

Client-Server Paradigm
get BIG file
Compression
algo_A
Size_X = algo_A(BIG file)
Agent Paradigm
Compression
algo_B
get BIG file
Size_Y = algo_B(BIG file)

Size_X > Size_Y

RMI - 1



RMI - 2



1.Srv registers RO and registry reads srv codebase
2.Clt demands Refer name
3.Srv returns Refer
4. Clt searches stub code
5.Clt request stub from the srv codebase
6.URL srv returns stub class and any other class needed
7.Clt executes a methode on the RO and RO doesn't know a parameter(class)
8. RO downloads the class using clt codebase. If it fails then it uses srv codebase

Plus firewall

Mobile Agent System Design

- Mobility
  Strong or weak
- Routing
  Address transfer or List of sites with their offered services
- Communication
  Message Passing, RMI (RPC), Publish Subscribe, Broadcast
  Com. Inter-group or Intra-group, Local Inter-group or Global
- Naming
  Local Name + Machine Name, Domain Name Server, Global Name
- Language: compiler or interpreter
  Portability, robustness, security, performance

Characteristics		Mobile Agent Systm
Language	*Tcl/Tk:* *Java:* *C/C++:* *Orient Object:* *others:*	*TACOMA, Ara, Agent Tcl.* *Ara,Aglets,Voyager,MARS,Concordia.* *TACOMA,Ara,Messangers(melange C)* *Telescript(~C++),Obliq.* *Messangers(~postcript).*
Communication	*Message-passing:* *Broadcast:*	*Agent Tcl, Aglets, MARS, Odyssey.* *RAMA, REJO/ROS.*
Protocol	*TCP/IP:* *RMI:* *others:*	*TACOMA, Odyssey, MARS.* *Odyssey, RAMA, REJO/ROS.* *Aglets(ATP=AgentTransferProtocol)*
Persistency		*Concordia*
Security	*Authentification:*	*Ara* *Concordia (Tamper)*
Mobility	*Strong:* *Weak:*	*Ara,AgentTcl,Telescript,RAMA,Ajanta,* *Concordia,REJO/ROS.* *Tacoma,Aglets,Obliq,Messengers,Voy.*
Routing	*Itinerary:* *Ring:*	*Concordia* *RAMA*

ROS

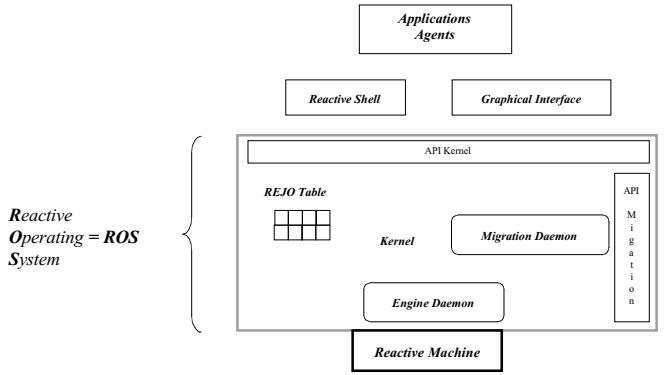
Reactive Operating System

ROS is a system built on top of Reactive Model which executes reactive objects, REJOs.

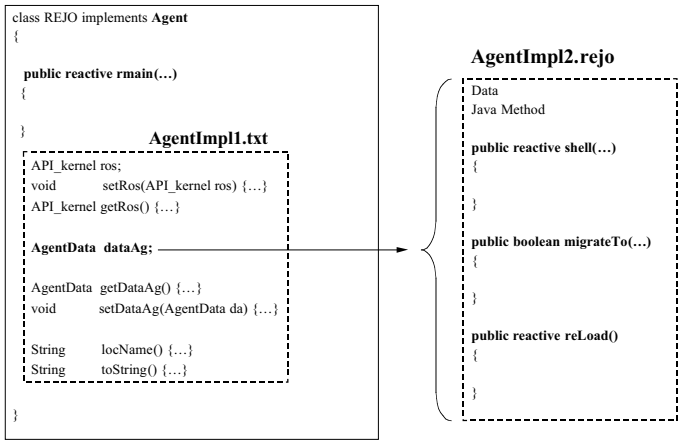
ROS architecture is similar to Distributed Operating System (DOS) architecture, ***micro-kernel***.

ROS was implemented with Junior and thus it has almost all RAMA advantages.

Architecture



Agent structure - 1




```
public reactive shell(handSusp, handResume )
{
 Object[] obj;

      local("step")
         until( locName() + "!kill" ●)
               par
               {
                  loop
                     until( locName() + "!susp" ●)
                           loop(
                              gen "step";                    || getGroup()+"!kill"
                              stop;)
                     handler{
                         call handSusp ;
                         stop;
                         wait locName() + "!resume" ●;
                         call handResume;
                      }
               ||
                  control("step"){
                    par
                    {
                        freezable( locName() + "!migra" ●)
                          call body();
                    ||
                        loop(
                          wait locName() + "!migra" ●, obj;
                          if( migrateTo(obj) )
                            gen locName() + "!kill" ●;
                          else
                            gen locName() + "!reload" ●;
                          stop;
                        )
                    ||
                        wait locName() + "!reload" ●;
                        run reLoad();
                    }
                  }
               }
      handler
          call handTermin();
}
```

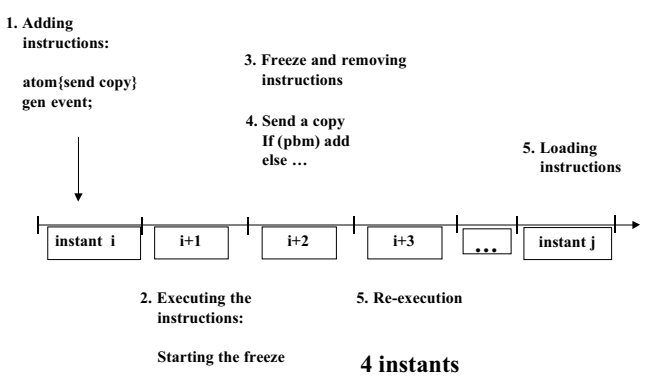
```
public reactive reLoad()
{
   par
   {
     setMigra(false);
     freezable( locName() + "!migra" ●){
        call handWarmUp();
        call body();
     }
   ||
     stop;
     wait locName() + "!reload" ●;
     run reLoad();
   }
  }
}
```

Agent structure - 2

Agent structure - 3

```
public boolean migrateTo(Object[] obj)
{
 String dest="", ROSname="";

      if( obj.length < 2 )     Traitement des paramètres
        …

       Program res = ros.unLoad( locName() + "!migra" );
       if( res == null)
         …

      setBody( res );
      setMigra(true);

      try{
          SendingThe = (MigrationServ) lookup("//"+dest+"/MigrationServAt "+ ROSname );
          if( SendingThe.Agent(rejo) == 0 )
             return false;
     }catch(Exception e) {…}
}
```

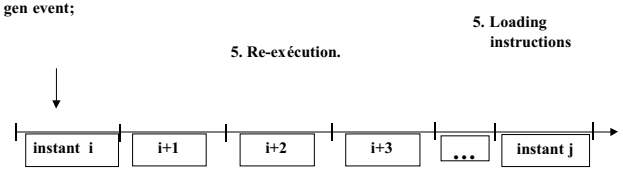
Migration - 1



Migration - 2

1. Adding instructions:
   atom{send copy}

2. gen event;



3. Freeze and removing instructions

4. Sending a copy
   If (pbm) add
   else …

3 instants

Migration - 3

1. gen “migra”, “ads”;
   gen “migra”, “ROSname”;

4. Re-execution

5. Loading instructions

2. Freeze and removing instructions

3. Sending a copy
   If (pbm) run
   else …

2 instants

Conclusions

REJO is a *language* that:

- Makes easier the reactive programming:
  syntax without parentheses
  modularity because of reactive methods
- Keeps Junior properties:
  semantics, cooperative programming, …
- Has an object oriented programming style:
  inheritance, polymorphism
- Offers the means to build mobile agents.

Conclusions

ROS is a *platform* that:

- Executes Reactive Objects (REJOs).
- Allows REJOs to migrate.
- Shows that the Reactive Synchronous Model may be used instead of cooperative threads.
- Has an modular architecture analogue to that of an DOS: micro-kernel.

Future activities 1

- Static Balancing?
  REJO generates a comb tree that is … in REWRITE … STORM
- To analyze the code to:
  Simplify the tree (if or if reactive), …
- To introduce Link instruction
  3 types of variables … meth.var
- Modifications?
  local(eve1, eve2) = local(eve1) local(eve2)
  mix of if and when
  which semantics? if(a==3 && “E”), if(a==3 || “E”)
- Addings:
  this (ou self), generate ads, msg, obj.
  rfor(init; cond; increm) body = <until, loop, if, gen, stop>

```
Class X
{
 int v1;
  reactive m()
  {
   int v2;
     link(obj)
     {
        obj.v3;
     }
  }
}
```

Future activities - 2

Events as numbers
gen 3; wait 5;

wait "str"+*; wait *+var;
wait locName()+*; wait *+"!migra"

Questions?

Raul.Acosta_Bermejo@sophia.inria.fr
http://www.inria.fr/mimosa/rp/ROS/

Problèmes dans la construction d'agents

- Sécurité x
  Programmation coopérative
  Protéger un agent du système et des autres agents
  RMI Socket factory = {crypter, compresser, ≠ protocole}
  Policy,
  Pbm avec RMI registry => JNDI(Java Naming and Directory Interface).
- Routage x
  Topologie, échange d'adresse et service
- Services de persistance, de nomage, de services.
- Langage de programmation √
  Selon le domaine d'application définir des comportement
- Architecture du SAM √
  Couche, groupe, ..

Et CORBA dans tout ça?
Java IDL