OBJETS REACTIFS EN JAVA : JUNIOR

Approche réactive =

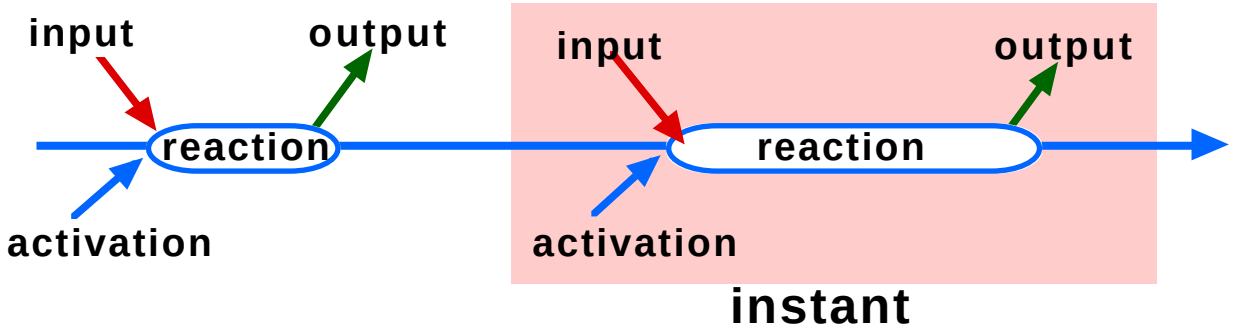
concurrence + instants + création dynamique

instants de durée non nulle

le nombre de composants peut changer durant l'exécution

- **Junior = approche réactive en Java**
- **Pas d'utilisation des threads Java**
- **Descendant des SugarCubes**
- **Sémantique formelle + implémentations efficaces**

Le modèle



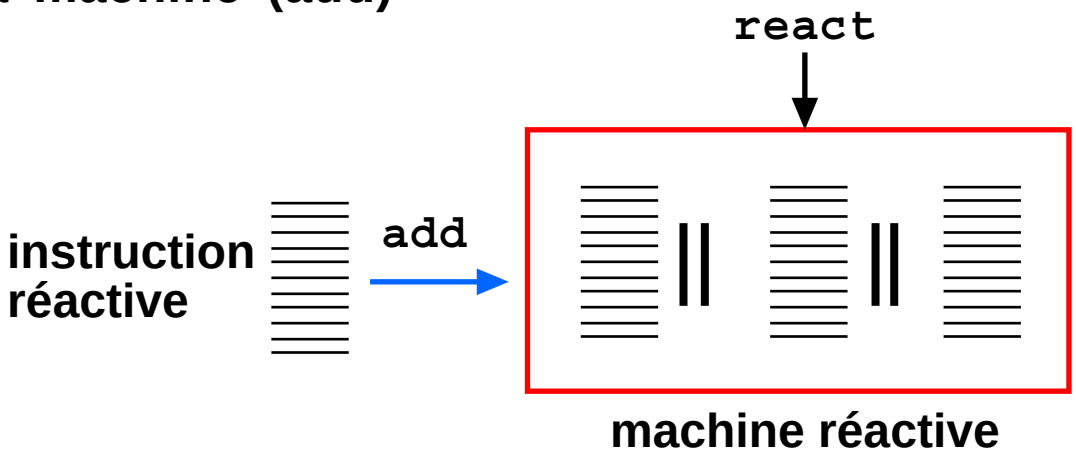
Communication à base d'événements diffusés instantanément

Les programmes réactifs réagissent aux activations : réaction = instant = tous les composants parallèles ont réagi et tous les événements ont été diffusés

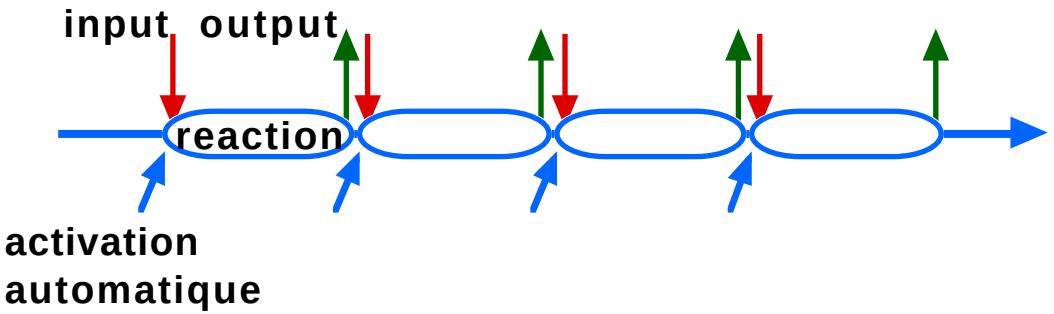
On ne décide qu'un événement est absent à un instant qu'à la fin de celui-ci

Programmation en Junior

- Ecrire un programme sous forme d'instruction réactive
- Déclarer une machine réactive
- Mettre le programme dans la machine (add)
- Activer la machine (react)



Un mode de fonctionnement possible :



Instructions réactives

Séquence : `Seq(inst,inst)`

Parallélisme : `Par(inst,inst)`

Instant : `Stop()`

Boucle infinie : `Loop(inst)`

Boucle finie : `Repeat(exp,inst)`

Atome : `Atom(Action)`

```
new Loop(
    new Seq(
        new Atom(new Print("hello")),
        new Stop()))
```

```
loop
  call Print()("hello");
  pause
end
```

Instructions événementielles

Evénement : `Event(event,inst)`

Génération : `Generate (event)`

Attente : `Await(event)`

```
new Seq(new Await("event"),new Atom(new Print("hello")))
```

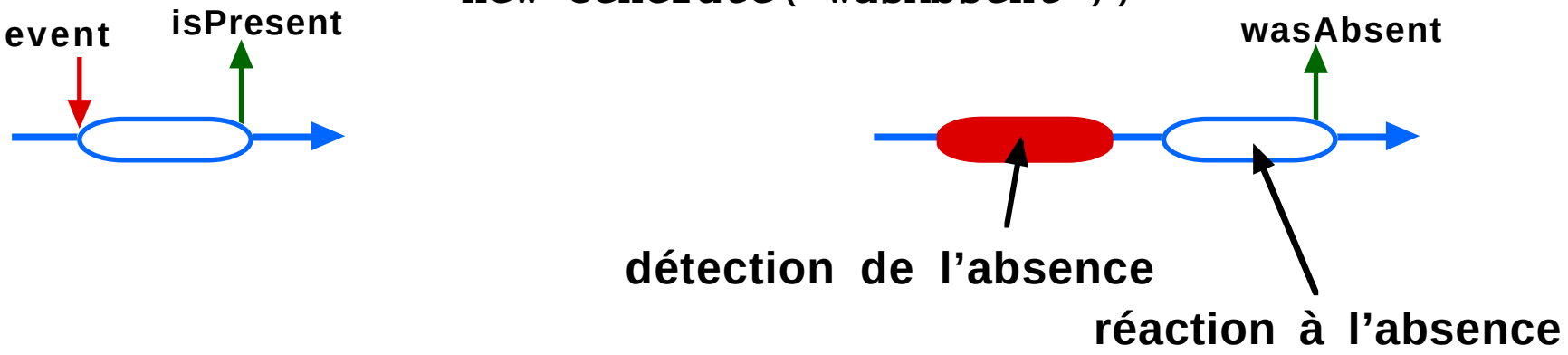
```
new Seq(
  new Par(new Await("event1"),new Await("event2")),
  new Generate("event3"))
```


```
(
  await immediate event1
||
  await immediate event2
);
emit event3
```

Test d'événement

Présence : When(event,inst,inst)

```
new When("event",
         new Generate("isPresent"),
         new Generate("wasAbsent"))
```



On ne sait qu'un événement est absent, qu'à la fin de l'instant ; pas de réaction instantanée à l'absence

```
new When("event",new Generate("isPresent"),
                 new Generate("event"))
```

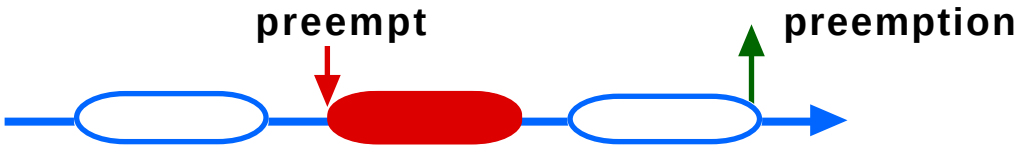
Préemption

Préemption : Until(event,body,handler)

```
new Until("preempt",
          new Loop(new Generate("go"),new Stop()),
          new Generate("preemption"))
```

```
weak abort
  loop
     emit go; pause
  end
when preempt do
   emit preemption
end
```

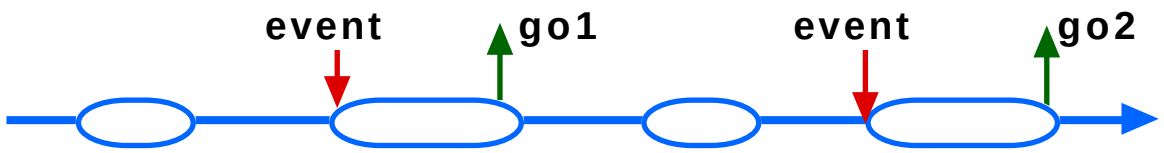

```
new Until("preempt",
          new Await("event"),
          new Generate("preemption"))
```



Filtrage par un événement

Filtrage : `Control` (event,inst)

```
new Control("event",
      new Seq(new Generate("go1"),
      new Seq(new Stop(),new Generate("go1"))))
```

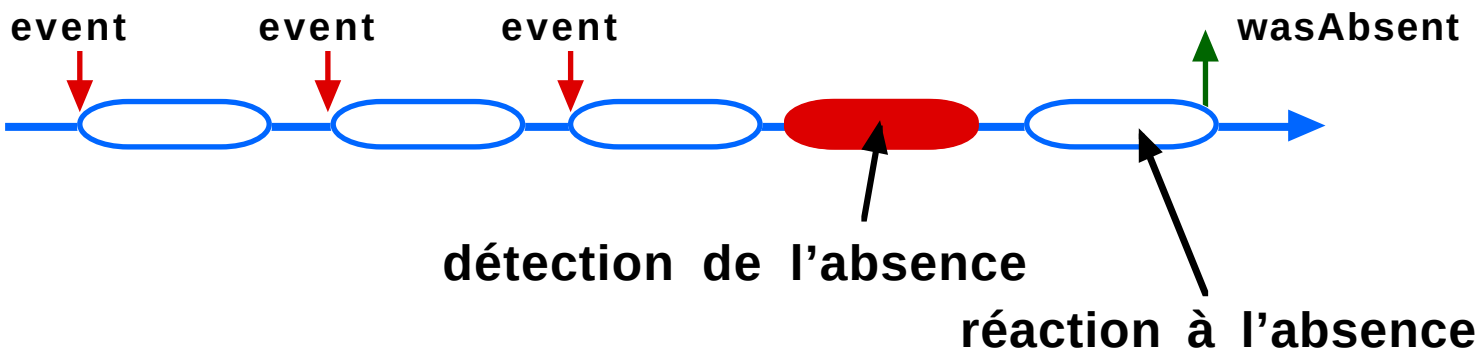


Configurations

Combinaison booléenne d'événements : And,Or,Not

```
new Await(new And("event1","event2"))
```

```
new Seq(
    new Await(new Not("event")),
    new Generate("wasAbsent"))
```

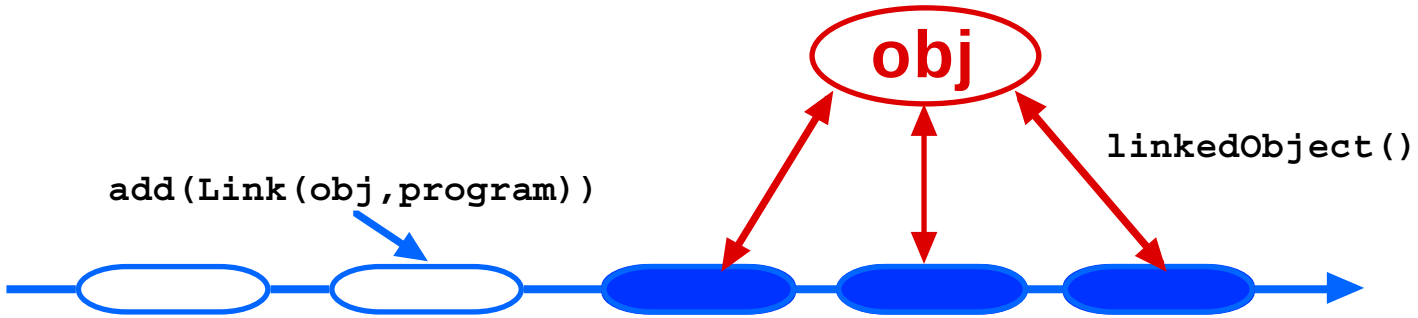


Interfaçage avec Java

Liens avec Java : `Atom(Action),Link(object,inst)`

```
new Link(obj,new Atom(action))
```

Lors de l'exécution de `action`, l'objet implicite est `obj` ; il est désigné par `linkedObject()`

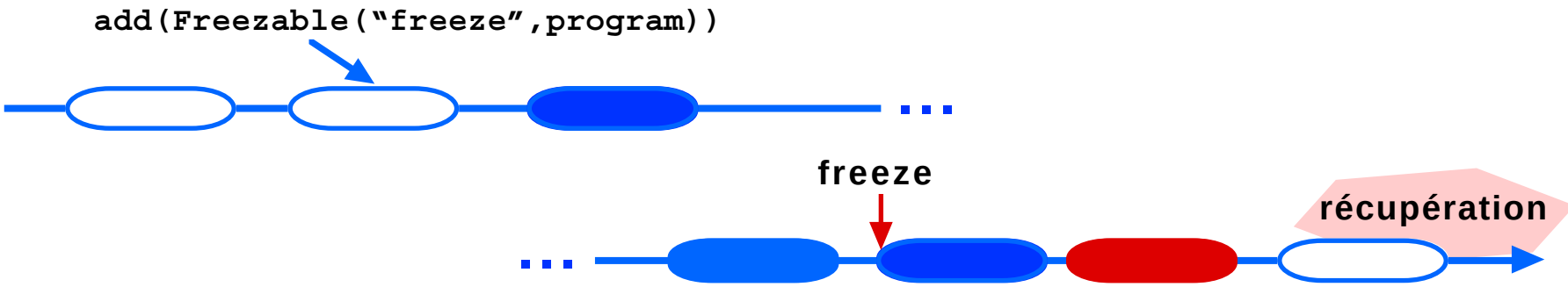


Gel d'instructions

Gel : `Freezable(event,inst), getFrozen(event)`

```
machine.add(new Freezable("freeze",program))

machine.add(
    new Seq(new Generate("freeze"),
    new Seq(new Stop(),
            new Atom(new GetFrozen("freeze")))))
```



Enlève `program` de la machine et récupère "ce qui reste à faire", sous forme d'une instruction

Evénements valués

Evénements valués : `Generate(event,Object)`

Environnement :

```
Object[] previousValues(event)
```

```
new Seq(new Generate("evt",obj1),
new Seq(new Generate("evt",obj2),
new Seq(new Stop(),
        new Atom(action))))
```

`previousValues("evt")` **retourne**
`[obj1,obj2]`

Jr = API Junior

```
import junior.*;
import junior.extended.*;

public class HelloWorld
{
   public static void main(String[] args){
     Machine machine = Jr.SyncMachine();
        machine.add(Jr.Atom(new Print("hello, world!")));
     for (int i = 0; i < 3; i++){
        System.out.print("instant "+i+": ");
        machine.react();
        System.out.println("");
     }
   }
}
```

Output:

```
instant 0: hello, world!
instant 1:
instant 2:
```

Jr.Program, Jr.Machine

les instructions réactives sont des programmes

```
public interface Program extends java.io.Serializable
{
    Program copy();
}
```

les instructions réactives ne sont pas réentrantes

```
public interface Machine
{
    boolean react();
    void add(Program program);
    void generate(Identifier event);
    void generate(Identifier event, Object val);
    Program getFrozen(Identifier event);
}
```

les événements sont désignés par des identifiers

les instructions réactive doivent être gelées avant d'être extraites de la machine

Programmes élémentaires

```
Program Jr.Nothing()
Program Jr.Stop()
Program Jr.Seq(Program first, Program second)
Program Jr.Par(Program left, Program right)
Program Jr.Loop(Program body)
```

Interfaçage avec Java

```
Program Jr.Atom(Action action)

public interface Action extends java.io.Serializable
{
    public void execute(Environment env);
}

public interface Environment
{
     Object[] previousValues (Identifier id);
     Object[] currentValues (Identifier id);
     Program getFrozen (Identifier id);
    Object linkedObject();
}
```

objet Java lié par une instruction Link

Wrappers

```
Program Jr.If(BooleanWrapper cond,Program thenInst,Program elseInst)
```

évalué au premier instant d'exécution, pas à la construction

boucles finies

```
Program Jr.Repeat(IntegerWrapper wrapper,Program body)
```

liaison d'un objet Java dans l'environnement

```
Program Jr.Link(ObjectWrapper wrapper, Program body)
```

génération d'un événement

```
Program Jr.Generate(IdentifierWrapper iwrap,ObjectWrapper owrap)
Program Jr.Generate(IdentifierWrapper iwrap)
```

génération avec valeur

wrappers constants

```
IdentifierWrapper Jr.ConstWrapper(Identifier id)
ObjectWrapper Jr.ConstWrapper(Object obj)
IntegerWrapper Jr.ConstWrapper(int n)
BooleanWrapper Jr.ConstWrapper(boolean b)
```

Réactions aux événements

configurations = combinaisons booléennes d'événements

```
Configuration Jr.Presence(IdentifierWrapper wrapper)
Configuration Jr.And(Configuration c1,Configuration c2)
Configuration Jr.Or(Configuration c1,Configuration c2)
Configuration Jr.Not(Configuration c)
```

attente d'une configuration

```
Program Jr.Await(Configuration config)
```

préemption

```
Program Jr.Until(Configuration config,Program body,Program handler)
```

test d'une configuration

```
Program Jr.When(IdentifierWrapper wrapper,Program thenInst,
                                          Program elseInst)
```

filtrage par un événement

```
Program Jr.Control(IdentifierWrapper wrapper,Program body)
```

gel d'une instruction

```
Program Jr.Freezable(IdentifierWrapper wrapper,Program body)
```

Facilités d'écriture

Evénements sous forme de Strings

```
Identifier StringIdentifier(String str)
Program Generate(String event,Object val)
```

Remplacements par Nothing

```
Program Until(Configuration config, Program body)
Program Until(Identifier event, Program body)
```

Remplacements de wrappers

```
Program Repeat(int count, Program body)
```

TopReq

```
Program eachTop =
   Jr.Loop(
     Jr.Seq(Jr.Await("Top"),
            Jr.When("Absent",
                    Jr.Seq(Jr.Generate("Alarm"),Jr.Stop()))));

Program eachReq =
   Jr.Loop(
     Jr.Par(Jr.Stop(), // Req et Top simultanés
            Jr.Until("Top",Jr.Seq(waitFirstReq(),
                                  Jr.Seq(Jr.Generate("Ok"),
                                  Jr.Seq(Jr.Stop(),replyOqp()))))));

Program waitFirstReq =
   Jr.Until("Req",
            Jr.Loop(Jr.Seq(Jr.Generate("Absent"),Jr.Stop())));

Program replyOqp =
   Jr.Loop(
     Jr.Seq(Jr.Await("Req"),
     Jr.Seq(Jr.Generate("Oqp"),Jr.Stop())));
```

Conclusion

Puissance expressive

instants, parallélisme, événements diffusés, dynamicité

Réutilisabilité

pas de problème de causalité

Sémantique formelle

micro-steps, relativement simple

Implémentations efficaces

sans threads, grand nombre d'événements et composants parallèles

mais

Interfaçage basique avec Java

Syntaxe lourde

Références

http://www.inria.fr/mimosa/rp/Junior

Objets réactifs en Java, F. Boussinot, Presses Polytechniques Universitaires Romandes, Collection Technique et Scientifique des Télécommunications, 2000

The Junior Reactive Kernel, L. Hazard, J-F. Susini, F. Boussinot, Rapport de recherche INRIA 3732, 1999 **(sémantique)**

Programming in Junior, L. Hazard, J-F. Susini, F. Boussinot, Rapport de recherche INRIA 4027, 2000 **(Jr)**