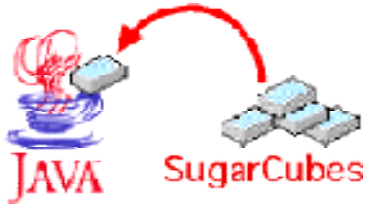


Les SugarCubes

Frédéric Boussinot, Jean-Ferdy Susini

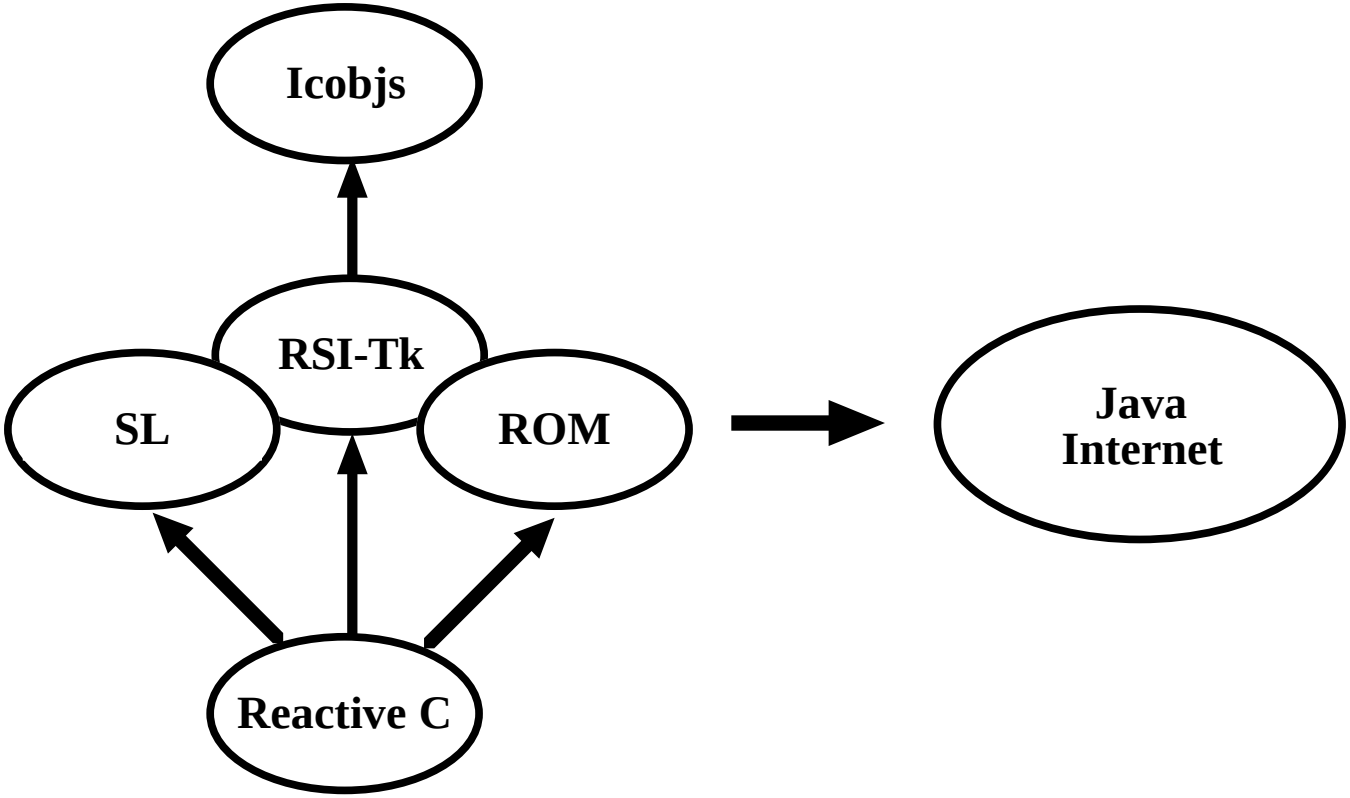
Projet Mimosa commun avec
l'École des Mines de Paris - CMA

Travaux soutenus par France Télécom-R&D

Plan

- Approche Réactive et SugarCubes
- Les Cubes
- Interaction avec Java
- Scripts Réactifs

Historique

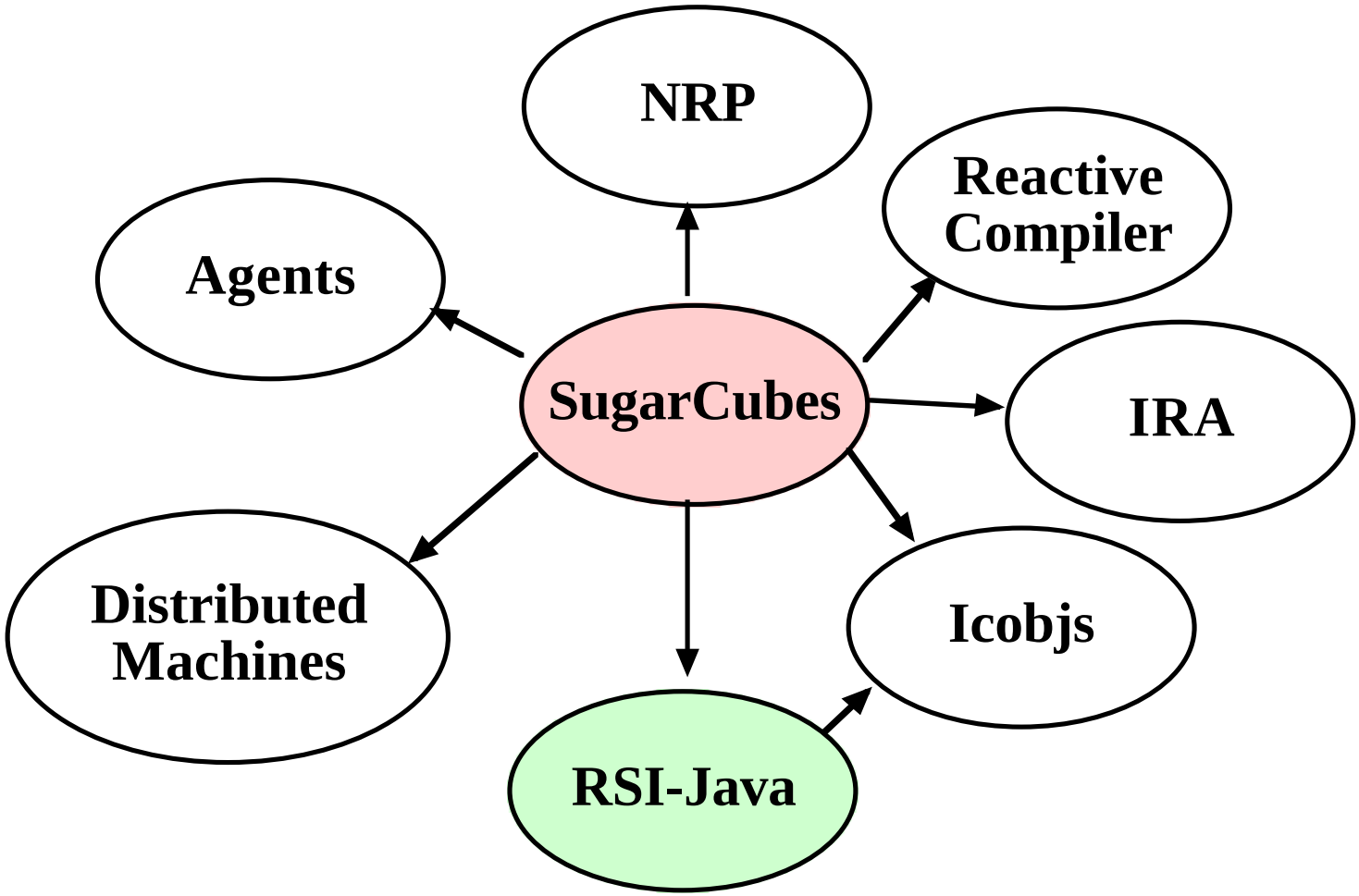


Historique

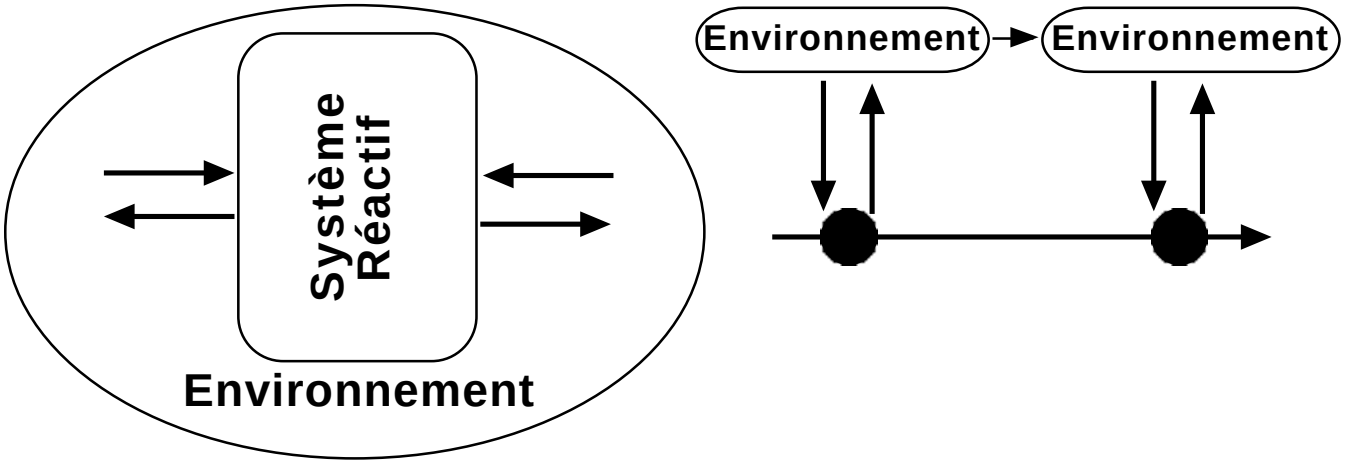
Pourquoi Java?

- **simplicité (Orienté Objet)**
- **API très riche**
- **portabilité**
- **support réseau (Socket, RMI, ORBs...)**

Historique


NRP
Reactive Compiler
Agents
SugarCubes
IRA
Distributed Machines
Icobjs
RSI-Java

Approche Réactive

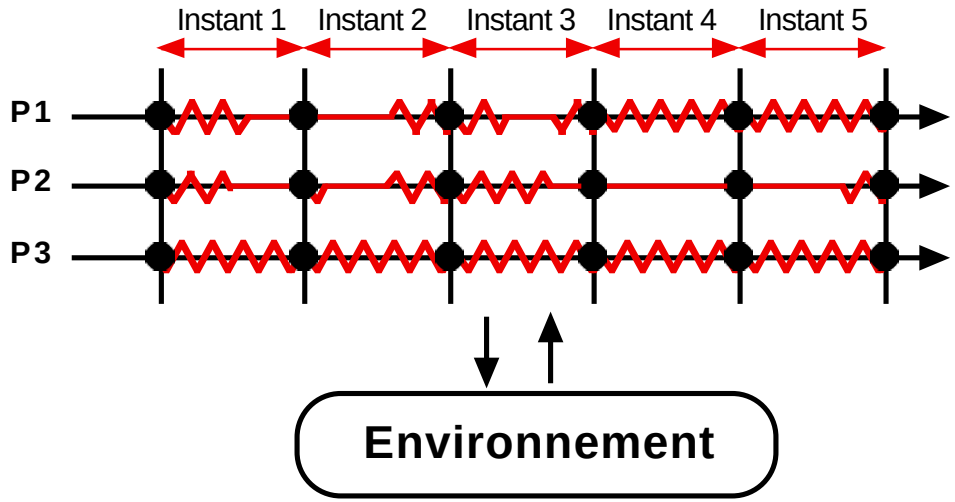


Approche Synchrone (Esterel) :

* **Parallélisme + instants**
* **Diffusion instantanée d'événements**

Approche Réactive :
pas d'hypothèse d'atomicité de la durée des instants

Approche Réactive



- l'instant a une durée et peut être découpé en micro étapes;
- l'environnement évolue au cours d'un instant (génération d'événements);
- l'absence d'un événement : décidée à la fin d'un instant
  - pas de réaction instantanée à l'absence;
  - pas de problèmes de causalité.

SugarCubes

Réactif + objets java =

Ensemble de classes et d'interfaces Java
Ancêtre de Junior

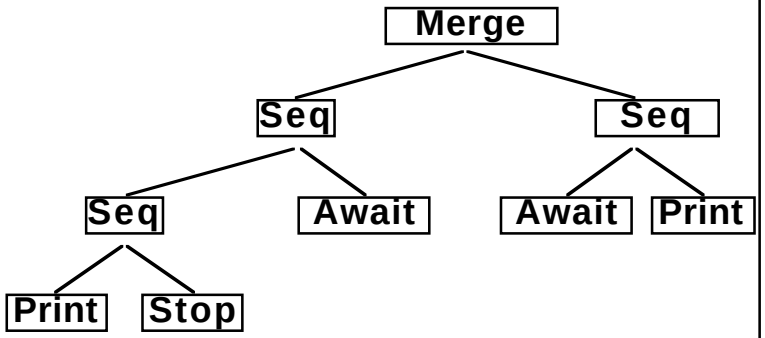
2 points de vue possibles :

- **Framework**
- **Langage de description de comportements**

SugarCubes

4 grands types d'objets :

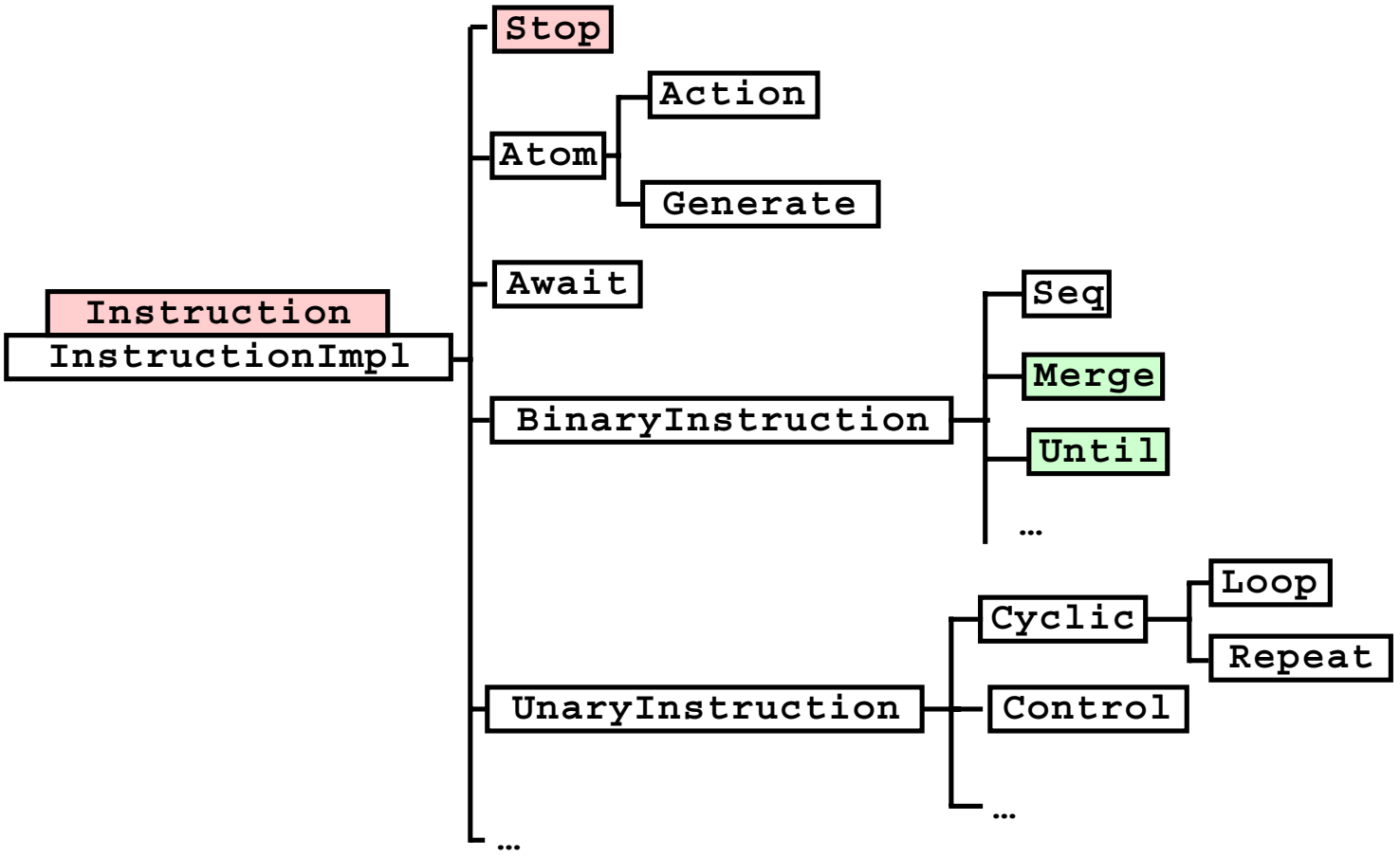
- Instructions réactives
- Classes événementielles
- Classes d'interfaçage avec Java
- Machines d'exécution réactives



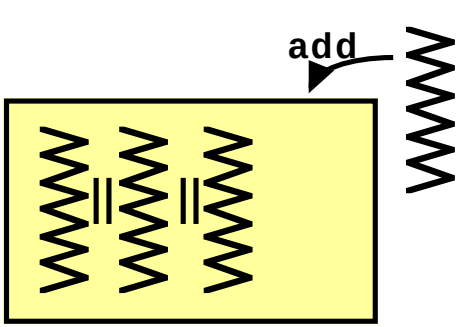

```
Program p = SC.merge(
    SC.seq(
      SC.print("HelloWorld")
      ,SC.stop()
      ,SC.generate("e")
    )
    ,SC.seq(
      SC.await("e")
      ,SC.print("e!")
    )
  );
```

Instructions Réactives

Les instructions réactives sont des objets java.

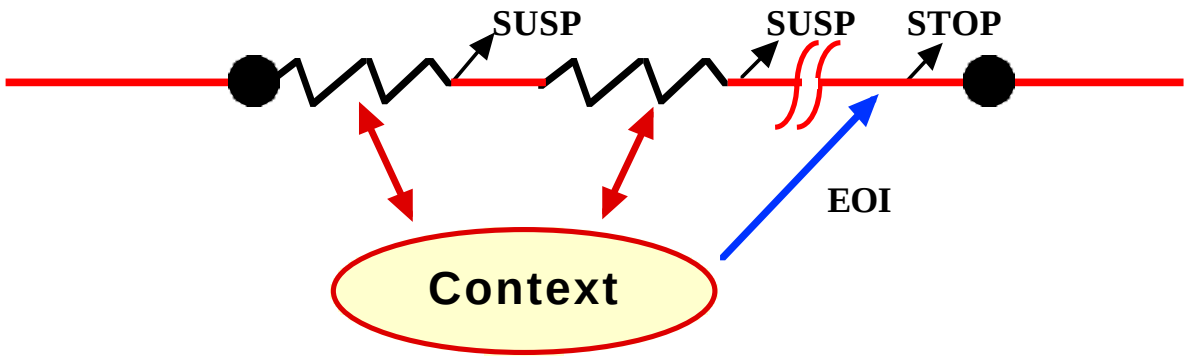


Machine d'exécution

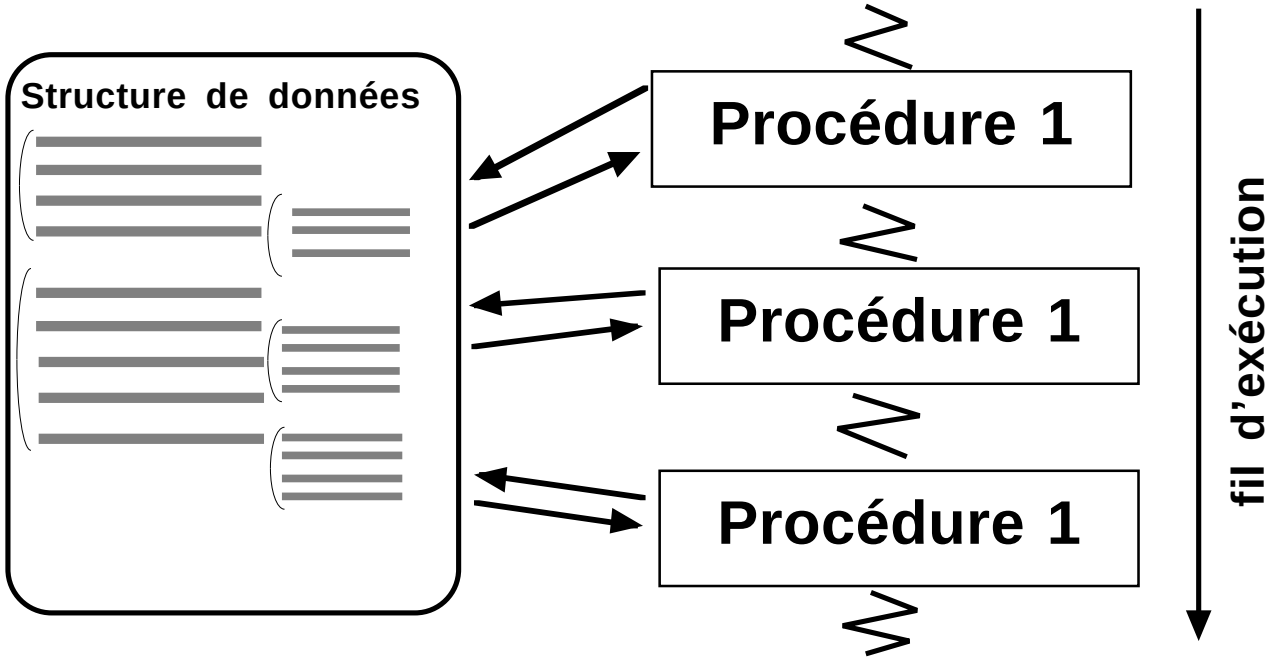



```
Machine m = SC.machine();

Instruction p = ......

m.add(p);
```

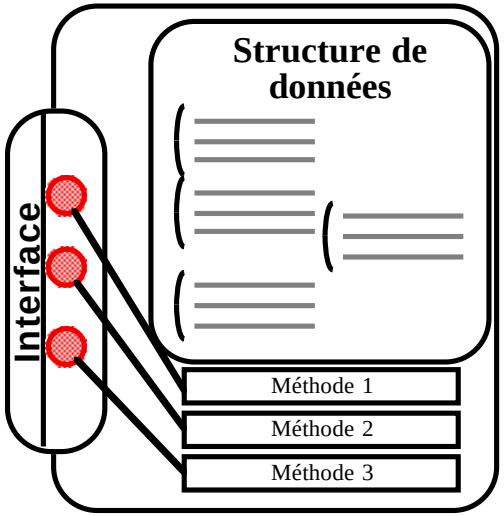
* Possède un programme à exécuter = instruction réactive,
* Fournit le contexte d'exécution des instructions réactives
  - événementiel
  - Java
  ...
* Décide la fin des instants.

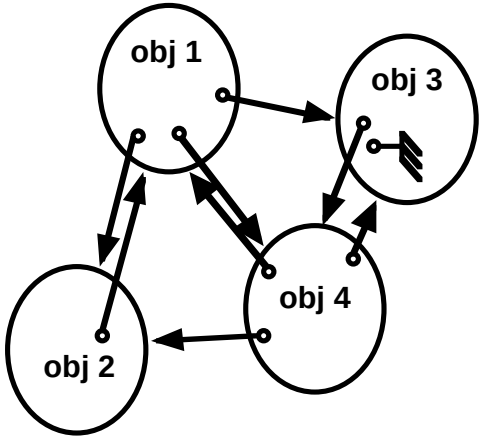


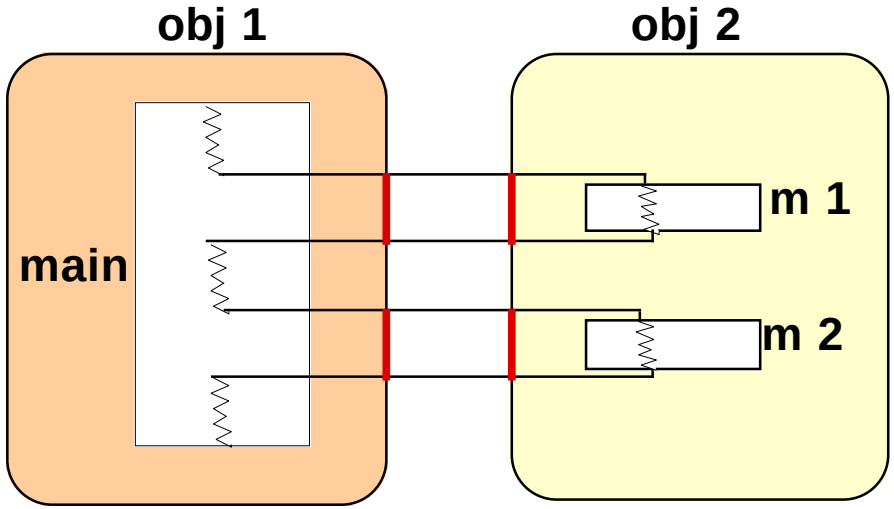
Programme


Structure de données
Procédure 1
Procédure 1
Procédure 1
fil d'exécution

Objets



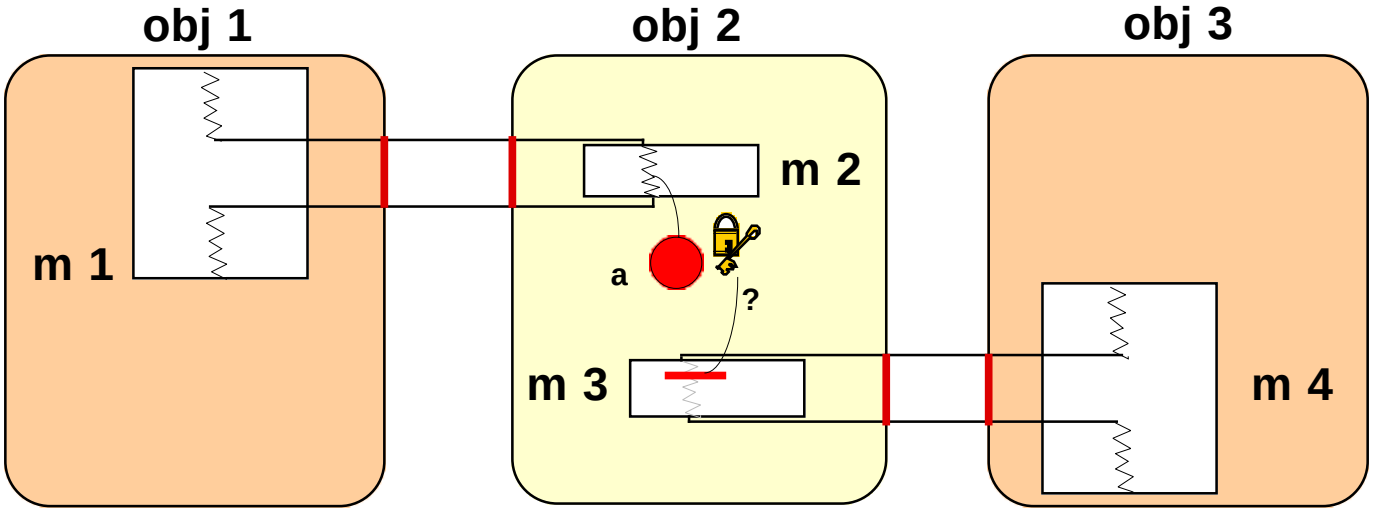




Le fil d'exécution passe le contrôle d'un objet à un autre au gré des appels de méthodes

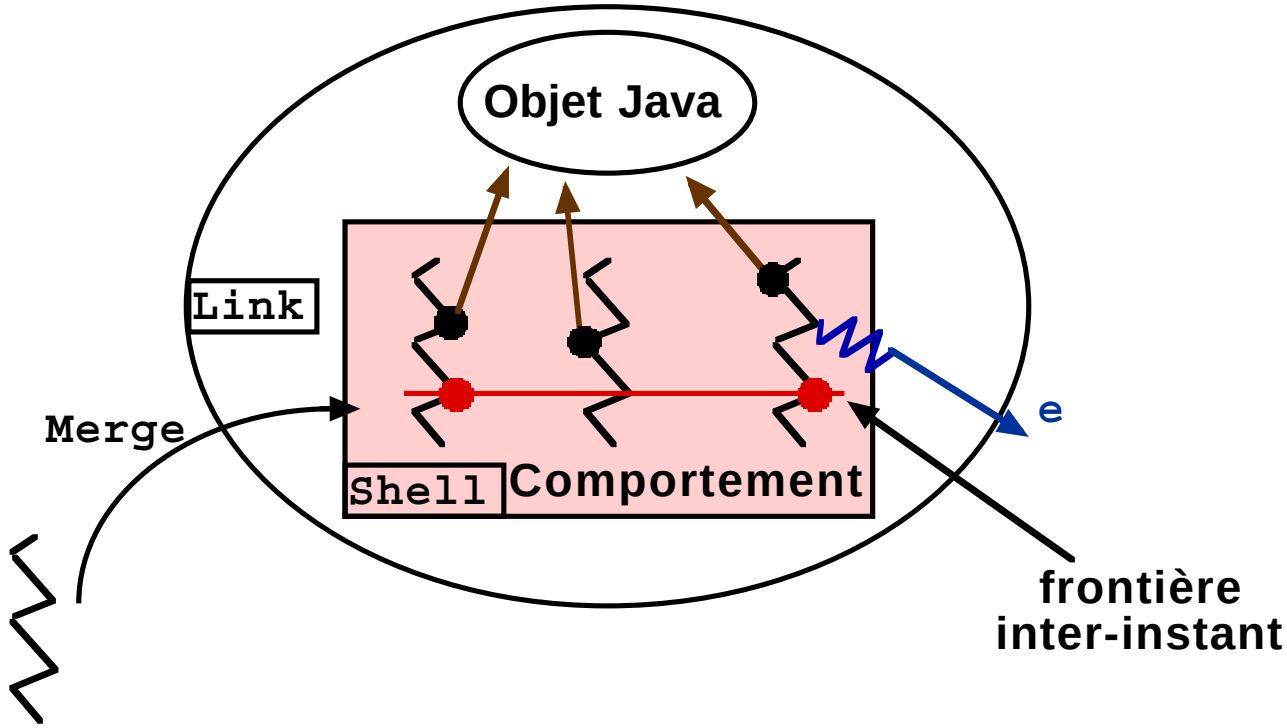
Objets et concurrence

contrôle d'exécution en multi-threads :

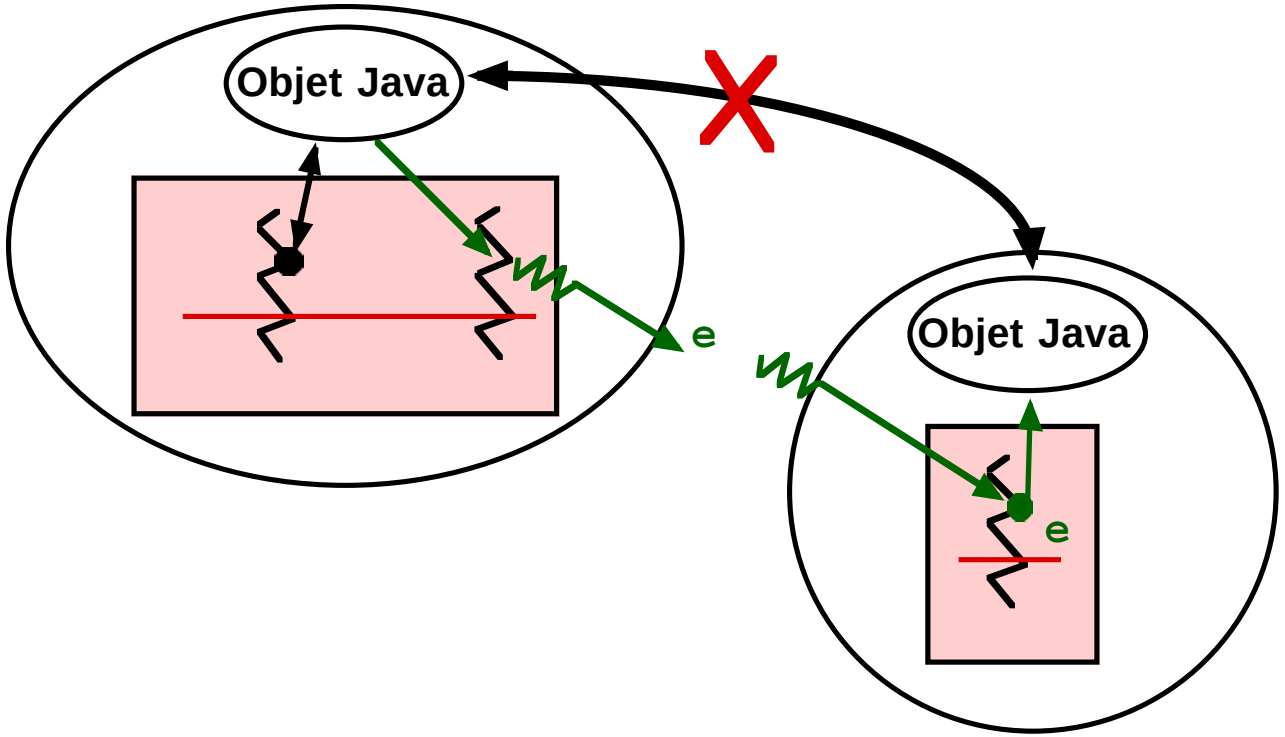


Lorsque plusieurs fils de contrôle d'exécution (Thread) aboutissent simultanément sur le même objet, on est obligé d'en tenir compte dans l'implémentation de cet objet en verrouillant l'accès à certains champs.
Cette situation, peut conduire à des deadlocks.

Cube


Objet Java
Link
Merge
Shell
Comportement
e
frontière
inter-instant

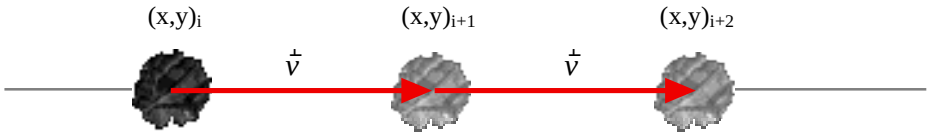
Cubes



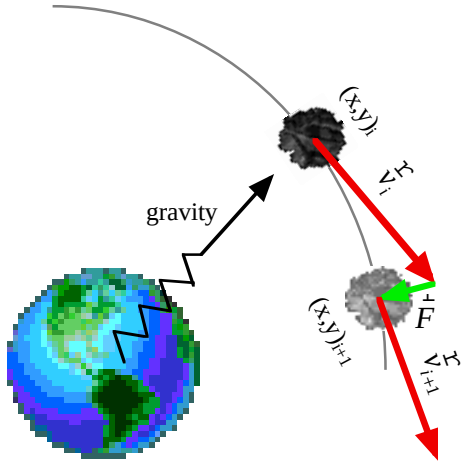
Les Cubes communiquent entre eux par événements valués.
Chaque occurence au cours d'un instant implique une réaction : exécution d'une opération atomique particulière.

Événements valués

Un objet réactif avec un comportement physique

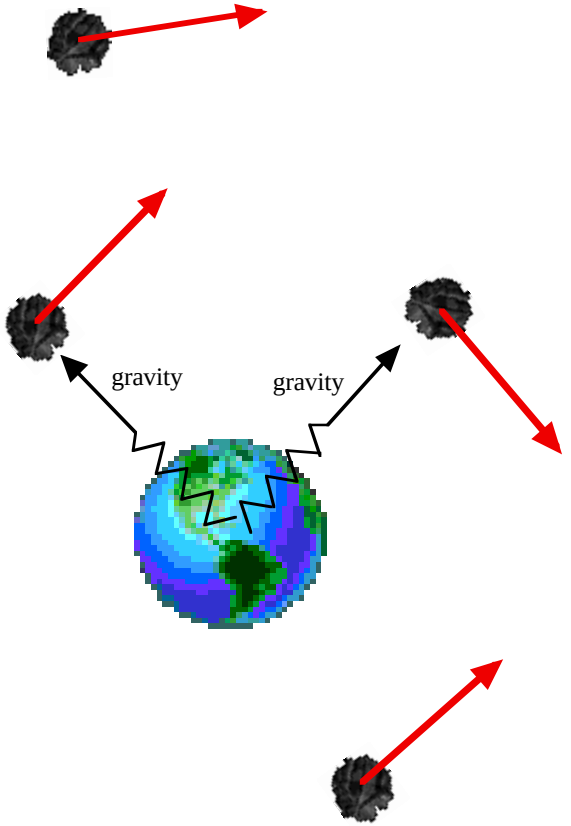


On ajoute en parallèle un objet réactif planète



Évenements diffusés

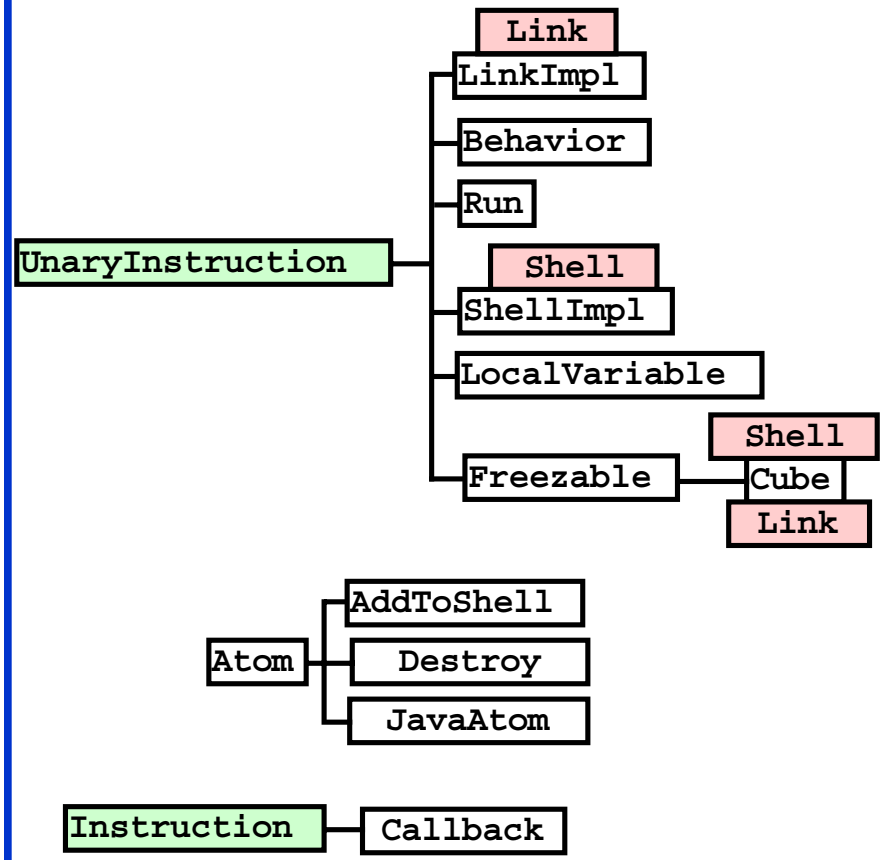
La diffusion instantanée des événements assure la cohérence



Plusieurs planètes peuvent générer l'événement "gravity", l'ensemble des occurrences sera reçu par tous les objets à l'écoute.

Intéraction SugarCubes-Java

Instructions :



Interfaces :

Link

Shell

à implémenter par l'utilisateur :

JavaAction

JavaBooleanExpression

JavaCallback

JavaStringExpression

JavaIntegerExpression

...

Intéraction SugarCubes-Java

création d'un Cube

objet Java associé au comportement

```
SC.cube("exemple",new Frame("exemple")
      ,SC.seq(
            SC.action(new InitFrame())
            ,SC.loop(
                  SC.seq(
                        SC.generate("DisplayHere",new GraphicsArg())
                        ,SC.stop()
                        )
                  )
            )
      ,new DisposeFrame()
      ,new DisposeFrame()
      ,new WarmupFrame()
      );
```

génération d'un événement valué

actions atomiques à exécuter à des moments critiques de la vie d'un cube:

- terminaison
- gel
- réveil

Interface Java

Action atomique

accès au contexte d'exécution

```
public interface JavaAction extends java.io.Serializable
{
    public void execute(Object self,LocalVariables vars,ReactiveEngine engine);
}
```

objet Java associé

variables locales

Évaluation d'expression atomique

```
public interface JavaIntegerExpression extends JavaExpression
{
    int evaluate(Object self,LocalVariables vars,ReactiveEngine engine);
}
```

l'évaluation atomique de cette méthode retourne un entier

Constantes

```
public final class JavaIntegerValue implements JavaIntegerExpression
{
    protected final int i;
    public JavaIntegerValue(int i){ this.i = i; }
    public final int evaluate(Object self,LocalVariables vars,ReactiveEngine engine){
        return i;
    }
}
```

Interface Java

Variables Locales

```
SC.localVariable("btn",theClass, ...);
```

Interface

```
public interface LocalVariables{
    public Object getVariableValue(String varName,Class expectedType)
                                                    throws NoSuchLocalVariable;
    public Object getVariableValueNoCheck(String varName)
                                                    throws NoSuchLocalVariable;
    public Object setVariableValue(String varName,Object value)
                                                    throws NoSuchLocalVariable;
}
```

Interface Java

Reactive Engine

```
public final class ReactiveEngine{
…
    public final void addProgram(Program aComponent){...}
    public final void generateEvent(Identifier eventName,Argument[] args){...}
    public final void addTask(Task t, boolean beforeReaction){...}
    public final void removeTask(Task t, boolean beforeReaction){...}
     public final Program getFrozenProgram(String instructionName){...}
     public final Identifier identifierFromString(String aString){...}
     public final Identifier cubeDestructionIdentifier(String aCubeName){...}
     public final String curentCubeName(){...}
}
```

Tâches inter instant

```
public interface Task
{
        public static final boolean AFTER_REACTION = false;
        public static final boolean BEFORE_REACTION = true;
        public void doIt();
}
```

Méthode appelée inter instant par la machine d'exécution

Callback réactive

Évaluation d'une callback réactive

```
public interface JavaCallback extends java.io.Serializable
{
   void execute(Object self,LocalVariables vars,ReactiveEngine engine
                                                ,Argument args[]);
}
```

Valeurs véhiculées par l'occurrence

Utilisation d'une Callback réactive

```
SC.cube("exemple",new DemoObject()
        ,SC.merge(
                SC.loop(
                        SC.action(new Move())
                        ,SC.stop()
                )
                SC.loop(
                        SC.callback("DisplayHere",new DisplayReaction())
                )
        )
);
```

on réagit à chaque instant aux éventuelles occurrences de l'événement DisplayHere

Scripts réactifs

Réactif + interprétation =


```
Program program =
    SC.merge(
        SC.seq(
            SC.await("e")
            ,SC.print("e!")
        )
        ,SC.generate("e")
    );
```

Sous forme de script réactif :

```
    (await e;{System.out.println("e!")})
||
    generate e
```

Scripts Réactifs

Les Cubes en RSI-Java :

```
object nom with {objet}
      corps
   methods
      m1
      m2
   on terminate
      {final_action}
   on freeze
     {freeze_action}
   on warmup
      {warmup_action}
end
```

opérations sur les objets : `freeze, destroy, send`

Exemple

associe un objet de type Frame au script réactif

```
object myWindow with {new java.awt.Frame()};
  await show;
  {this.setVisible(true)};
  halt

  on terminate {dispose()}
end.
```

action atomique exécutée lorsque l'objet réactif termine

Scripts Réactifs

Appels Externes :

- **invocation de méthodes publiques sur : (i)l'objet java associé au cube, (ii)l'un de ses champs publiques ou (iii)une classe publique**
- **invocation de constructeurs publiques (opérateur new)**
- **accès à la valeur d'un champ public d'un objet ou d'une classe publique**
- **affectation d'une valeur à un champ publique de l'objet ou d'une classe publique à l'aide de l'opérateur `=` .**
- **accès (lecture, affectation) aux variables locales**
- **gestion des types de base : byte, short, int, long, float, double, boolean, char, String, null.**
- **objet java associé référencé par `this`.**