Le langage *oRis*

(Version 2.2 — 28 mars 2000)

Donner un comportement aux entités
(objets, objets actifs et ... agents)

Fabrice Harrouet
Laboratoire d'Informatique Industrielle
Ecole Nationale d'Ingénieurs de Brest
`harrouet@enib.fr`
`http://www.enib.fr/~harrouet/`

Propos

- ▷ Modéliser et ajuster un système interactivement
- ▷ Composants du système $\equiv$ entités autonomes
- ▷ Exprimer le rôle et le mode de fonctionnement de chaque composant
- ▷ Interagir avec le système modélisé en cours de fonctionnement
- ▷ Permettre au système d'être modifié ou de se modifier dans le temps
- ▷ Donner suffisamment d'expressivité pour traiter des domaines d'application variés

$$\Downarrow$$

Outil de prototypage, langage interprété hors ligne et en ligne

Notion d'objet, granularité instance

- ▷ Langage procédural
  - ◇ Des fonctions
  - ◇ Pas de variables globales !
- ▷ Langage à objets
  - ◇ Syntaxe proche de `C++`, `java`
  - ◇ Structure des instances $\equiv$ Classe
    - ○ Attributs, méthodes, héritage multiple
  - ◇ Pas de variables statiques !
- ▷ Granularité de la description au niveau **instance**
  - ◇ Attributs et méthodes particuliers à certaines instances
  - ◇ Liaison dynamique poussée jusqu'aux instances
  - ◇ Classe $\simeq$ comportement par défaut

Notion d'objet, restriction de l'accès

- Tout est `public` !
  - Pas génant pour le prototypage
- Signification de `public` en POO ?
  - Contrôle au niveau des classes, difficile de nuancer
- Contrôle au niveau des instances, mot-clef `that`
  - Exemple : `obj1` invoque `obj2->doIt()`
    - `doIt()` invoquée **sur** `this=obj2`
    - `doIt()` invoquée **par** `that=obj1`
  - Contrôle plus fin de l'accès
    - `if(that!=this) error("Illegal access");`
    - `if(!that->isA("allowedClass")) error("Illegal access");`
    - `that` est-il dans les accointances ?

Faciliter l'interaction avec les objets

- ▷ S'adresser facilement à un objet quelconque
  - ◇ Nommage des instances : `Avion.5` , `Predator.2` . . . `obj->name()`
- ▷ Outils d'introspection
  - ◇ Liste des classes : `getClasses()`
  - ◇ Liste des objets d'une classe : `getAll("Type")`
  - ◇ Relations d'héritage : `obj->isA("Type")` , `obj->className()`
  - ◇ Inspection et manipulation des classes et des instances
  - ◇ Inspection et manipulation du code
  - ◇ . . .
- ▷ Localisation, détection et visualisation 2D/3D

Notion d'objet actif

- ▷ Les objets ont leur propre unité d'exécution
- ▷ Ils ont un but, ils ne sont pas nécessairement pilotés de l'extérieur (c'est néanmoins possible)
- ▷ Objet actif = Objet + méthode `main()`
- ▷ Méthode `main()` $\equiv$ activité principale du comportement
- ▷ Plusieurs comportements en parallèle
- ▷ Plusieurs façons d'ordonnancer

Le moteur de simulation

- ▷ Animer les objets actifs et exécuter les divers traitements
- ▷ Unité d'exécution :
  - ◇ Méthode `main()` d'un objet actif
  - ◇ Bloc `start{...}` ou `A.1::start{...}` externe à tout bloc
  - ◇ Bloc `start{...}` dans un bloc de code (embranchement)
- ▷ Cycle de simulation
  - ◇ Cycle $\equiv$ exécution d'une **portion** de chaque unité d'exécution
  - ◇ Chaque unité d'exécution est désignée une fois et une seule au cours d'un cycle de simulation
  - ◇ Interruptible $\longrightarrow$ parallélisme

Différents types de multitâche

- Multitâche **coopératif**
  - Changement de contexte explicite
    (`yield()`, fin de `main()`, attente de ressource ...)
  - Paramétrer la simulation avec `setInterval(0); // default`
- Multitâche **préemptif**
  - Changement de contexte sur occurence d'un signal d'horloge
  - Paramétrer la simulation avec `setInterval(dt); // dt in ms`
  - `dt` significatif si supérieur aux tranches de temps allouées par le système sous-jascent ($\simeq$ 10ms)
- Multitâche **profondément parallèle**
  - Changement de contexte après chaque instruction élémentaire
    (debug $\rightarrow$ mise en évidence de sections critiques)
  - Paramétrer la simulation avec `setInterval(-1); // dt < 0`

Désignation des unités d'exécution

- Désignation ordonnée
  - Toujours le même ordre
  - Paramétrer la simulation avec `setPickingMode(serialPicking);`
  - Priorités indésirables entre les objets actifs en mode coopératif
- Désignation aléatoire sans remise
  - Un ordre différent à chaque cycle
  - Paramétrer la simulation avec
    `setPickingMode(randomPicking); // default`
  - Répartition plus équitable de l'opportunité de réagir

Sections critiques

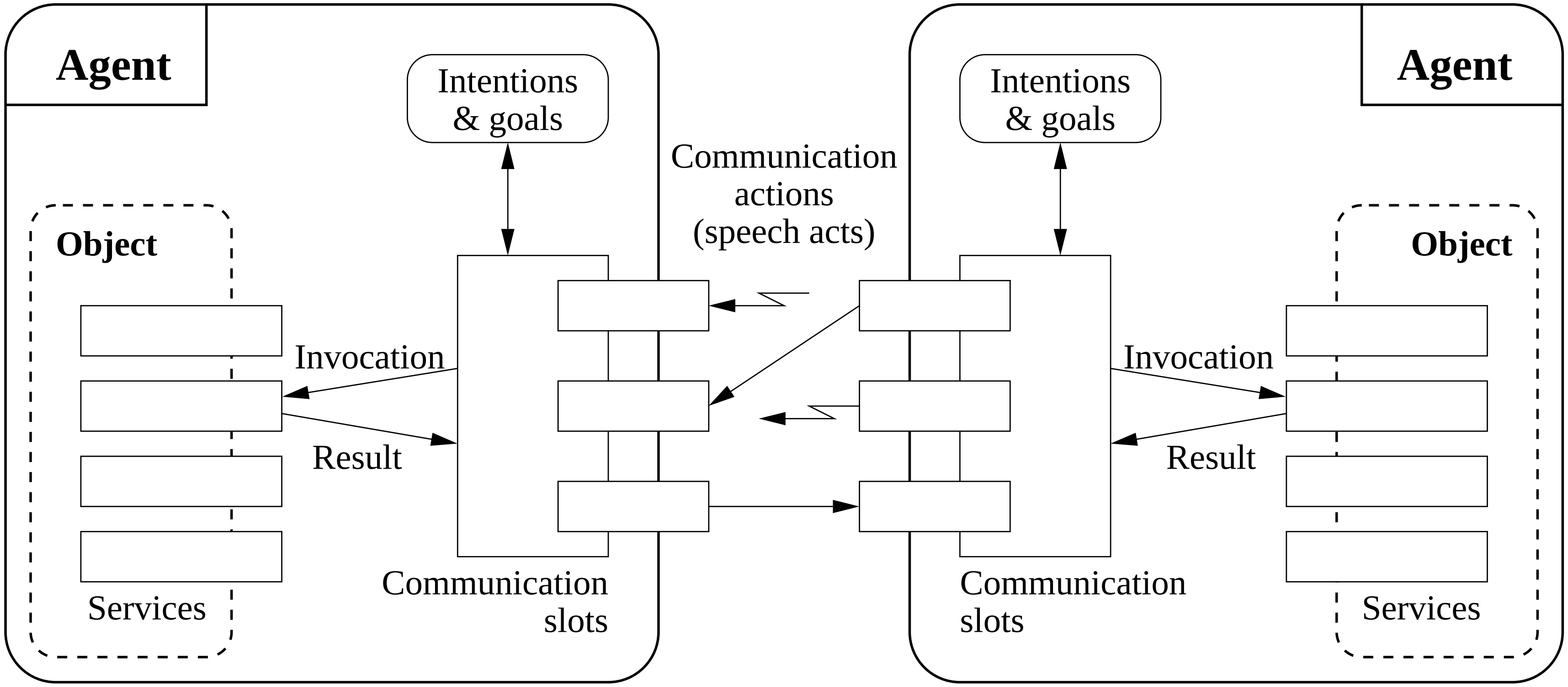
- Propos
  - Ne pas être interrompu par l'ordonnanceur
- Solution
  - Exécuter tout un bloc d'instructions comme s'il s'agissait d'une instruction élémentaire
- Moyen
  - Un bloc `execute{...}` ou `A.1::execute{...}` externe à tout bloc
  - Un bloc `execute{...}` dans un bloc de code
  - Les constructeurs et destructeurs
  - Méthodes *"réflexes"*
    - sur modification d'attribut
    - sur réception de message
    - sur événement d'IHM

Exclusion mutuelle

- ▷ Propos
  - ◇ Etre temporairement le seul à utiliser certaines ressources
- ▷ Solution
  - ◇ Utilisation de sémaphores d'exclusion mutuelle
- ▷ Moyen
  - ◇ Classe `Semaphore`
  - ◇ Initialisation variable de la ressource : `sem=new Semaphore(1);`
  - ◇ Méthode `lock()` : mise en attente de l'unité d'exécution si ressource indisponible
  - ◇ Méthode `tryLock()` : prise de la ressource si disponible et échec sinon
  - ◇ Méthode `unlock()` : libération de la ressource, elle redevient disponible

Notion d'agent

Communication par **actes de langage** (message $\neq$ invocation)



Notion d'agent

▷ Sémantique variée des messages :

◇ demandes d'exécutions, réponses

◇ demandes d'informations, informations

◇ appels d'offres, offres

◇ . . .

▷ Nécessite des moyens de communication variés

▷ Structure interne complexe (robustesse à l'échec et aux incohérences)

Communication entre objets

- ▷ Par défaut, communication synchrone
  - ◇ Invocations de méthodes : `obj->f();`
  - ◇ Lectures / écritures directes d'attributs : `obj->i=8;`
  - ◇ Temps de traitement pris sur le temps d'exécution de l'objet appelant
- ▷ Utilisation de liens réflexes : `new AttrCB(obj,"attr");`
  - ◇ Objet connecté à un attribut d'une instance (contrôle, bornage)
  - ◇ Déclenchement de méthodes avant et après modification de l'attribut
  - ◇ Possibilité de propager les modifications (détection des boucles)
  - ◇ L'origine de la modification importe peu, cohérence assurée malgré tout
  - ◇ Temps de traitement quasi-nul (callback ininterruptible) à la charge du lien réflexe lui-même

Communication entre agents

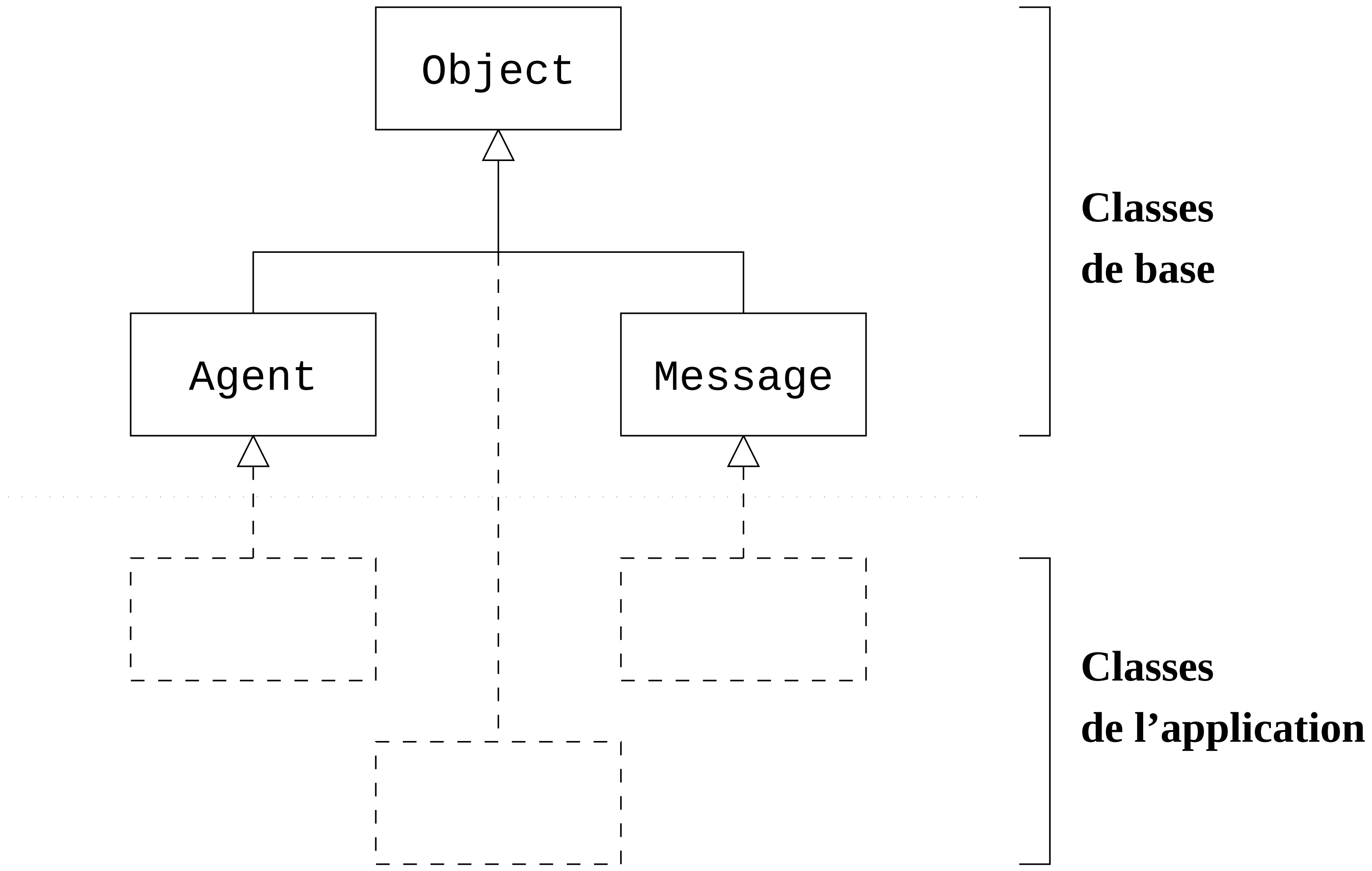
- ▷ Messages pouvant véhiculer une information quelconque
  - ◇ `Message msg=new Message;`
  - ◇ Information minimale : son émetteur, déterminé par `that`
- ▷ Communication point à point asynchrone
  - ◇ `msg->sendTo(agt);`
  - ◇ Destinataire désigné explicitement lors de l'envoi du message
  - ◇ Agents dotés de boîtes à messages et de méthodes de consultation
  - ◇ Temps de traitement pris sur le temps d'exécution de l'agent destinataire
  - ◇ Aucune garantie sur le traitement du message

Communication entre agents

▷ Communication par diffusion : `msg->broadcast();`

- ◇ Destinataires inconnus lors de l'envoi du message (événement)
- ◇ Agent sensibles ou non à certains types de messages (modifiable)
  `setSensitivity("evt","f"); cancelSensitivity("evt");`
- ◇ Réaction aux messages par déclenchement de la méthode appropriée chez chaque agent sensible
- ◇ Temps de traitement quasi-nul (callback ininterruptible) à la charge du destinataire
- ◇ Possibilité de traitement asynchrone, placement automatique dans la boîte à messages
  `setSensitivity("evt","asynchronousReception");`

Les classes de base d'*oRis*



Un langage dynamique

- Langage interprété pour :
  - Intervenir dans le déroulement de la simulation
  - Ajuster le fonctionnement de la simulation
- Introduction de code à tout instant par :
  - Une chaîne de caractères
  - Un fichier
  - Une saisie dans une fenêtre de dialogue
  - Une trame réseau
- Forme du code strictement identique à celle du script initial
- L'intégralité du langage *oRis* est disponible en ligne

Un langage dynamique

- ▷ Interactions avec les objets (créations, destructions, invocations . . .)
- ▷ Surdéfinition en cours de simulation
  - ◇ De fonctions
  - ◇ De méthodes au niveau des classes
  - ◇ De méthodes au niveau des instances (granularité instance)
- ▷ Ajout en cours de simulation
  - ◇ De nouvelles fonctions
  - ◇ De nouvelles classes
  - ◇ De nouveaux attributs et méthodes au niveau des classes
  - ◇ De nouveaux attributs et méthodes au niveau des instances (granularité instance)

Intégration de code compilé

▷ Ajouter du code compilé en `C/C++` ($\simeq$ plugin)

◇ `loadLibrary("myLibrary");`

▷ Définir des classes compilées (associées à des classes *oRis*)

◇ Correspondance entre instance *oRis* et instance `C++`

▷ Passage dynamique interprété $\leftrightarrow$ compilé

◇ La déclaration (prototype) est la même dans les deux cas

```
void f(void); // declaration
```

◇ Le mécanisme de surdéfinition ne modifie que la définition (distinction *interprété/compilé* à ce niveau)

```
void f(void) {...} // definition
native void f(void); // definition
```

$\Rightarrow$ Améliorer les performances, **Intégrer d'autres développements**

Fonctionnalités diverses

▷ Ramassе-miettes automatique contrôlable

- ◇ Objets transitoires détruits dès qu'ils ne sont plus reférencés

```
Object obj=new Object; // new :  transitoire
Object obj=NEW Object; // NEW : non transitoire
obj->setTransient(true); obj->isTransient();
```

▷ Types de base

- ◇ `int` : entier signé 32 bits
- ◇ `float` : réel double précision
- ◇ `string` : chaîne de caractères (dynamiquement extensible)
- ◇ *NomDeClasse* : référence sur une instance (instanciation dynamique)
- ◇ *Type*`[]` : tableau (dynamiquement extensible)
- ◇ Tous ont une valeur logique : `0 0.0 "" NONE []` (faux)

Fonctionnalités diverses

▷ Parser de constantes

◇ `string str=valueDump(data);`

◇ `valueParse(data,str);`

▷ Opérateurs

◇ `= := ++ += + -- -= - *= * /= / %= %`

◇ `== :== != :!= <?  >?  < > <= >= !  && || ?:`

◇ `~ &= & |= | ^= ^ <<= << >>= >>`

▷ Opérateurs de tableaux

◇ `t.length() t.insert(d) t.pushFront(d) t.pushBack(d)`
`t.erase() t.remove(i) t.popFront() t.popBack()`
`t[i] t.front() t.back() t.sort()`
`t.seekMin() tab.seekMax() t.seek(d) t.deepSeek(d)`

Fonctionnalités diverses

- Opérateurs sur instances
  - `bool Object::equalsTo(Object obj)`
    - Utilisé par : `:== :!= deepSeek`
  - `bool Object::lessThan(Object obj)`
    - Utilisé par : `< <= > >= <? >? seekMin seekMax`
  - `void Object::affect(Object obj)`
    - Utilisé par : `:=`
- Chaînes de caractères
  - Tiennent sur plusieurs lignes (`\n` inclus)
  - Opérateurs
    - `+ += = == != < <= > >= <? >? s.length() s1.seek(s2)`

Fonctionnalités diverses

- Entrées sorties
  - `println(e1,e2,...); str=format(e1,e2,...);`
  - `str=readln();`
  - `warning(e1,e2,...); error(e1,e2,...);`
- Macros
  - `include "stuff.pkg"`
  - `beginPackage STUFF_PKG` ... `endPackage`
  - `replace PRIVATE by if(that!=this) error("Illegal access");`
- Commentaires
  - `// Le reste de la ligne`
  - `/* Les commentaires /* peuvent etre */ imbriques */`

Packages

- ▷ `include "attrCB.pkg"`
  - ◇ Lien réflexe attachés à un attribut d'une instance
  - ◇ `AttrCB`
- ▷ `include "browser.pkg"`
  - ◇ Fenêtres graphiques d'inspection et d'interaction avec l'application
  - ◇ `GlobalBrowser ClassBrowser ObjectBrowser ModuleBrowser` (+ variantes)
- ▷ `include "contain.pkg"`
  - ◇ Conteneurs génériques typés ($\simeq$ STL)
  - ◇ `Container Vector Stack Set Map`
- ▷ `include "control.pkg"`
  - ◇ Interaction avec la fenêtre de contrôle principale
  - ◇ `OrisControl`

Packages

- `include "convert.pkg"`
  - Convertions et operations sur les types
  - `Converter IntConv FloatConv StringConv ObjectConv`
- `include "fcm.pkg"`
  - Cartes cognitives floues
  - `Concepts GraphicConcepts Fcm LearnFcm CorrelationFcm HebbianFcm`
- `include "files.pkg"`
  - Lecture et écriture dans des fichiers textes et binaires
  - `InputFile TextInputFile BinaryInputFile OutputFile TextOutputFile BinaryOutputFile`

Packages

- `include "fuzzy.pkg"`
  - Contrôleurs flous
  - `FuzzyLing FuzzySet FuzzyVariable TNorm TCoNorm Inference Combination FuzzyController FuzzyFcmController` (+ variantes)
- `include "fuzzyBrowser.pkg"`
  - Fenêtres graphiques d'inspection des contrôleurs flous
  - `FuzzyBrowser FuzzySetBrowser FuzzyVarBrowser` (+ variantes)
- `include "grapher.pkg"`
  - Fenêtres graphiques de tracé de courbes (XLib)
  - `GuiGrapher Grapher` (+ variantes)

Packages

▷ `include "gui.pkg"`

- Composants d'interface graphique (XLib)
- `GuiElem GuiArea GuiScrolledArea GuiHScrolledArea GuiVScrolledArea GuiColorArea GuiWindow GuiFrame GuiTabGroup GuiTab GuiMenu GuiMenuBar GuiMenuItem GuiMenuSepar GuiLabel GuiEditLabel GuiButton GuiToggle GuiToggleGroup GuiOptionMenu GuiCursor GuiScroller GuiList GuiEditor GuiTextDialog GuiButtonDialog GuiFileChooser GuiColorChooser`

▷ `include "ipc.pkg"`

- Communication inter-processus
- `IPCMemory IPCMutex`

Packages

- ▷ `include "java.pkg"`
  - ◇ Utilisation de code Java
  - ◇ `JavaInterface JavaObject`
- ▷ `include "mailRead.pkg"`
  - ◇ Fenêtre graphique de visualisation des messages
  - ◇ `MailReader` (+ variantes)
- ▷ `include "maths.pkg"`
  - ◇ Fonctions et constantes mathématiques
- ▷ `include "object2d.pkg"`
  - ◇ Objets graphiques 2D (XLib) pouvant se percevoir et être visualisés
  - ◇ `Object2d GuiViewer2d Viewer2d` (+ variantes)

Packages

▷ `include "object3d.pkg"`

◇ Objets graphiques 3D (OpenGl) pouvant se percevoir et être visualisés

◇ `Object3d GuiViewer3d Viewer3d` (+ variantes)

▷ `include "prolog.pkg"`

◇ Utilisation d'un interpréteur Prolog

▷ `include "reflect.pkg"`

◇ Inspection du code de l'application

◇ `AttributeDescription ModuleDescription ObjectDescription ClassDescription GlobalDescription TypeDescription`

▷ `include "remote.pkg"`

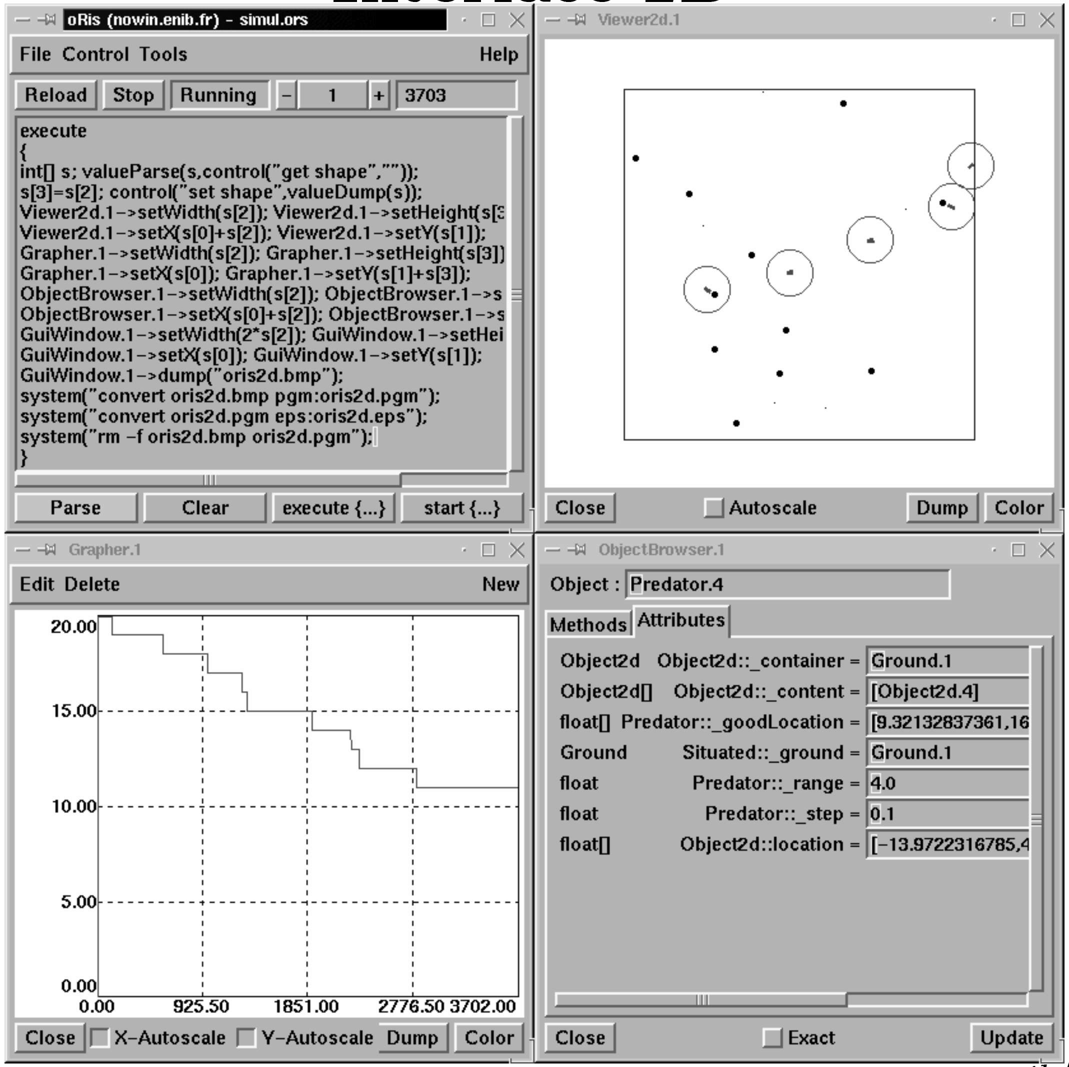
◇ Contrôle d'une session oRis distante

◇ `RemoteServer RemoteClient RemotePinger RemotePanel` (+ variantes)

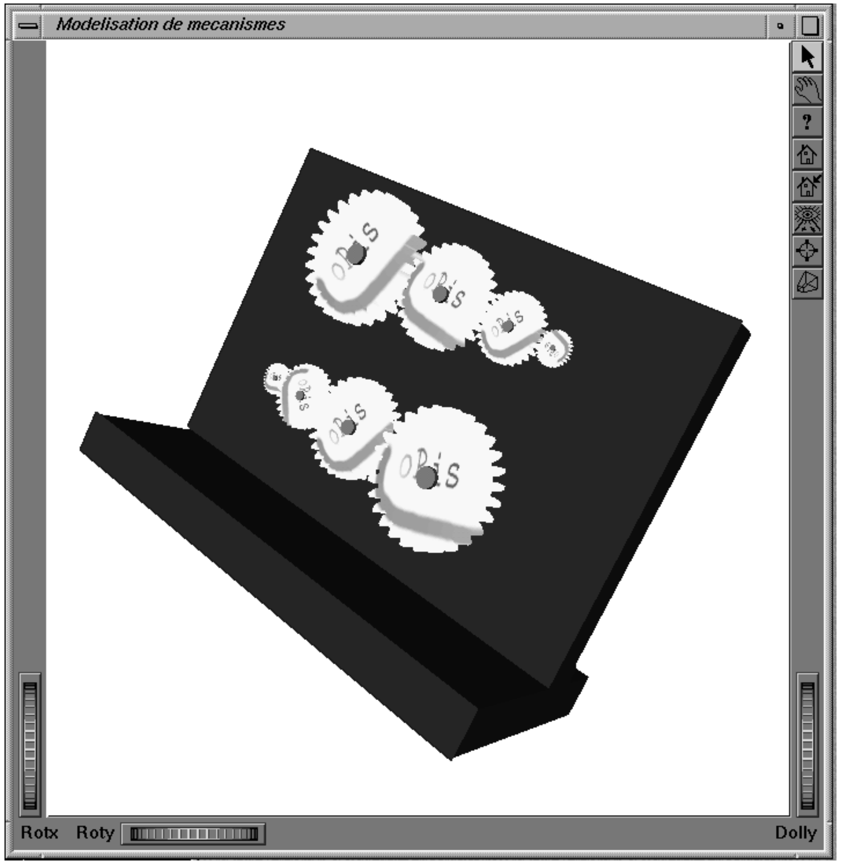
Packages

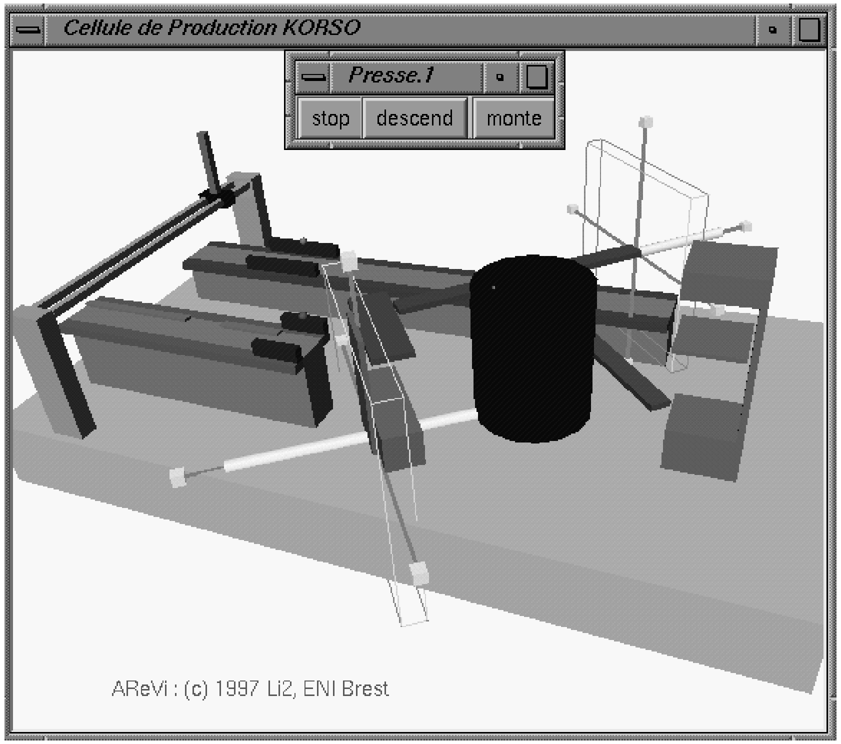
- ▷ `include "semaphor.pkg"`
  - ◇ Sémaphores d'exclusion mutuelle
  - ◇ `Semaphore`
- ▷ `include "tcpUdp.pkg"`
  - ◇ Communication par socket en TCP/IP et UDP/IP
  - ◇ `TCPConnection TCPListener UDPBroadcaster`
- ▷ En cours de finalisation
  - ◇ Logique temporelle
  - ◇ Modèle PAC

Interface 2D



Intégration dans *ARéVi*





Utilisation

- ▷ Modélisation de méta-populations animales (carabes)
- ▷ Optimisation des communications dans un réseau informatique
- ▷ Cellule de production (emboutissage) KorSo
- ▷ Simulation de réponses immunitaires
- ▷ Traitement d'images
- ▷ Ethologie (bancs de poissons, prédation)
- ▷ Robotique
- ▷ Support d'enseignement (POO et SMA)

Travaux en cours

- ▷ Prototypage interactif **et coopératif** :
  - ◇ Une même session pour plusieurs utilisateurs sur plusieurs machines
  - ◇ Maintien de la cohérence par routage des messages (instances répliquées $\longrightarrow$ instances réelles)
  - ◇ Mise à jour de la visualisation par *"dead reckoning"*
  - ◇ Pas de centralisation des objets (simulation distribuée)
- ▷ Spécification de plus haut niveau pour les agents
  - ◇ Utilisation du formalisme KQML
  - ◇ Règles d'écriture
  - ◇ *"Boîtes"* de comportement