

Introduction à la simulation : principaux concepts

Ecole Thématique CNRS de Porquerolles :
Modélisations et simulations multi-agents de systèmes complexes
pour les Sciences de l'Homme et de la Société :
principes et méthodes de conception et d'usage

<http://perso.univ-rennes1.fr/denis.phan/PorquerollesXAgents/>

E. RAMAT

Laboratoire d'Informatique du Littoral - Calais

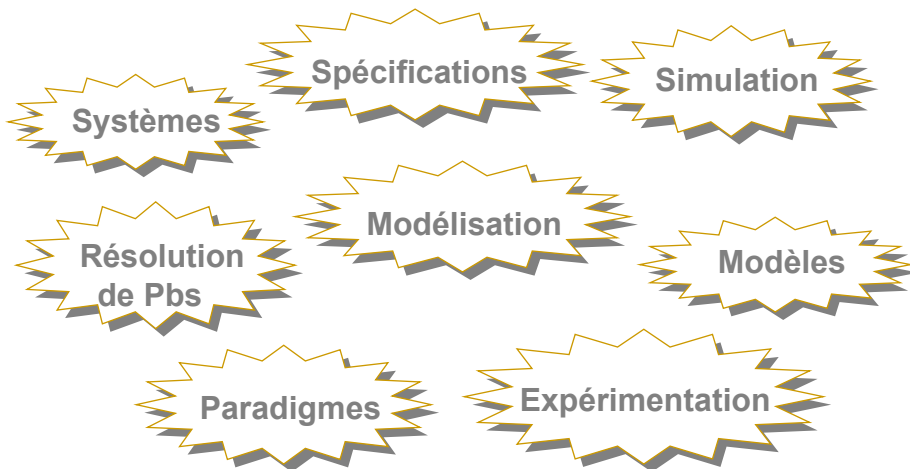
ramat@lil.univ-littoral.fr

Plan

- Introduction : pourquoi la simulation ? Le trio : système - modèle - schéma d'expérimentation
- Simulation à temps discret vs simulation à événements discrets
- DEVS : un *framework* pour la spécification à événements discrets
- La partie de l'aléatoire dans la simulation : la question des générateurs aléatoires
- La simulation distribuée : problématique et techniques

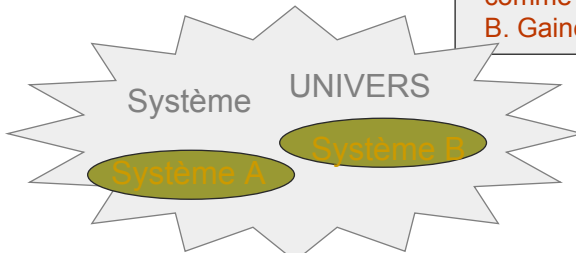
Introduction à la simulation

Introduction à la Simulation



Qu'est ce qu'un système ?

' Un système est ce que l'on distingue
comme étant un système '
B. Gaines

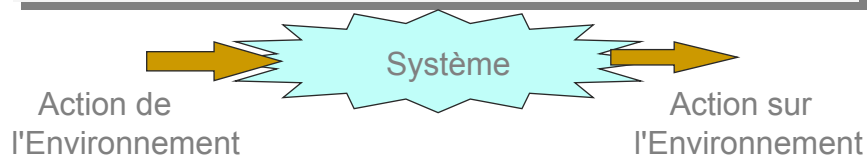


Selon cette définition, un système est caractérisé par le fait que l'on sait distinguer ce qui lui appartient de ce qui ne lui appartient pas

Qu'est ce qu'un système ?

'Un système est une source potentielle de données'
B. Zeigler

Le système est ici supposé contrôlable et/ou observable.
Des variables, générées par l'environnement, agissent sur
le comportement du système qui, à son tour, réagit sur cet
environnement.

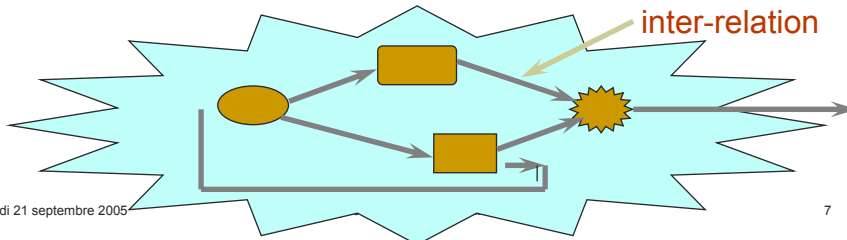


Qu'est ce qu'un système ?

Un système réel est une combinaison de un ou plusieurs éléments structurels inter-reliés tels que :

- les états d'un élément sont influencés par ses propres états et ceux des autres éléments. [Laux]

Cette définition met en avant l'aspect d'inter-influence des composants du système réel dont le comportement observé est, en partie, le résultat de ces influences



Mercredi 21 septembre 2005

7

Qu'est ce qu'un schéma d'expérimentation ?

'Une expérimentation est un processus par lequel on récolte des données sur un système en agissant sur ces entrées' - F. Cellier

Un schéma d'expérimentation définit un ensemble limité de circonstances sous lesquelles est conduite l'expérimentation:

- variables observées,
- séquencements des entrées,
- conditions initiales,
- conditions d'arrêt,
- collecte et codage des données.

Mercredi 21 septembre 2005

Introduction à la simulation : principaux concepts

8

Qu'est ce qu'un modèle ?

" Un modèle M d'un système S pour une expérimentation E est toute chose à laquelle on peut appliquer E pour répondre à des questions concernant S " -
M. Minsky

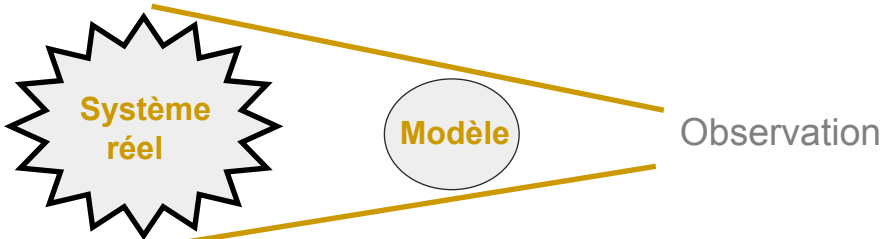
Un modèle est une forme intelligible d'un système construite pour permettre de trouver une réponse à un problème précis

**Un système muni d'un schéma
d'expérimentation est donc déjà
un Modèle**

Qu'est ce qu'un modèle ?

On ne peut résoudre un problème directement sur un système réel
Nous ne Raisonons que sur des Modèles.

Quel que soit le système considéré, nous n'en percevons
que le modèle que nous nous en sommes fait



Objet d'une modélisation

Construire une représentation simplifiée et observable du comportement et / ou de la structure d'un système réel afin de résoudre un problème d'analyse ou de conception.

La représentation simplifiée du système réel est un **MODELE**.

Qu'est ce qu'effectuer une Modélisation ?

" La modélisation est un processus par lequel on organise les connaissances portant sur un système donné " - B. Zeigler

Cette organisation est élaborée par l'intermédiaire d'un paradigme pour permettre de résoudre un problème donné sur le modèle.

→ Toutes les disciplines de la **science** et de l'**ingénierie** sont concernées par la modélisation. Le modèle est l'**outil de base de la résolution de problèmes**.

Qu'est ce qu'effectuer une Modélisation ?

Les scientifiques construisent des modèles pour comprendre l'univers

Activité d'Analyse

Les ingénieurs construisent des modèles pour modifier l'univers

Activité de Conception

Modéliser

(1)

- "Modéliser" c'est abstraire de la réalité une description d'un système dynamique.
- La modélisation a pour but d'apporter une représentation simple à utiliser décrivant le système.
- La modélisation est comme un langage dont l'objectif est de décrire des systèmes selon divers niveaux d'abstraction (ou points de vue).
- Les modèles sont utilisés pour mieux communiquer ou pour mieux comprendre le système.

[Modéliser (2)]

- Un modèle nous permet d'utiliser des "interfaces", des métaphores accessibles par tout le monde (équations, graphes, diagrammes...).
- Le langage naturel n'est pas un outil adéquat (trop imprécis et ambiguë).
- Il faut toujours se demander à quelles questions on désire répondre.
- Il est nécessaire de se fixer des limites et de définir un monde fermé.

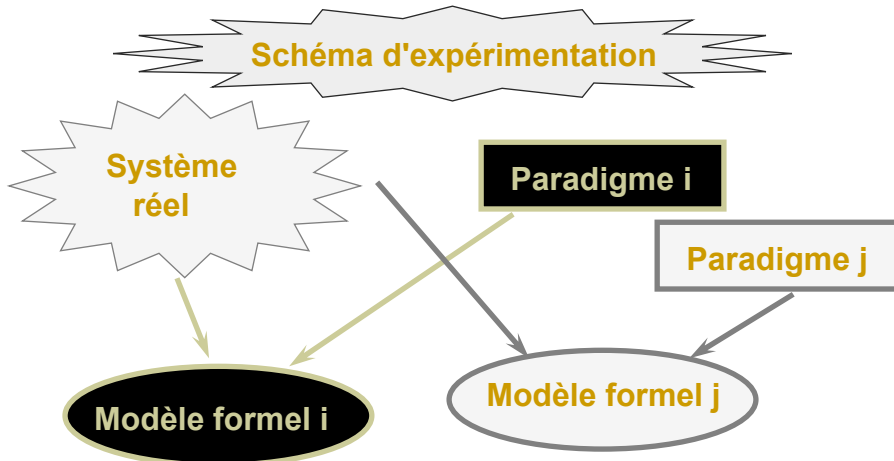
[Qu'est ce qu'un paradigme ?]

Un paradigme est un ensemble de concepts, de lois et de moyens visant à définir une collection de modèles.

Un langage de programmation algorithmique est un paradigme dans lequel un programme est le modèle d'une activité séquentielle.

Le symbolisme 'portes logiques' est un paradigme dans lequel un schéma est un modèle d'un circuit digital.

Système, Modèles et Paradigmes



Un système réel est-il ?



synchrone,
asynchrone,
continu,
discret,
séquentiel,
parallèle,
déterministe
aléatoire
.....

Qu'est ce que la simulation ?

La simulation est la reproduction du comportement **dynamique** d'un système **réel** s'appuyant sur un **modèle** afin d'arriver à des **conclusions applicables au monde réel**.

La simulation a pour objet d'observer le comportement en fonction du temps d'un modèle d'un système.

La Simulation informatique

Ordinateur



On s'intéresse à des modèles **"exécutables"** de systèmes



descriptions intelligibles et formelles sur lesquelles pourront s'exécuter des **algorithmes** en un **temps fini**

Construction d'un Modèle simulable

La construction d'un modèle simulable concerne 3 éléments:

ANALYSE

- le système réel,
- le modèle,
- l'ordinateur.

CONCEPTION

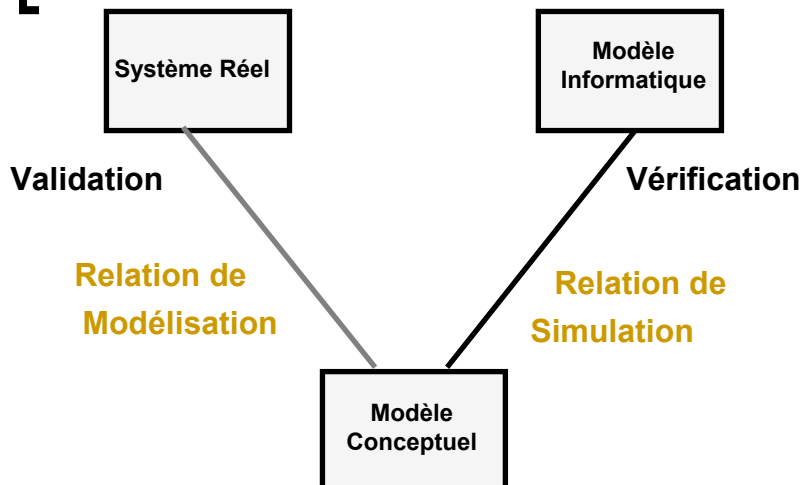
- le cahier des charges,
- le modèle,
- l'ordinateur.

Il faut établir entre ces trois éléments les relations:

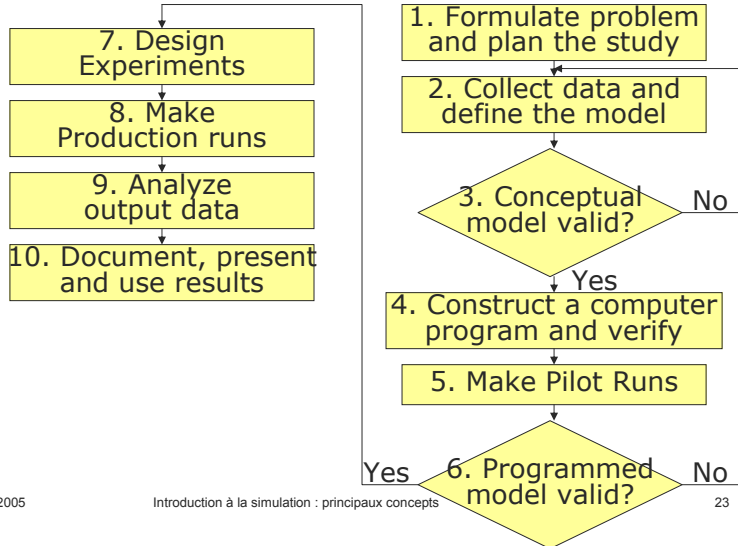
Modélisation = relation entre le système réel ou le cahier des charges et le modèle,

Simulation = relation entre le modèle et l'ordinateur

Modélisation - Simulation



Modélisation - Simulation



Modélisation - Simulation

Simulation informatique



Discretisation
du temps
ou
de l'espace
ou
des deux



Différentes classes de modèles simulables

Modélisation - Simulation

Que doit-on représenter ?

- **L'état du système** : l'ensemble des variables d'états qui décrivent le système à un instant t .
- **L'horloge** : la variable qui représente la valeur courante de l'avancement du temps dans la simulation.
- **La liste des événements** : une liste contenant les événements à venir avec leur date d'occurrence.
- **Les mesures** : les variables utilisées pour stocker les informations statistiques du modèle simulé.
- **L'initialisation** : une procédure qui initialise l'ensemble des variables d'états du système.
- **La procédure d'avancement du temps** : trouver le prochain événement et mettre à jour l'horloge.
- **La gestion des événements** : chaque type d'événement possède son propre procédure de traitement qui doit être exécuté lors de son occurrence.

Catégories de Modèles et de Spécifications

1ère catégorisation: par rapport à la **base de temps**

Modèle à temps continue, le temps est spécifié comme évoluant de manière continue, le temps est un nombre réel.

Modèle à temps discret le temps avance par sauts d'une valeur entière à une autre, le temps est un entier.

Catégories de Modèles et de Spécifications

2ème catégorisation: par rapport aux ensembles de **valeurs des variables** descriptives du modèle.

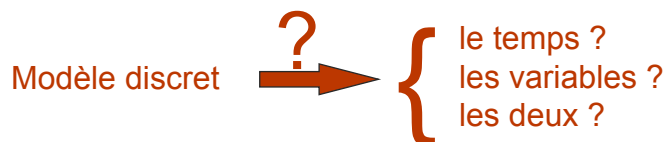
Modèle à états discrets: les variables prennent leurs valeurs dans un ensemble discret,

Modèle continue: les variables descriptives sont des nombres réels,

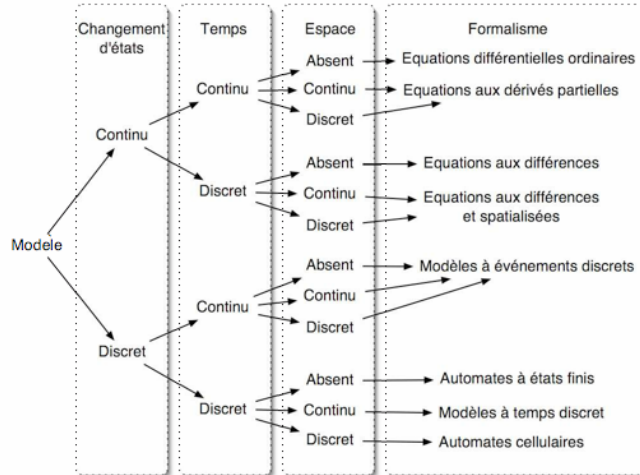
Catégories de Modèles et de Spécifications

Il existe de nombreuses autres catégorisations

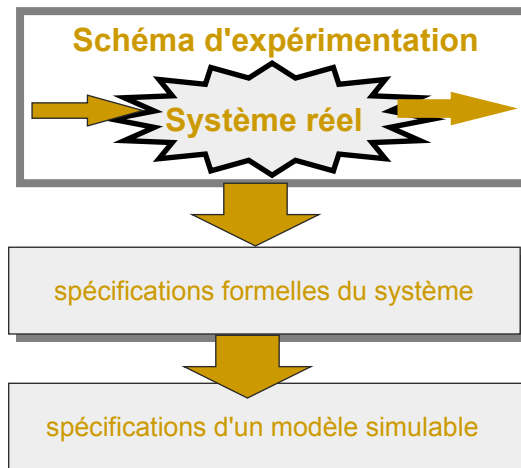
Les catégorisations par rapport **au temps et aux variables** sont indispensables pour situer un modèle de simulation



Formalismes



Spécifications formelles



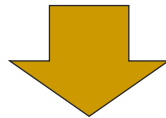
Spécifications formelles



Exemple :

La spécification par équations différentielles est :

- à variables continues
- à temps continu

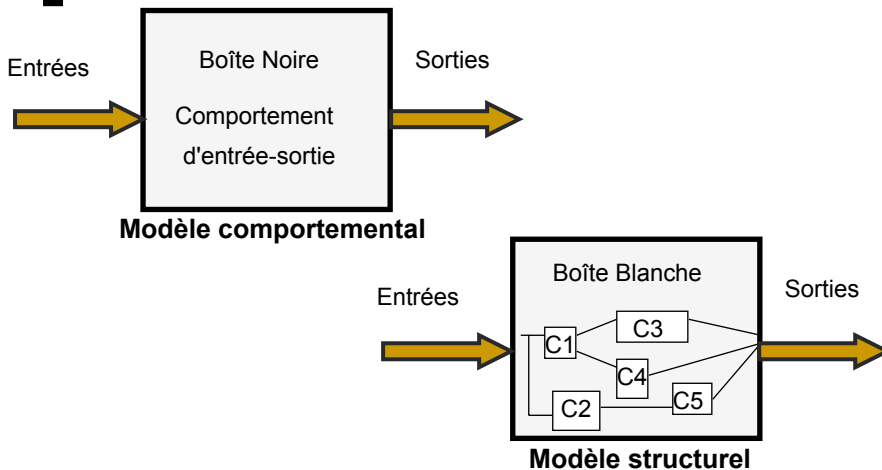


Simulation informatique

Discrétisation du temps

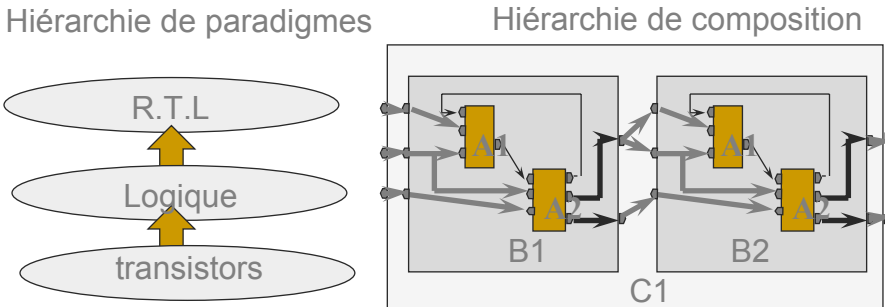
Approximation par méthodes de calculs numériques d'intégrales

Comportement et Structure



Description Hiérarchisée

Hiérarchie de composition → Facilité de description
 Hiérarchie de paradigmes → Changement de niveau pour résoudre un pb



Mercredi 21 septembre 2005

Introduction à la simulation : principaux concepts

33

Modèles simulables

Représentations possibles selon :

A - Deux Vues

Comportementale

Structurelle

B - Deux hiérarchies

hiérarchie de paradigmes

hiérarchie de description

plusieurs niveaux d'abstraction plusieurs niveaux de description

Mercredi 21 septembre 2005

Introduction à la simulation : principaux concepts

34

Résumé

Système réel → Système dynamique → Spécif. Simulable



Paradigme de spécification avec **Concept de ports**

↓
Description Modulaire et Hiérarchisée

Approches de la spécification de systèmes

Les différents niveaux de spécification

- Définition d'un modèle conceptuel
 - Indépendance vis à vis du formalisme spécifiant la dynamique du système
- Plusieurs niveaux [zeigler2000]:
 - Niveau 0 : schéma d'observation
 - Niveau 1 : observation de la relation entrée-sortie
 - Niveau 2 : observation de la fonction d'entrée-sortie
 - Niveau 3 : système dynamique
 - Niveau 4 : système couplé

Niveau 0 : schéma d'observation

Un schéma d'observation est une structure

$$O = \langle T, X, Y \rangle$$

avec: T: Base de temps,
X: ensemble des valeurs d'entrée,
Y: ensemble des valeurs de sortie.

T : est un ensemble d'entiers (temps discret) ou de réels (temps continu).

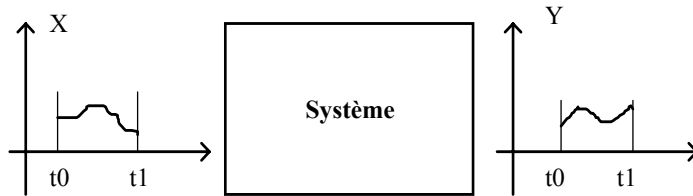
X : Le système est vu comme étant soumis aux éléments de X

Y : est un ensemble représentant l'interface au travers de laquelle le système influe sur son environnement.

X et Y sont des sous-ensembles quelconques d'entiers, de réels ou de symboles

A ce niveau, on définit les frontières du système en définissant ses entrées-sorties et la nature des variables et du temps qui seront utilisés.

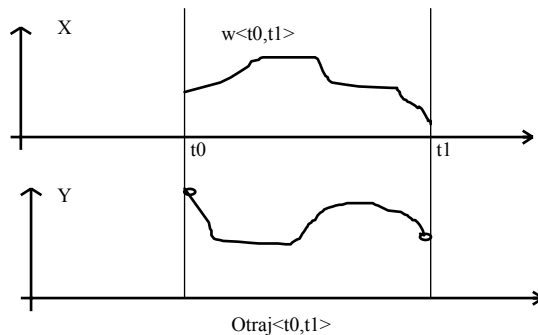
Niveau 0 : schéma d'observation



Niveau 0 : schéma d'observation

On définit un segment de sortie ou trajectoire de sortie:

$$O_{traj,w} : \langle t1, t2 \rangle \dashrightarrow Y$$



Niveau 1 : Observation de la relation d'entrée-sortie

Une Observation de la relation d'entrée-sortie IORO est une structure:

$$\text{IORO} = \langle T, X, \Omega, Y, R \rangle$$

avec :

$\langle T, X, Y \rangle$ schéma d'observation,

Ω ensemble des segments d'entrée,

R est la relation d'entrée-sortie qui définit le comportement du système,

$$R = \{(\omega_i, \text{otraj}_i)\}$$

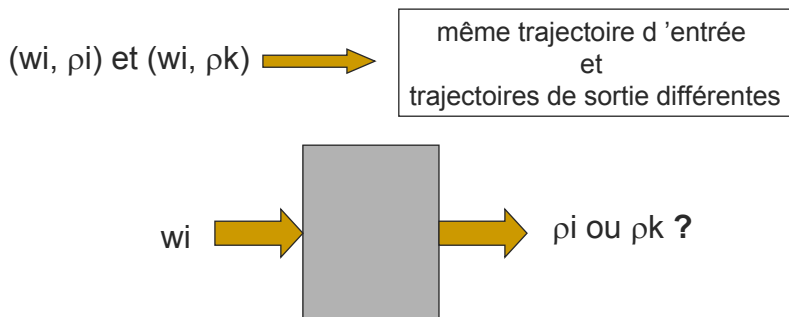
On note ρ les trajectoires de sorties.

Tout couple (ω, ρ) est appelée paire de segments d'Entrée/sortie,

ω et ρ étant observé sur le même intervalle de temps.

Niveau 1 : Observation de la relation d'entrée-sortie

Cette spécification est non-déterministe



La spécification est ambiguë ou encore incomplète

Niveau 2 : Observation de la fonction d'entrée-sortie

Une Observation de Fonction d'E/S, IOFO, est une structure:

$$\text{IOFO} = \langle T, X, \Omega, Y, F \rangle$$

F est l'ensemble des fonctions d'E/S

Par rapport au niveau précédent, la relation R est partitionnée en un ensemble F de fonctions f_i , chaque f_i définissant une réponse unique pour un segment d'entrée. En fait, chaque fonction f_i est définie pour un état initial donné.

A ce niveau, on possède une connaissance de l'état initial du système car tout segment d'entrée w donne une réponse unique $f(w)$.

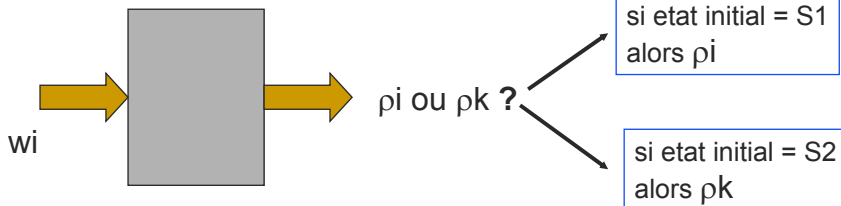
Chaque fonction f_i est associée à un état initial

Niveau 2 : Observation de la fonction d'entrée-sortie

Cette spécification est déterministe. On pourra l'opérationnaliser directement en construisant des abstractions des différents états initiaux du système réel.

(w_i, ρ_i) et (w_i, ρ_k) →

même trajectoire d'entrée
et
trajectoires de sortie différentes,
MAIS ETAT INITIAL DIFFERENT



Le comportement du système est décrit par des fonctions

Niveau 3 : Système dynamique

A ce niveau, on spécifie une abstraction du comportement interne du système.

Un Système d'E/S ou **Système Dynamique** est une structure S:

$$S = \langle T, X, Y, \Omega, Q, \Delta, \Lambda \rangle$$

T: base de temps,

X: ensemble des valeurs d'entrées,

Y: ensemble des valeurs de sortie,

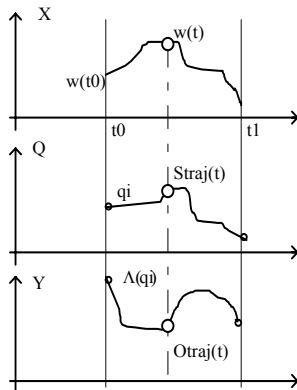
Ω : ensemble des segments d'entrée admissibles $w: \langle t1, t2 \rangle \rightarrow X$ sur T

Δ : $Q \times \Omega \rightarrow Q$ fonction de transition globale,

La fonction de transition globale du système définit l'état final obtenu après l'application d'un segment d'entrée et ne définit rien sur les états intermédiaires. Il s'agit de spécifier le comportement à reproduire et non **comment** le reproduire.

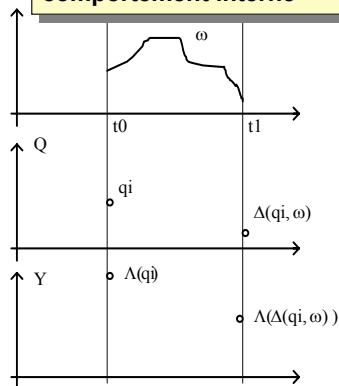
Spécification comportementale d'un système dynamique

Donne le Quoi et pas le Comment



observation

On ne veut pas modéliser le comportement interne

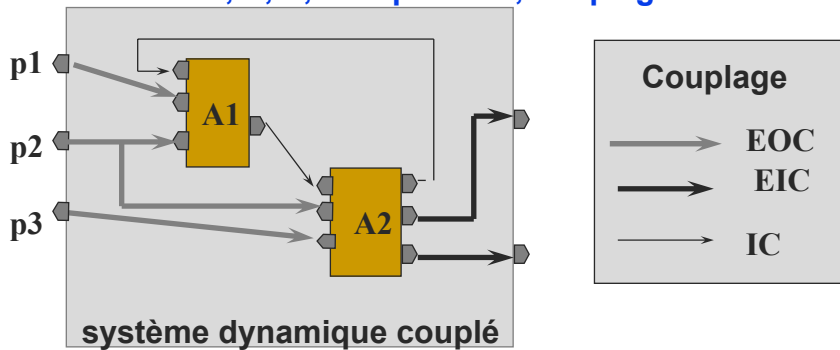


spécification

Niveau 4 : Système couplé

A ce niveau, on a connaissance de la structure interne du système, on spécifie les composants qui le constituent et leurs inter-relations:

CS = < T, X, Y, Composants, couplages >



$X_n = \{ p1, p2, p3 \}$

$Y_n = \{ p4, p5 \}$

$C = \{ A1, A2 \}$

Mercredi 21 septembre 2005

Introduction à la simulation : principaux concepts

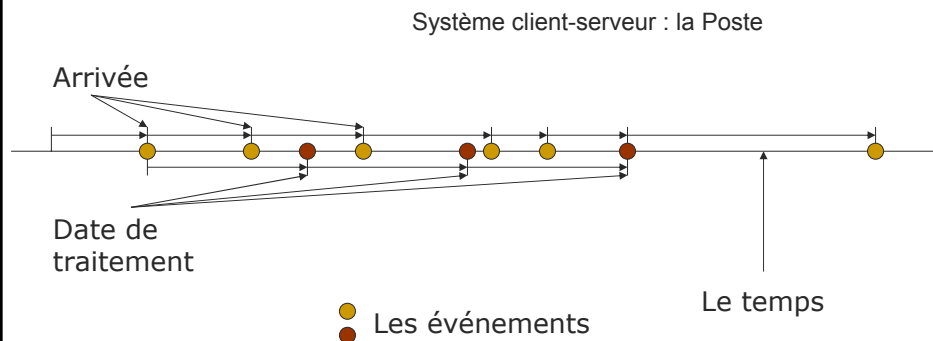
47

Simulation à événements discrets

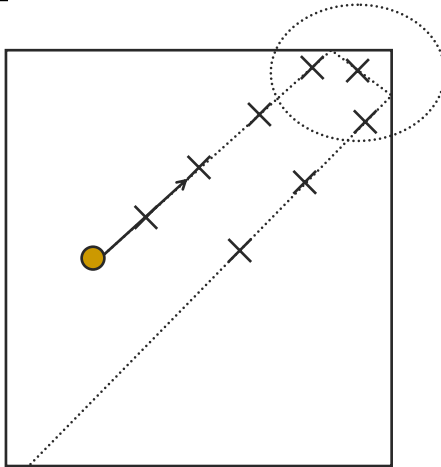
Événements discrets ou temps discret ?

- Le temps et les variables d'états changent lorsque des événements surviennent.
- L'état total du système est calculé à chaque pas de temps.
- Le temps discret est un cas particulier des événements discrets.

Exemple 1



[Exemple 2]



En temps discret, comment être conforme au modèle physique ?

[La multi-modélisation et le couplage de modèles]

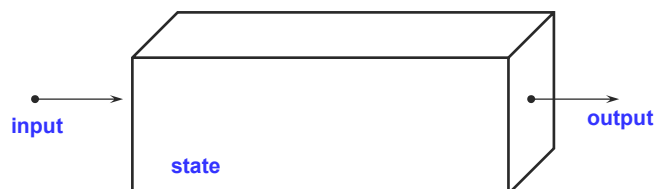
[DEVS]

- DEVS = Discrete Event Specification
- Formalisme à événements discrets : base de temps continu
- Couplage de modèles atomiques pour construire des modèles complexes
- Mécanisme de simulation indépendant de l'implémentation

[DEVS]

1965

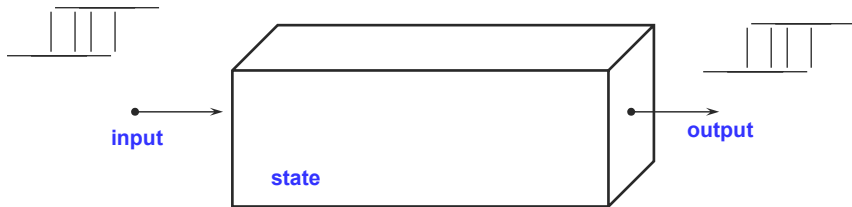
SYSTEM



[DEVS]

1965

SYSTEM



[DEVS]

1970

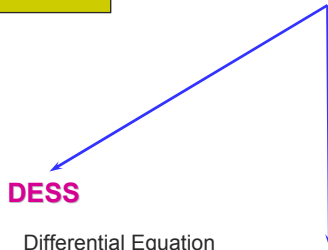
SYSTEM

DESS

Differential Equation
System Specification

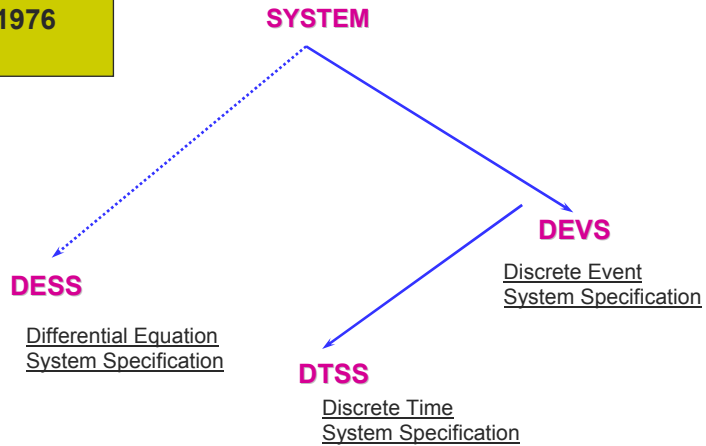
DTSS

Discrete Time
System Specification



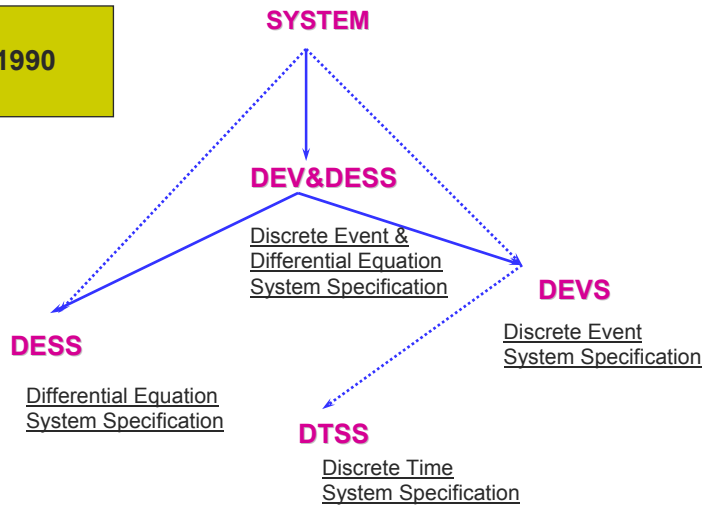
DEVS

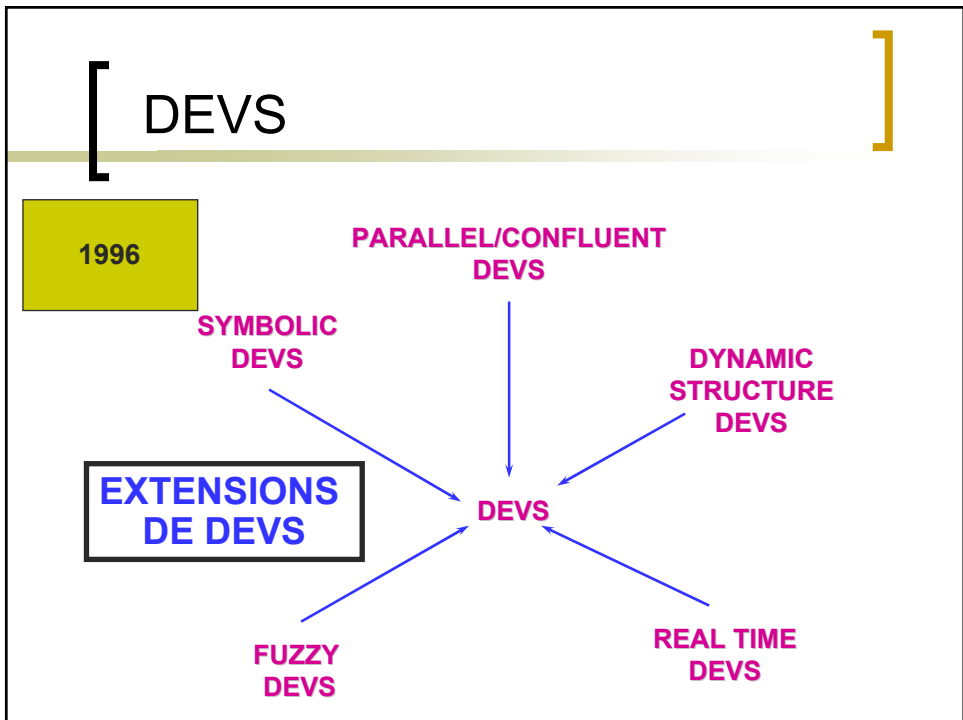
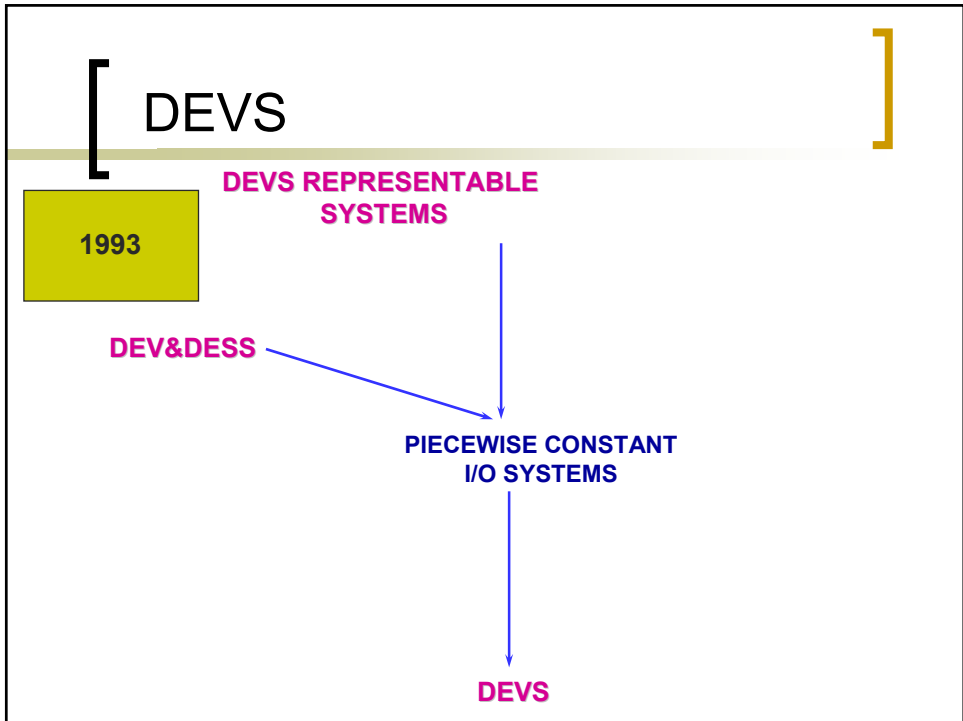
1976



DEVS

1990





DEVS - Définition

■ Modèle atomique :

$$DEVS = (\underline{X}_M, \underline{Y}_M, \underline{S}, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$$

- où \underline{X}_M est l'ensemble des ports et des valeurs d'entrée
- \underline{Y}_M l'ensemble des ports et des valeurs de sortie
- \underline{S} l'ensemble des états du système
- δ_{ext} la fonction de transition externe, δ_{int} la fonction de transition interne
- δ_{con} la fonction de transition " conflit "
- λ la fonction de sortie
- ta la fonction d'avancement du temps

DEVS - Définition

Un modèle à événements discrets est une structure:

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle.$$

δ_{int} : fonction de transition interne

$$\delta_{int} : S \rightarrow S$$

δ_{ext} : fonction de transition externe

$$\delta_{ext} : S \times X \rightarrow S$$

λ : fonction de sortie,

$$\lambda : S \rightarrow Y$$

ta : fonction durée de vie d'un état.

$$ta : S \rightarrow R^+$$

DEVS - Définition

● Interprétation

● Durée de vie d'un état

$t_a(s_i)$: est le temps durant lequel le modèle demeure dans l'état s_i , si aucun événement externe ne survient.

● Etat total

L'ensemble Q des états totaux du système est :

$$Q = \{(s_i, e) \mid s_i \in S, 0 < e < t_a(s_i)\}.$$

e représente le temps écoulé dans l'état s_i

DEVS - Définition

● Interprétation

● Fonction de Transition

● δ_{int} :

Le modèle étant rentrée dans l'état s à t_i , il ira dans s' ,

$$s' = \delta_{int}(s),$$

si aucun événement externe ne survient avant $t_i + t_a(s)$.

● δ_{ext} :

Si un événement externe survient, le système étant dans l'état s depuis le "temps écoulé" e , il transitera vers l'état s' :

$$s' = \delta_{ext}(s, e, x)$$

1- L'état futur est fonction du temps écoulé dans l'état

2- A chaque changement d'état, e est remis à 0

DEVS - Définition

● Interprétation

● Fonction de Sortie

La fonction de sortie est exécutée avant chaque activation de la fonction de transition interne.

En respect de la non-instantanéité de réponse d'un système physique, les sorties sont retardées par rapport aux entrées.

Remarque :

Si l'on désire activer la fonction de sortie sur occurrence d'un événement externe, il faut affecter la durée de vie de l'état actuel à 0 ($ta(s) := 0$).

DEVS - Définition

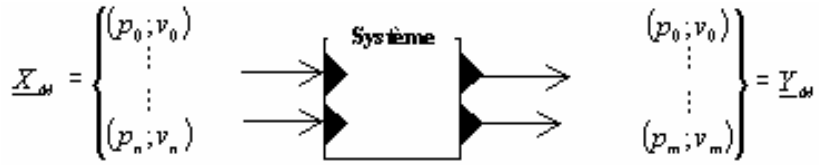
Déf. : Etat actif - Etat passif

Un état dont la durée de vie est infinie est dit passif, sinon il est dit actif.

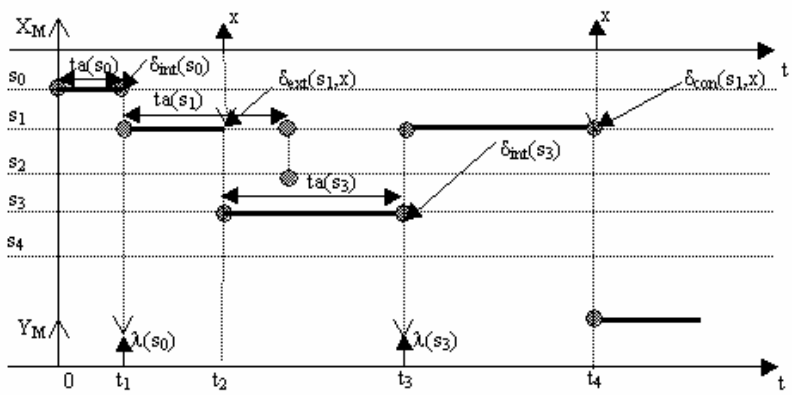
Si s est passif le modèle ne peut évoluer que sur occurrence d'un événement externe,

Remarque : $\delta \text{int}(s)$ n'est pas définie pour un état passif

[DEVS]

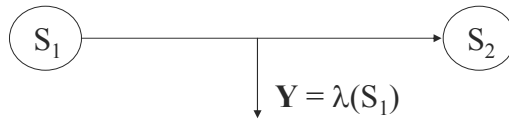


[DEVS]



[DEVS - Graphe d'états]

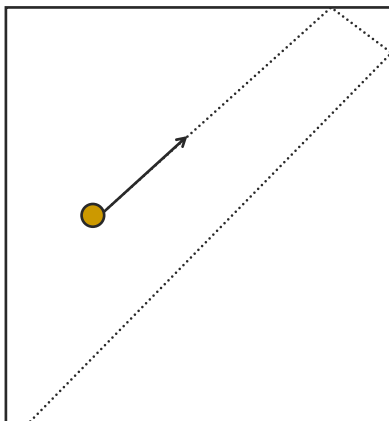
■ Transition interne $\delta_{\text{int}}(S_1)$



■ Transition externe $\delta_{\text{ext}}(S_1)$



[Exemple]



$$S = ((x, y), (dx, dy))$$

$$ta(S) = \frac{1}{v} \min \begin{cases} x_{\max} - x/dx & \text{si } dx > 0 \\ x - x_{\min}/dx & \text{si } dx < 0 \\ y_{\max} - y/dy & \text{si } dy > 0 \\ y - y_{\min}/dy & \text{si } dy < 0 \end{cases}$$

$$\delta_{\text{int}}(S) = \dots$$

$$\delta_{\text{ext}}(S) = \emptyset$$

[DEVS couplé]

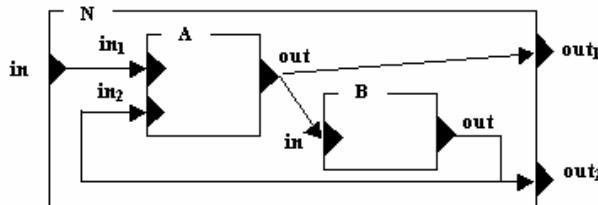
$$N = (X, Y, D, \{M_d/d \in D\}, EIC, EOC, IC)$$

$$EIC = \{((N, a), (d, b)) / a \in IPorts, b \in IPorts_d\}$$

$$EOC = \{((N, a), (d, b)) / a \in OPorts, b \in OPorts_d\}$$

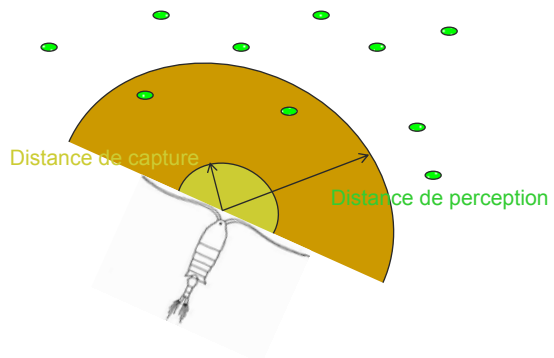
$$IC = \{((i, a), (j, b)) / i, j \in D, i \neq j, a \in OPorts_i, b \in IPorts_j\}$$

[DEVS couplé]

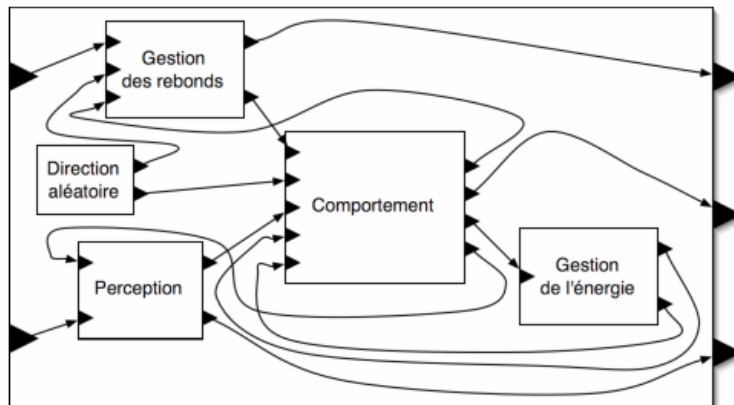


- $IPorts_N = \{in\}$ et $OPorts_N = \{out_1, out_2\}$
- $D = \{A, B\}$
- $EIC = \{((N, in), (A, in_1))\}$
- $EOC = \{((A, out), (N, out_1)), ((B, out), (N, out_2))\}$
- $IC = \{((A, out), (B, in)), ((B, out), (A, in_2))\}$

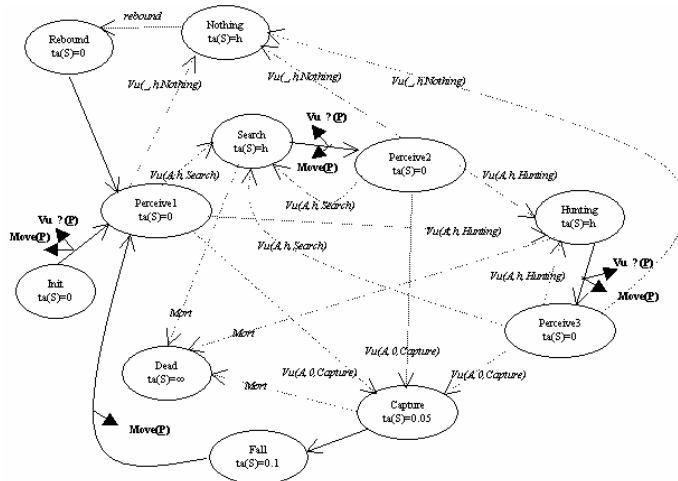
Exemple



Exemple



Modèle comportemental



Mercredi 21 septembre 2005

Introduction à la simulation : principaux concepts

75

Première forme de couplage



Perturbation

$$\begin{aligned} \frac{dX_1}{dt} &= A - F \\ \frac{dX_2}{dt} &= A - \frac{C}{M_N} \\ \frac{dX_3}{dt} &= F - G \\ \frac{dX_4}{dt} &= G \\ \frac{dX_5}{dt} &= C_{st} + C_{sda} + C_{sw} \end{aligned}$$

Mercredi 21 septembre 2005

Introduction à la simulation : principaux concepts

76

DEVS

La spécification permet de construire un modèle comportemental **atomique**

Ce modèle devra être **modulaire**



posséder des ports d'entrée et de sortie par lesquels transite toute interaction avec son environnement

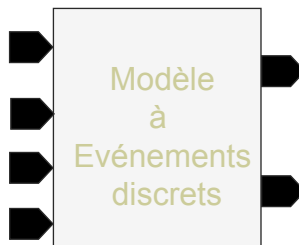
Concept d'Encapsulation

Le modèle sera spécifié selon deux vues :

- une vue **externe** définissant les **ports d'entrée-sortie**
- une vue **interne** définissant un **comportement**.

DEVS

Ports
d'entrée



Ports
de sortie

atomique



comportemental
non-décomposable

modulaire



interface d'E/S

événements discrets



base de temps : réels

Simulation d'un modèle atomique à événements discrets

l'état du modèle: s ,

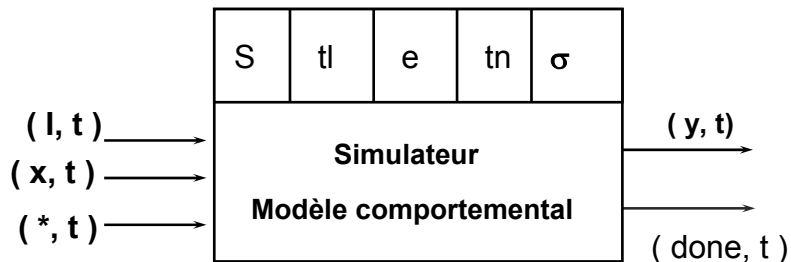
la date du dernier événement: t_l ,

la date du prochain événement interne: $t_n = t_l + ta(s)$,

le temps écoulé depuis le dernier événement: $e = t - t_l$

le temps restant avant le prochain événement interne σ :

$$\sigma = t_n - t = ta(s) - e$$



Mercredi 21 septembre 2005

Introduction à la simulation : principaux concepts

79

Simulation d'un modèle atomique à événements discrets

Ce simulateur reçoit trois types de messages:

(x, t) : événement externe,

$(* , t)$: synchronisation, événement interne,

(l, t) : initialisation.

Il émet deux types de message:

(y, t) , événement de sortie de date t ,

$(done, t_n)$ message prochaine transition.

Mercredi 21 septembre 2005

Introduction à la simulation : principaux concepts

80

Simulation d'un modèle atomique à événements discrets

```
When receive : (x, t)
  IF t1 <= t <= tn THEN
    e = t - t1
    s =  $\delta$  ext ( s, e, x)
    t1 = t
    tn = t1 + ta (s)
    send (done, tn) au père
  ELSE "erreur"
  end IF
end when
```

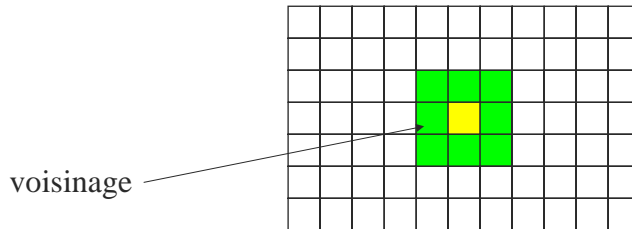
```
When receive i-message (i, t)
  t1 = t-e
  tn = t1 + ta (s)
```

```
When receive (*,t)
  IF t = tn THEN
    y =  $\lambda$  (s)
    send y-message (y,t) au père
    s =  $\delta$  int ( s )
    t1 = t
    tn = t1 + ta (s)
    send (done, tn) au père
  ELSE " erreur "
  end IF
end when
```

Cell-DEVS : DEVS pour les automates cellulaires

[Cell-DEVS]

- Modélisation d'un espace multi-dimensionnel discret
→ Automate cellulaire



[Cell-DEVS]

- *Timed cell-devs* [Wainer & Giambiasi]
 - Proposition de modélisation DEVS et de simulateur
 - Automate cellulaire temporisé
- Deux modèles :
 - Un modèle atomique pour les cellules
 - Un modèle couplé pour l'automate

[Cell-DEVS]

- L'état d'une cellule est calculée en fonction de l'état des voisins
- L'état d'une cellule est « transmis » aux cellules voisines après un délai de « transport »
- Le modèle DEVS d'une cellule est définie par une structure :

$$TDC = \langle I, X, Y, S, N, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D \rangle$$

[Cell-DEVS]

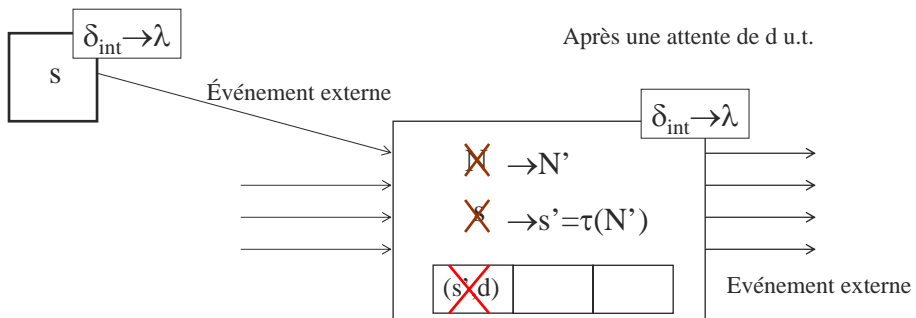
- I représente l'interface de la cellule :
 - η : taille du voisinage
 - P^X et P^Y : les ports d'entrée et de sortie*Il y a autant de ports que de voisins*
- $S = \{(s, \text{phase}, \sigma_{\text{queue}}, \sigma)\}$
où s est l'état de la cellule
- phase prend les valeurs : active ou passive
- σ_{queue} contient les états passés de la cellule qui sont à transmettre aux voisins
- σ est la durée avant transmission de l'état de la cellule aux voisins

[Cell-DEVS]

- N : ensemble des états des cellules voisines
- d : temps de « transport »
- τ : fonction de calcul de l'état de la cellule en fonction de l'état du voisinage (N)

[Cell-DEVS]

Fonction de transition externe



[Le simulateur (1)]

When receive i-message (i, t)

$tl = t - e$

$tn = tl + ta(s)$

phase=passive

σ_{queue} is empty

$\sigma = \infty$

END WHEN

$ta(s) = \sigma$

[Le simulateur (2)]

When receive : (x, t)

IF $tl \leq t \leq tn$ THEN

$e = t - tl$

update N

$s' = \tau(N)$

IF ($s' \neq s$) THEN

$s = s'$

$tl = t$

IF (phase=passive) THEN

phase=active

$\sigma = d$

ELSE

for ($a_i \in \sigma_{queue}$) $a_i.\sigma = a_i.\sigma - e$

$\sigma = \sigma - e$

ENDIF

insert (s,d) into σ_{queue}

$tn = tl + ta(s)$

send (done, tn) to parent

ENDIF

ELSE "erreur"

END IF

END WHEN

[Le simulateur (3)]

```
When receive (*,t)
  IF t = tn THEN
    for (ai ∈ σqueue) ai.σ = ai.σ - first(σqueue).σ
    s' = first(σqueue).s
    y = s'
    delete the first element of σqueue
    send y-message (y,t) to parent
    IF (σqueue is empty)
      phase = passive
      σ = ∞
    ELSE
      phase = active
      σ = first(σqueue).σ
    ENDIF
    tl = t
    tn = tl + ta (S)
    send (done, tn) to parent
  ELSE " error "
  END IF
END WHEN
```

[Cell-DEVS]

- La durée d'attente de transmission peut être une constante et peut être identique dans toutes les cellules
 - automate cellulaire synchrone
- Cette durée peut être différente selon les cellules et varier dans le temps.

[Cell-DEVS]

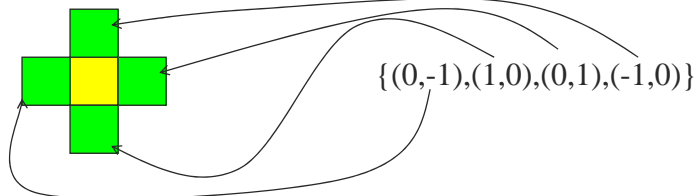
- L'automate est défini par une structure de modèles couplés :

CTD = $\langle \eta, N, \{m,n\}, C, B, Z, \text{select} \rangle$

- η : taille du voisinage
- N : type de voisinage
- $\{m,n\}$: taille de l'espace (nb de cellules)
- C : ensemble des modèles des cellules
- B : définition de la frontière si nécessaire
- Z : définition des couplages externes et internes
- select : fonction de sélection

[Cell-DEVS]

- Le type de voisinage peut être défini génériquement au niveau de l'automate



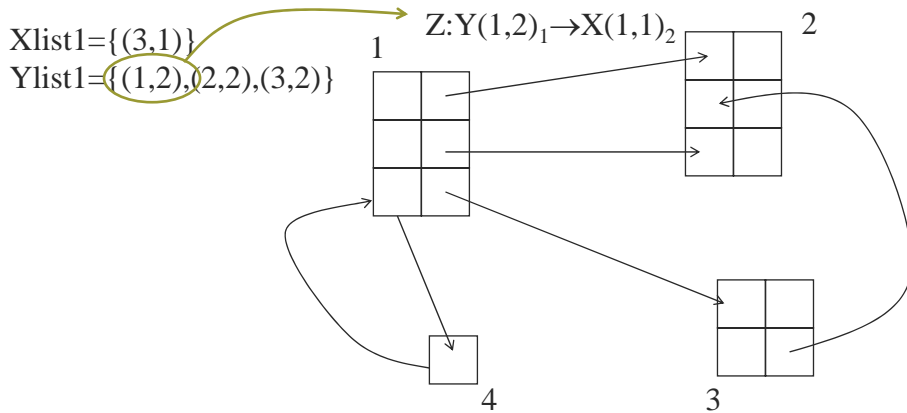
[Cell-DEVS]

- Z définit les couplages entre les cellules de l'automate
- La fonction de sélection permet de définir la politique d'évaluation des états des cellules

[Cell-DEVS]

- Si le modèle est un modèle couplé d'automates cellulaires alors la structure admet deux attributs supplémentaires Xlist et Ylist
- Xlist et Ylist définissent respectivement la liste des cellules d'un automate possédant des entrées et/ou des sorties avec les autres automates.

[Cell-DEVS]

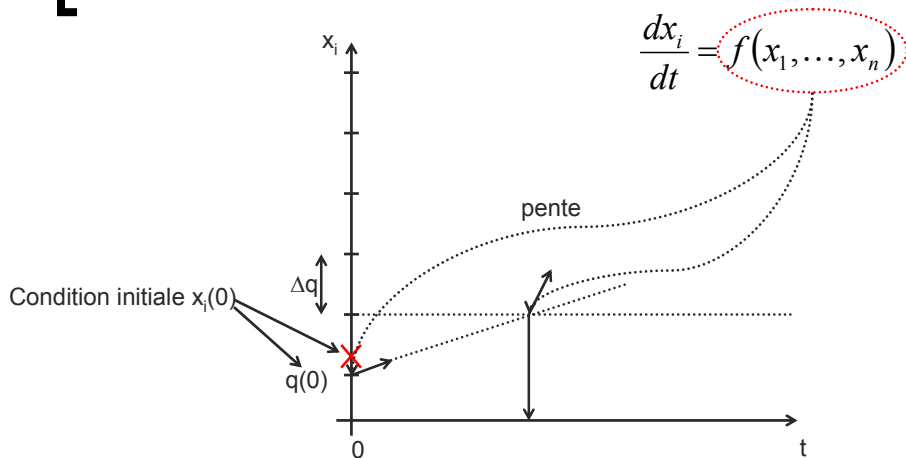


[Équations différentielles ordinaires et DEVS]

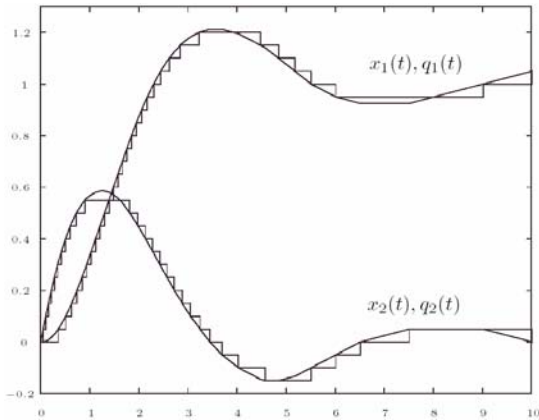
[QSS : Quantized State System]

- Méthode proposée par E. Kofman en 2001 pour résoudre des équations différentielles du 1er ordre basée sur :
 - Discrétisation des valeurs au lieu du temps : la quantification des fonctions
 - Le calcul du pas de temps en cohérence avec la pente calculée grâce au système.

[QSS : Quantized State System]



[QSS : Quantized State System]



[QSS : Quantized State System]

- Propriétés :
 - Erreur $\leq \Delta q$
 - Grande stabilité même pour des Δq grand !

[Dynamic structure DEVS]

[Dynamic structure DEVS]

- Offrir une structure de connexions dynamiques
- Au cours du temps, le couplage entre les modèles composants le modèle global évolue
- Les modèles peuvent apparaître et disparaître

Dynamic structure DEVS

- Une structure dynamique est définie par la structure suivante :

$$DSDEVN_{\Delta} = \langle X_{\Delta}, Y_{\Delta}, \chi, M_{\chi} \rangle$$

- où Δ est le nom du réseau
- X_{Δ} et Y_{Δ} sont respectivement les entrées et les sorties du modèle global
- M_{χ} est le modèle de la forme active notée χ .

Dynamic structure DEVS

- Le modèle M_{χ} est défini par une structure :

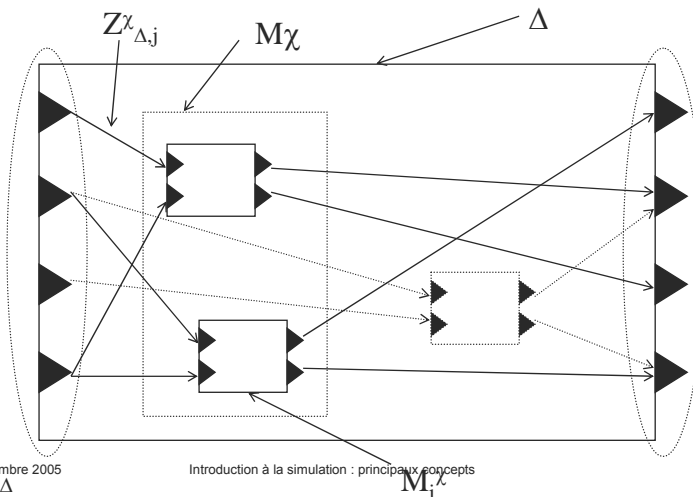
$$M_{\chi} = \langle X_{\chi}, S_{\chi}, Y_{\chi}, \delta_{int\chi}, \delta_{ext\chi}, \lambda_{\chi}, D_{\chi} \rangle$$

- Cette structure est équivalente à un modèle DEVS atomique
- Seul S_{χ} est modifié.
- Naturellement, la dynamique des fonctions de transition sont spécifiques.

Dynamic structure DEVS

- L'état du modèle dynamique se compose de :
 - D_χ : l'ensemble des composants (nom des modèles composants le modèle dynamique) actifs
 - M_{χ_i} : modèle du $i^{\text{ème}}$ composant actif
 - I_{χ_i} : l'ensemble des composants sous l'influence du $i^{\text{ème}}$ composant actif
 - $Z_{i,j}^\chi$: fonction de traduction entre les ports du modèle du composant i et ceux du modèle du composant j
 - Select : fonction de gestion des conflits entre composants
 - θ^χ : ensemble de variables d'état du modèle global

Dynamic structure DEVS



[Le simulateur (1)]

```
When receive i-message (i, t)
  send i-message(i,t) to {i/i ∈ Dx}
  waitUntil (done) from {i/i ∈ Dx}
  tl = t
  tn = min{tn,i/i ∈ Dx}
  send (done,tn) to parent
END WHEN
```

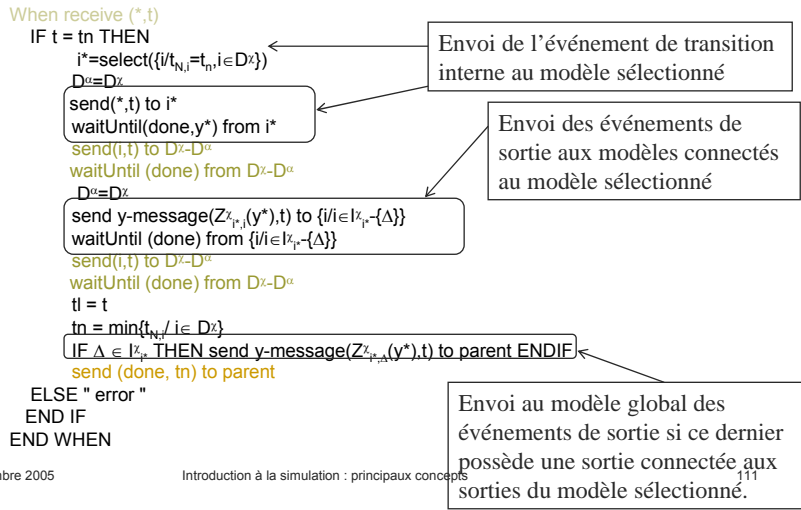
[Le simulateur (2)]

```
When receive : (x, t)
  IF tl <= t <= tn THEN
    Dα = Dx
    send(ZxΔ,i(x),t) to {i/i ∈ Ix}
    waitUntil (done) from {i/i ∈ Ix}
    send(i,t) to Dx-Dα
    waitUntil (done) from Dx-Dα
    tl = t
    tn = min{tn,i/i ∈ Dx}
    send (done,tn) to parent
  ELSE "error"
  END IF
END WHEN
```

Envoi de l'événement x à tous les modèles dont une entrée est connectée à une entrée du modèle global.

Envoi de l'événement d'initialisation à tous les modèles ajoutés au modèle global.

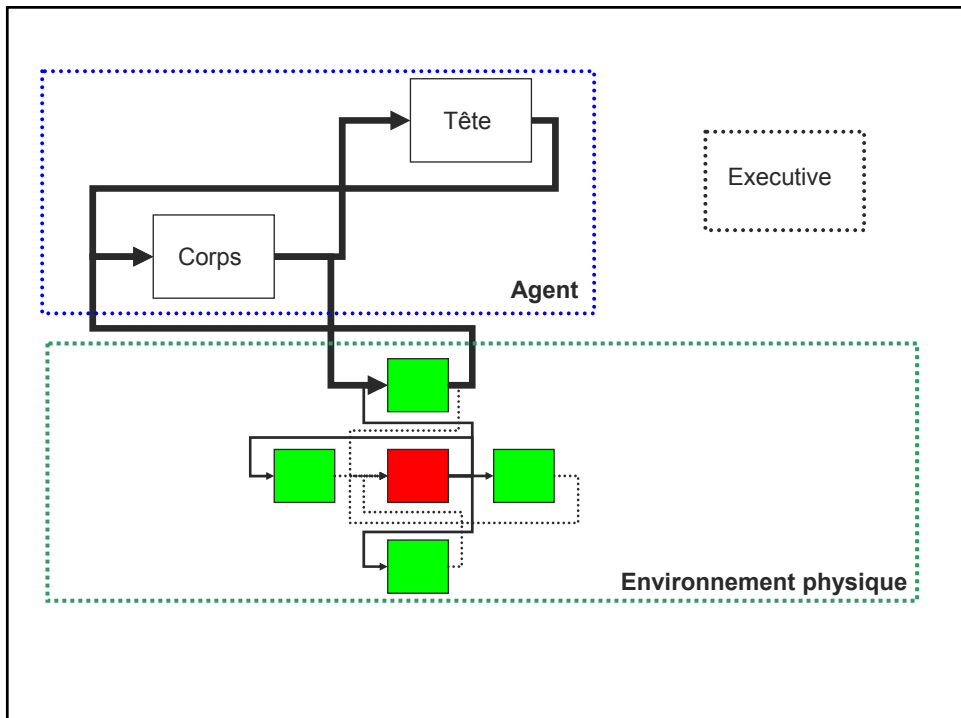
Le simulateur (3)



Exemples

Exemple 1

- L'exemple « Pompier »
 - Un environnement physique :
 - 2D discret → Automate cellulaire → Cell-DEVS
 - Dynamique de propagation du feu : **discrète ou continue**
 - Deux types d'agent : « Pompier » et « Incendiaire »
 - Des entités discrètes localisées sur une case = DS-DEVS + un modèle atomique pour les règles comportementales + un modèle atomique couplé à la cellule.
 - Perturbation de la température du milieu.
 - Augmentation de la température par les pyromanes.
 - Diminution de la température par les pompiers.



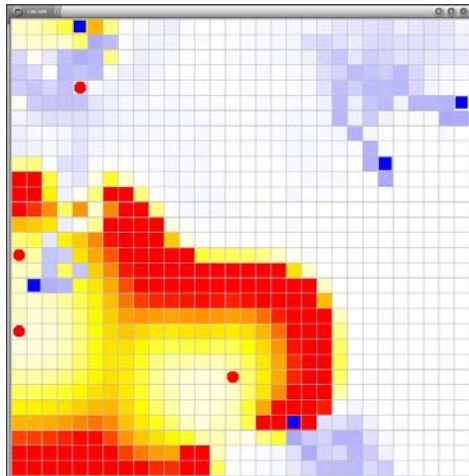
[Exemple 1]

Le cas à une dimension

$$\frac{\partial T(t, x)}{\partial t} = K \frac{\partial^2 T(t, x)}{\partial x^2} \quad \text{où K est le coefficient de diffusion}$$

$$\frac{\partial T_i(t)}{\partial t} \approx K \frac{T_{i-1}(t) - 2T_i(t) + T_{i+1}(t)}{(\Delta x)^2}$$

[Exemple 1]



[Exemple 2 : une pêcherie]

- Un environnement physique, un espace de pêche :
 - un espace découpé en un certain nombre de zones.
 - des zones dans lesquelles vive une seule espèce de poisson.
 - une zone = une équation différentielle de dynamique de populations (croissance, migration, ...)
 - mise en œuvre = Cell-DEVS + QSS

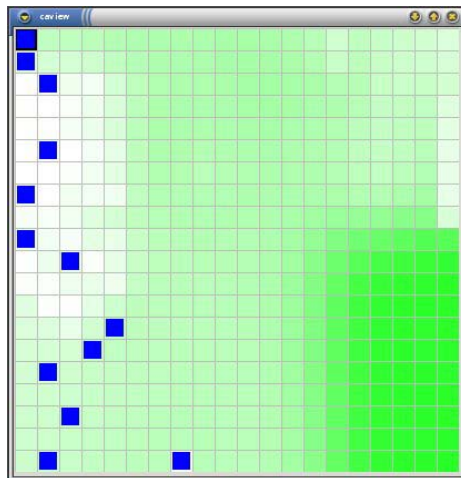
[Exemple 2]

- Des entités, des bateaux et un port : un modèle économique
- Les pêcheurs :
 - évoluent dans l'espace de pêche.
 - consomment du carburant.
 - prélèvent une certaine biomasse de poisson sur leurs zones cibles.
 - La biomasse prélevée = fonction de :
 - la capacité d'effort de pêche du pêcheur,
 - la capturabilité des poissons,
 - la durée de la pêche,
 - la densité présente dans la zone.
 - rentrent au port après une campagne

[Exemple 2]

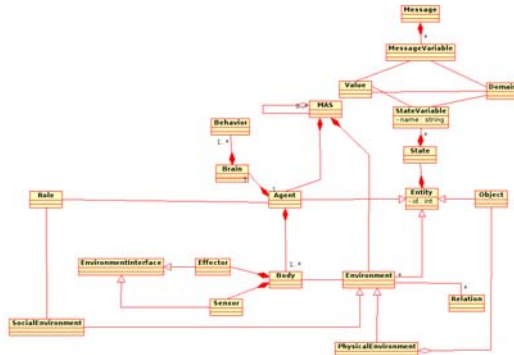
- Le port :
 - calcule le revenu des pêches.
 - affecte un nouveau scénario de pêche en fonction de la politique choisie par le pêcheur.
 - met à jour l'historique des pêches.
- Des fermetures de zones.
- Des prix de poissons variables en fonction de la zone.

[Exemple 2]



Conclusion

- Les SMA comme super-couche de DEVS et ses variantes



Les générateurs aléatoires

Les générateurs aléatoires

Les caractéristiques d'un bon générateur aléatoire

- 1) Les nombres générés doivent apparaître uniformément distribués sur $[0, 1]$ sans corrélation.
- 2) Le générateur doit être rapide et ne pas nécessiter beaucoup de mémoire – la première condition peut être satisfaite avec des tables pré-calculées mais dans ce cas pas la deuxième condition.
- 3) La série de nombres aléatoires doit être reproductible, afin de pouvoir mettre au point le modèle de simulation avec la même série ou de pouvoir utiliser la même série en phase de mise au point du modèle ou pour comparer des simulations avec des paramètres différents.
- 4) Il faut pouvoir générer plusieurs séries indépendantes.

Les générateurs aléatoires

- “Linear Congruential Generators”
- une séquence de nombre Z_1, Z_2, \dots , est définie récursivement :
$$Z_i = (aZ_{i-1} + c) \pmod{m},$$
- où m (le *modulo*), a (le multiplicateur), c (l'incrément) et Z_0 (la graine ou le germe) sont des entiers positifs.
- Afin d'obtenir une séquence de nombre aléatoires dans $[0, 1]$, il faut diviser par m : $U_i = Z_i/m$.
- De plus : $0 < m$, $a < m$, $c < m$ et $Z_0 < m$.
- Périodicité $< m$
- Utilisé par Java 1.4 avec :
 - $a = 25.214.903.917$
 - $c = 11$
 - $m = 2^{48}$ soit $281.474.976.710.656 !$
 - Mais Integer sur 2^{32} max soit $4.294.967.296$
 - Float sur 2^{24} max soit $16.777.216$

[Les générateurs aléatoires]

- C++/gcc : l'algorithme de Mersenne & Twister
- Période = $2^{19937}-1$
- Attention, float = 32 bits !
- Utiliser des Double = 64 bits.

[La Simulation distribuée et parallèle]

[Objectif]

- Passer du modèle à une implémentation
- Utiliser un maximum la puissance des ordinateurs
- Pourquoi utiliser qu'un ordinateur pour faire une simulation ?
 - Comment faire travailler plusieurs ordinateurs entre-eux ?
 - Quelles sont les possibilités ?
 - Quels sont les problèmes ?

[Solutions architecturales possibles]

- Un processeur
 - Limité par la mémoire et la puissance de calcul
- Multiprocesseurs :
 - Utilisation de mémoires partagées
 - Mais ordinateurs coûteux
- Interconnexion d'ordinateurs, clusters
 - Possibilité de recourir à une mémoire partagée
 - Solution peu onéreuse

Quelles méthodes de distribution adopter ?

1. Parallélisation automatique
 - Analyse automatique du code
 - Accélération obtenue reste faible
2. Distribution des expériences
 - Code séquentiel identique sur tous les processeurs
 - Paramètres d'entrées uniquement qui changent
3. Distribution des fonctions du simulateur

Quelles méthodes de distribution adopter ?

3. Distribution des événements
 - Utilisation d'un échéancier global
 - En général, utilisation d'une mémoire partagée
 - Intéressant quand le volume d'informations échangées est important
4. Distribution des éléments du modèle
 - Utilisation du parallélisme du modèle
 - Échange de messages datés entre les processus
 - Obligation d'utiliser une synchronisation des processus

Exécution distribuée → la causalité

- Toute conséquence possède une cause
- Simulation séquentielle : aucun problème
- Simulation distribuée ?
 - 2 actions A et B
 - A sur processeur P et B sur le processeur Q
 - Si B calculé avant A ????
- Causalité impose *ordre partiel* entre les transitions
 - Lamport [Lam78] a défini une méthode pour respecter cette causalité

Le problème de la causalité ?

- Généralement, problème rencontré lors de la distribution des événements du modèle
- Avec les autres types de distributions
 - Utilisation d'un échéancier global
- Pourquoi distribuer les éléments du modèle ?
 - Pas de mémoire partagée
 - Optimisation du parallélisme du modèle

[Le temps ?]

- Temps physique :
 - du 23 Janvier 2003 9h au 23 Janvier 2003 12h
- Temps simulé
 - Représentation du temps physique : [9.0-12.0]
- Temps Wallclock
 - Temps d'exécution de la simulation : 9:00 à 9:15

[Évolution synchrone]

- Temps simulé = horloge globale
- Toutes les horloges logiques (horloge de chaque processeur) avancent en même temps
- Existence d'un décalage dû à la propagation
- Assurance que lors du passage au temps t , tous les processeurs ont fini $t-1$
- Problème : inactivité de certains processeurs

[Évolution asynchrone]

- Chaque processeur avance à sa propre vitesse
- Problème ?

[Exemple de simulation]

Le trafic aérien et les aéroports

[Exemple d'un aéroport]

- 3 événements par avion :
 - Approche (avion sous contrôle de la tour)
 - Atterrissage
 - Décollage
- 3 états :
 - En_l'air : nombre d'avion en l'air
 - Au_sol : nombre d'avion au sol
 - Piste_libre : booléen
- 2 constantes temporelles
 - A : temps nécessaire pour atterrir
 - D : temps nécessaire pour pouvoir partir

- **Now** : date locale de simulation
- **InTheAir** : nombre d'avions en l'air (en approche ou en train d'atterrir)
- **OnTheGround** : nombre d'avions au sol
- **RunwayFree**: Booléen, vrai si la piste est libre

Arrival Event:

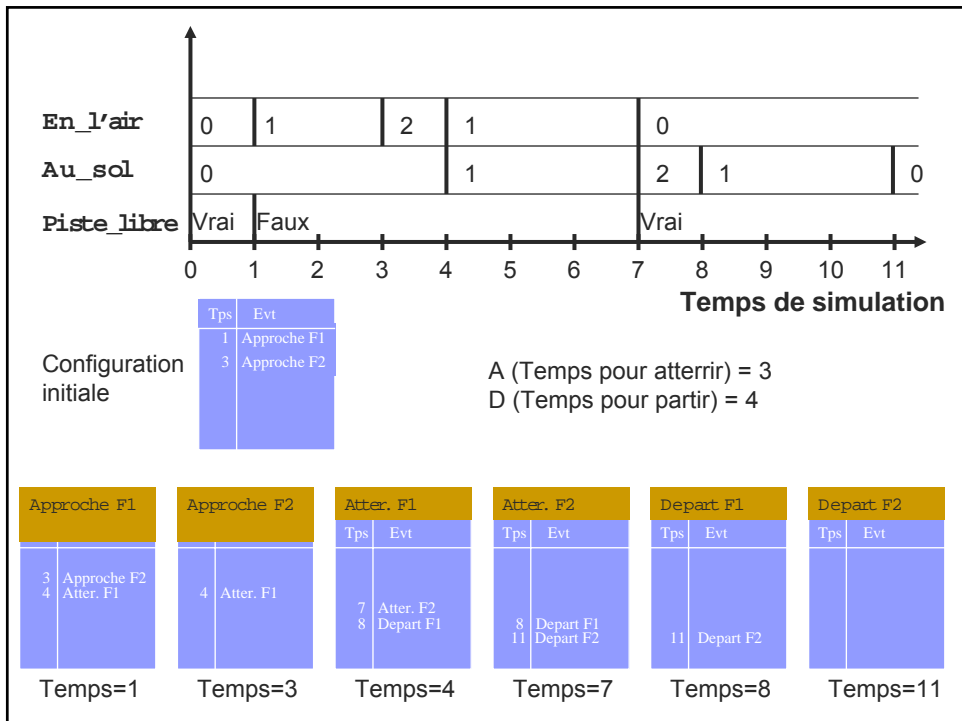
```
InTheAir := InTheAir+1;  
If (RunwayFree)  
  RunwayFree:=FALSE;  
  Schedule Landed event @ Now + R;
```

Landed Event:

```
InTheAir := InTheAir-1;   OnTheGround := OnTheGround+1;  
Schedule Departure event @ Now + G;  
If (InTheAir>0) Schedule Landed event @ Now + R;  
Else RunwayFree := TRUE;
```

Departure Event:

```
OnTheGround := OnTheGround - 1;
```

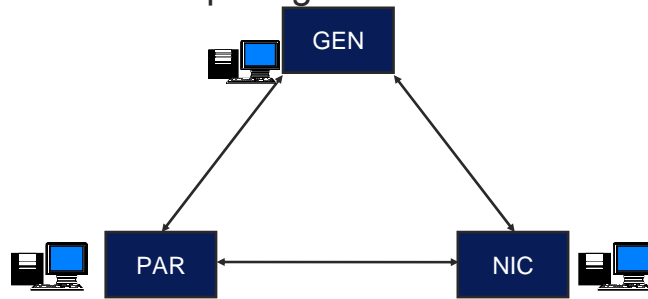


Le trafic de plusieurs aéroports ?

- Interaction entre eux
 - Obligation de s'échanger des messages
- *Exemple* : Avion décolle de Paris, il sera en approche à Genève 1h plus tard
 - Dans la simulation : programmer cette approche sur l'aéroport de Genève
- Simulation sur un processeur ou avec mémoire partagée : aucun problème mais limite des ressources
- Et sur plusieurs ordinateurs ?

Approche de l'exécution distribuée

- Simulation d'un modèle par ordinateur
- Communication par message
- Aucune mémoire partagée



- **Now** : date locale de simulation
- **InTheAir** : nombre d'avion en l'air (en approche ou en train d'atterrir)
- **OnTheGround** : nombre d'avions au sol
- **RunwayFree**: Booléen, vrai si la piste est libre

Arrival Event:

```
InTheAir := InTheAir+1;  
If (RunwayFree)  
  RunwayFree:=FALSE;  
  Schedule Landed event (local) @ Now + R;
```

Landed Event:

```
InTheAir := InTheAir-1;   OnTheGround := OnTheGround+1;  
Schedule Departure event (local) @ Now + G;  
If (InTheAir>0) Schedule Landed event (local) @ Now + R;  
Else RunwayFree := TRUE;
```

Departure Event : (D = delay to reach another airport):

```
OnTheGround := OnTheGround - 1;  
Schedule Arrival Event (remote) @ (Now+D) @ another airport
```

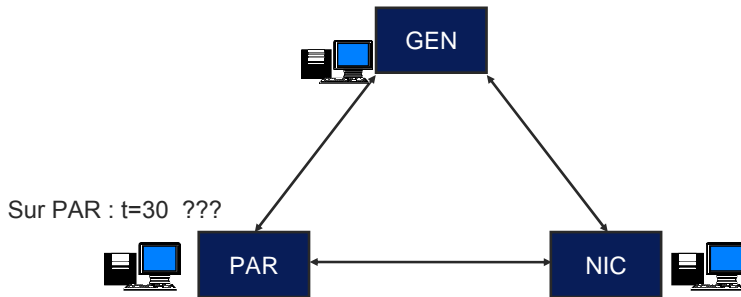
[La causalité ?]

Départ d'un avion à $t = 8$

GEN-PAR = 10

Calcul de l'arrivée à PAR : $t+10=18$

Envoi du message à PAR : une arrivée prévue à $t=18$



[Les règles d'or]

- Maintenir la causalité : temps réel (ou physique) est unique; sur un processeur, le temps simulé est unique
 - Mais sur plusieurs processeurs ???
- Maintenir la vivacité : le temps s'écoule

[2 types de synchronisation]

- Approche pessimiste (ou conservative) :
la simulation avance quand elle est sûre de pouvoir le faire
- Approche optimiste :
la simulation avance et détecte les violations de contrainte de causalité quand elles se produisent. Elle revient alors en arrière jusqu'à ce que la contrainte soit à nouveau satisfaite et la simulation repart.

[Hypothèses]

- Canaux FIFO
- Communications sûres
- Mémoire suffisante
- Tous les messages consécutifs provenant d'un même processeur sont estampillés avec des valeurs croissantes

[Approche pessimiste]

Synchronisation

[Approche pessimiste (ou conservative)]

- Approche synchrone, dirigée par le temps
- Approche asynchrone, dirigée par les événements

[Approche synchrone]

- Synchronisation forte : horloge globale
A chaque phase de calcul :
 - Chaque processeur exécute une action
 - Attente ensuite que tout le monde ait fini (barrière de synchronisation)
 - Quand détection de la terminaison, passage à la phase suivante

[Approche synchrone]

- Algorithme synchrone « faible » ou à fenêtre
 - Objectif : augmenter le nombre de processeurs actifs simultanément
 - Incrémente alors de $1 + \delta$
 - Calcul de δ ?
 - pré calculé par une analyse de la simulation
 - Calculé dynamiquement par un algorithme de calcul de minimum global

[Approche asynchrone (dirigée par les événements)]

- En distribué :
 - un processeur n'a pas de vision globale (pas de mémoire partagée)
 - Prise de décision en fonction des informations locales
 - Mais avant toute exécution d'un événement estampillé au temps T , le simulateur doit être sûr de ne pas recevoir dans le futur un message avec une estampille $T' < T$

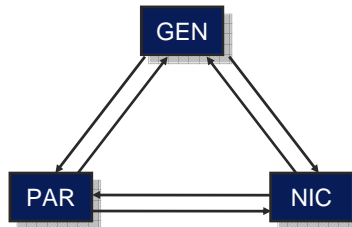
[Approche asynchrone (dirigée par les événements)]

- Solution ?
 - Un processeur doit attendre ?
 - Mais pendant combien de temps ?
 - Quelle décision prendre ?

Approche asynchrone : Chandy/Misra et Bryant "Null Message" Algorithm

Hypothèses

- Échange de messages avec estampille (daté) entre chaque ordinateur
- Topologie du réseau fixe
- Les messages sont expédiés dans l'ordre
- Le réseau achemine correctement les messages et les liens de communication conservent l'ordre d'expédition (canaux FIFO)



Chandy/Misra/Bryant "Null Message" Algorithm

WHILE (simulation n'est pas finie)

Wait Until chaque lien contient un message

Retirer le message avec la date minimale

Exécuter l'action correspondante

END-LOOP



Événement 2

Événement 4

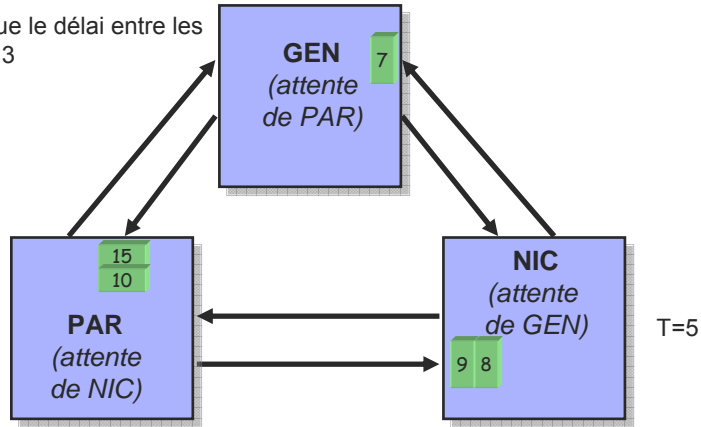
Événement 5

Attente d'un message

Problème : **DEADLOCK**

Une configuration de DEADLOCK

Supposons que le délai entre les aéroports est 3



Solution : NULL message

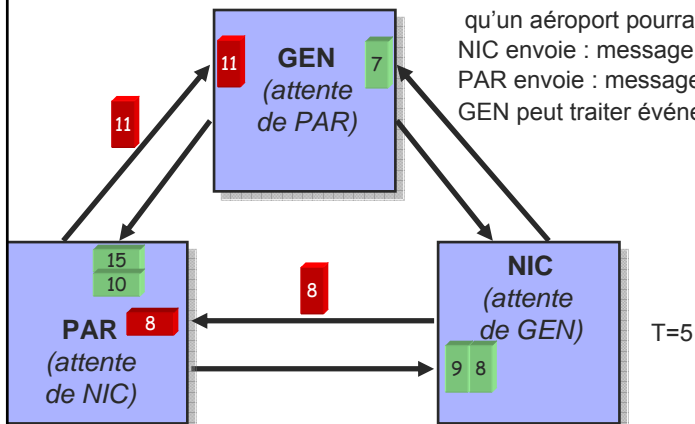
Supposons que le délai entre les aéroports est 3

Calcul de la date minimale de chaque message qu'un aéroport pourra recevoir

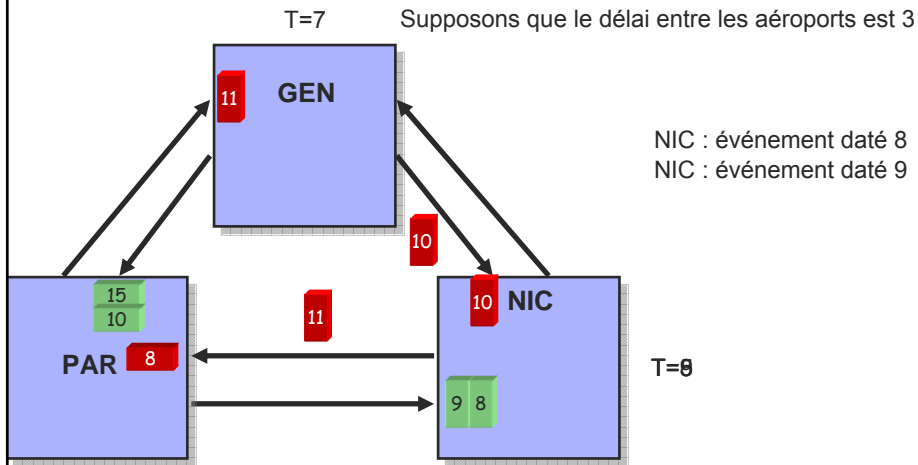
NIC envoie : message NULL estampillé $5+3=8$

PAR envoie : message NULL estampillé 11

GEN peut traiter événement daté à 7



Solution : NULL message



Mercredi 21 septembre 2005

Introduction à la simulation : principaux concepts

157

Algorithme du NULL message

Chaque processus i exécute :

WHILE (simulation n'est pas fini)

Wait Until chaque lien contient un message

Retirer le message avec la date minimale

Exécuter l'action correspondante

Envoyer à tous les voisins la date minimale qu'ils pourront recevoir de i .

END-LOOP

Mercredi 21 septembre 2005

Introduction à la simulation : principaux concepts

158

[Problème des approches conservatives]

- Non-exploitation du parallélisme totale de la simulation
- Configuration statique
 - Pas de création de nouveau processus au cours de la simulation
- Certaines simulations sont impossibles
 - Si l'incrément de l'estampille est nulle

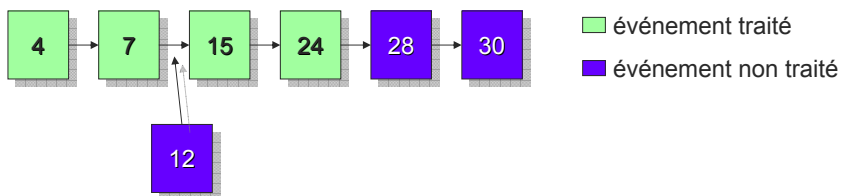
[Approche optimiste]

- Rappel : un simulateur par processeur
- Pas de vision globale
- Optimiste = « on fonce dans le brouillard »

[Time Warp (Jefferson)]

- Principe :
 - Chaque processus exécute les événements dans l'ordre
 - Si un message arrive avec une date antérieure : annulation des actions effectuées

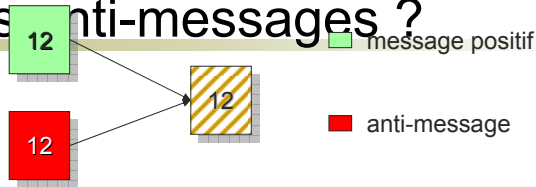
[Algorithme du Time Warp]



A la réception d'un message du « passé » :

- restaurer les états antérieurs
 - sauvegarder tous les états
- annuler les messages envoyés et à envoyer
 - utilisation d'anti-messages et de messages d'annulation

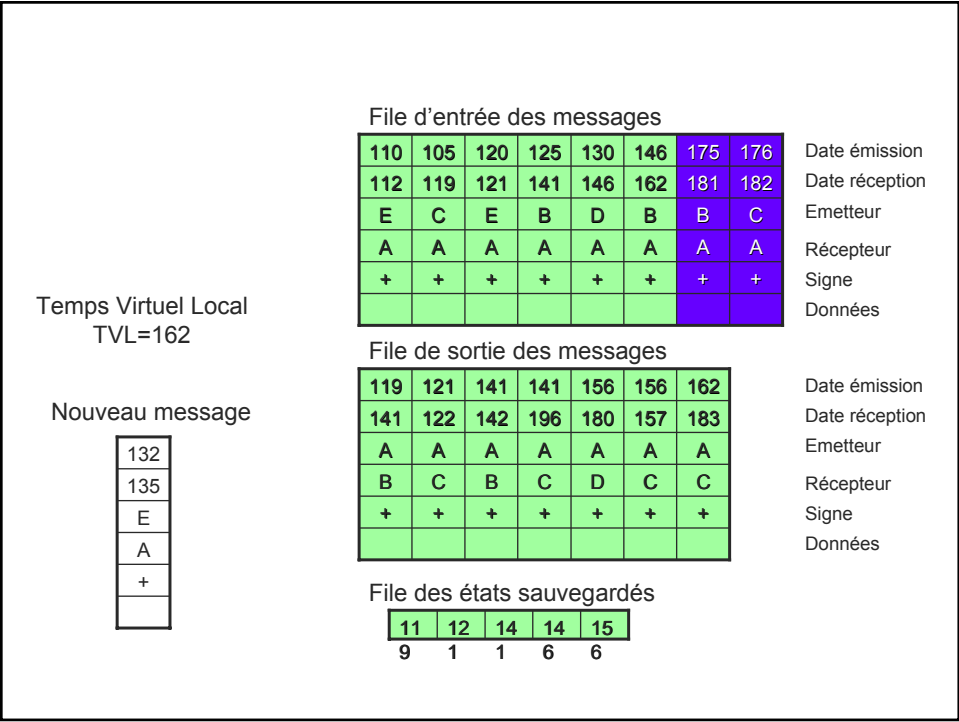
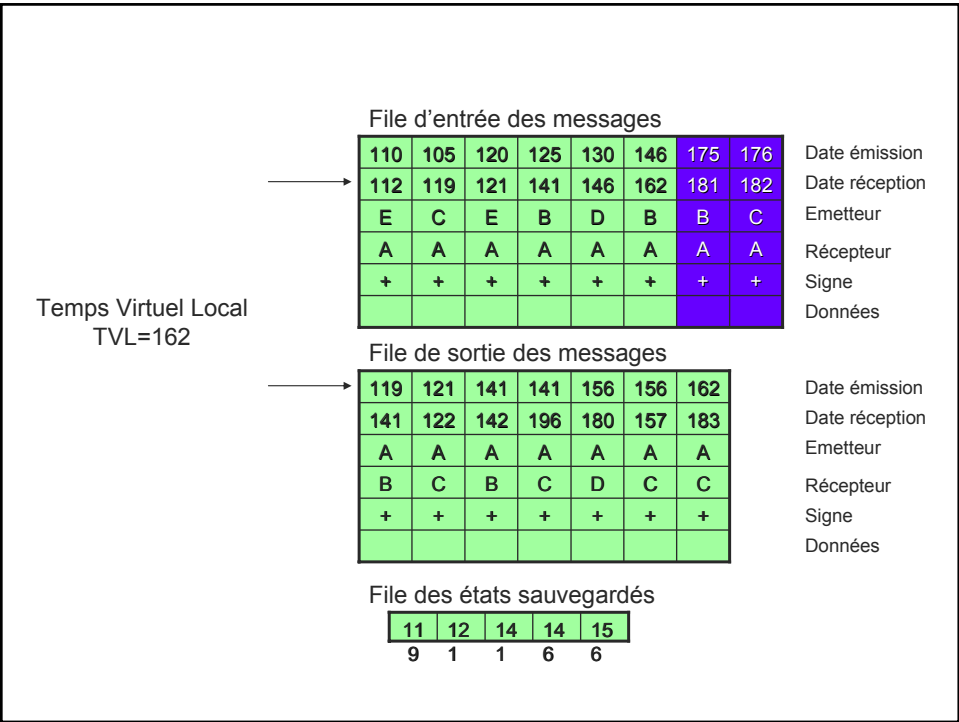
Les anti-messages ?

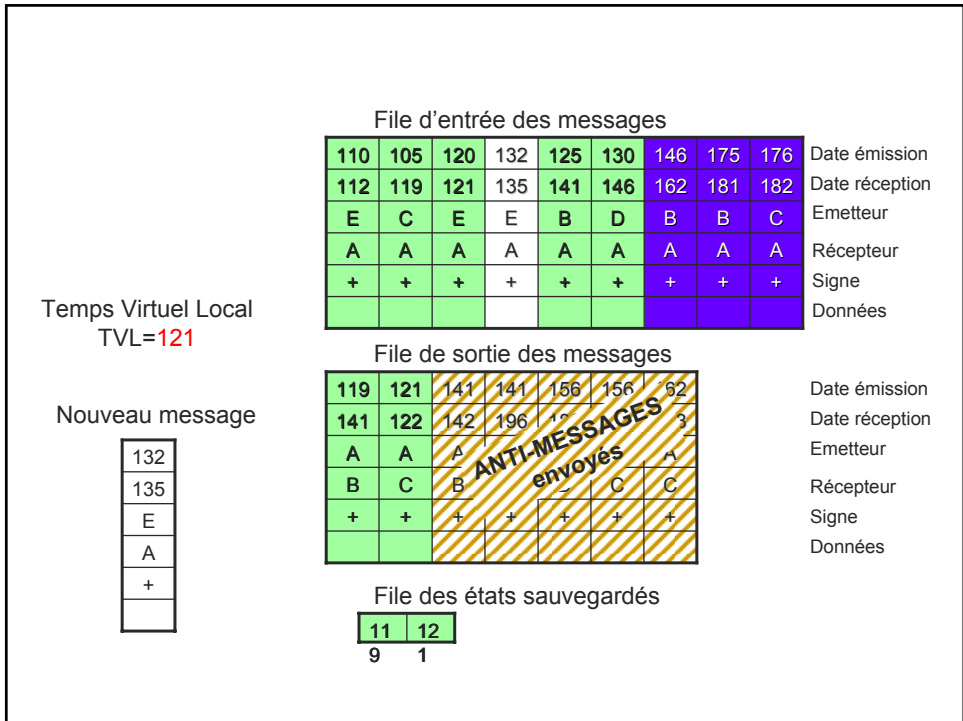


- Utilisée pour annuler un message
- A chaque message envoyé, il existe un anti-message correspondant
- Anti-message et message sont identiques, excepté un bit de signe
- Quand un message et son anti-message se rencontrent dans une file, ils se détruisent
- A l'expédition d'un message, conservation d'une copie de l'anti-message correspondant

Valeurs à sauvegarder

- Événements traités
- Messages envoyés et reçus
 - Date émission
 - Date réception
 - Emetteur
 - Récepteur
 - Signe
 - Données





Réception d'anti-messages

- Si le message est dans la file d'entrée du destinataire et non traité : annulation
- Sinon même procédure que pour la réception d'un message du passé : rollback

[Problème du rollback]

- Malgré les retours-arrière,
 - le système évolue-t-il globalement ?
 - quantité mémoire pour stoker les historiques sera-t-elle toujours suffisante ?
 - Comment détecter la terminaison ?
 - Comment gérer les entrées/sorties en tenant compte des retours-arrière ?

[Solution : *Temps virtuel global*]

- Sur tous les processeurs, il existe un temps minimale au-dessous duquel la simulation ne pourra jamais revenir.
- Utilité de ce Temps Virtuel Global ?
 - Libération de la mémoire occupée par les sauvegardes
 - Valider les actions définitivement
- Détection de la terminaison ?
 - File d'attente vide : $+\infty$
 - Toutes les files d'attente sont vides : $TVG = +\infty$
- Un algorithme de TVG par Samadi en 1985

[Calcul du TVG ou Min globale]

- Minimum de toutes les estampilles des messages non encore reçus et des temps locaux
 - Une approximation est suffisante
 - Calcul fréquent est nécessaire pour optimiser la mémoire
 - Ce calcul ralentit l'exécution

[Optimisation du rollback]

- Gafni en 1988 : *« l'arrivée d'un message en retard ne modifie pas suffisamment les calculs pour que les messages déjà envoyés soient faux »*
 - Annulation paresseuse : attente de l'effet de la réception d'un message en retard
 - Si provoque les mêmes messages que précédemment, OK
 - Sinon, on envoie un anti-message
 - Re-évaluation paresseuse : on s'occupe de l'état des processeurs en pas des messages
 - Si un message en retard arrive et provoque les mêmes résultats que précédemment, on repart là où l'on était

Avantages de ces optimisations

- Le speedup est plus élevé que le nombre de processeurs impliqués dans le calcul
 - Exemple : 2 processeurs - vitesse de la simulation distribuée doit être 2 fois + rapide qu'en séquentielle
 - Avec optimisation
 - plus de 2 fois + rapide avec 2 processeurs

Difficultés de l'approche optimiste

- Va-t-on passer plus de temps à faire des retours-arrière que des exécutions de la simulation ?
 - Jefferson en 85 :
Temps passé à faire des rollback = temps d'attente dans l'approche pessimiste
- Sauvegarde périodique des états du système
 - Nécessité de plus de mémoire dans l'approche optimiste qu'en pessimiste
- Tolérance et correction des erreurs pouvant être causées par une exécution erronée et annulée par un rollback
 - Pour l'instant, laissée à la charge de l'utilisateur
- Mise au point de la fréquence des sauvegardes des vecteurs d'états, gestion de la mémoire, ...

Performances

IL N'EXISTE PAS DE SOLUTION UNIVERSELLE

- Approche pessimiste :
 - File d'attente sur 5 processeurs : aucun gain
 - 292 voitures sur 292 routes, 33 Transputers: 19
- Approche optimiste :
 - Comportement des fourmis sur 32 proc. : 13
 - Simulation logique
 - sur 32 proc. : 25
 - sur 8 proc. : 2
 - Simulation de réseaux d'interconnexion : plus lent que l'approche pessimiste

Bibliographie

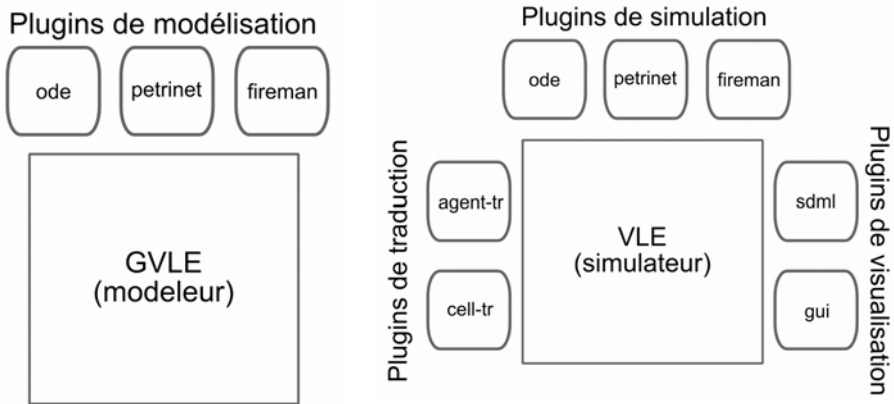
- Lamport 78 – *Time, clocks and the ordering of events in a distributed system* – Communication of the ACM, Vol 21,7
- Jefferson 85 – *Virtual Time* – ACM Transactions on Programming Languages and Systems, Vol 7,3
- Samadi 85 – *Distributed simulation, algorithms and performance analysis* – Thèse de doctorat, University of California, Los Angeles, USA
- Misra 86 – *Distributed discrete-event simulation* – Computing Survey, Vol 18,1
- Gafni 88 – *Rollback mechanisms for optimistic distributed simulation systems* – Proceedings of the SCS Multiconference on Distributed Simulation

VLE = Virtual Laboratory Environment

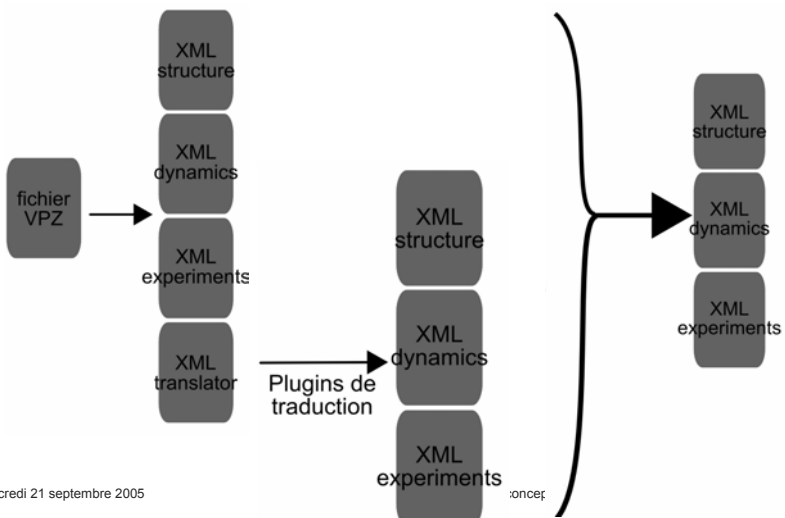
Objectifs de VLE

- Plate-forme de modélisation et simulation multi-formalisme
 - Couplage de modèles hétérogènes (ODE, FSA, ...)
- *Framework DEVS (Discrete Event Specification)* et ses extensions :
 - Automate cellulaire : Timed Cell-DEVS,
 - Structure dynamique : DS-DEVS,
 - ...
- Mapping / wrapping :
 - Traduction du formalisme du modélisateur en DEVS
 - Encapsulation du simulateur dans une enveloppe fonctionnelle DEVS

Architecture (1)



Architecture (2)



[VLE]

- VLE est distribué sous la Licence GPL : Open Source et Free (au sens GNU).
- VLE est développé pour Unix/Linux et pour un système X11
- La version Windows arrive ... bientôt.
- Le site Web de VLE, avec les sources, la documentation et des exemples : <http://vle.univ-littoral.fr>
- Pour les développeurs, le CVS repository : <http://www.sourceforge.net/projects/vle>
- `cvs -d:anonymous@cvs.sf.net:/cvsroot/vle co vle`
- Si vous voulez devenir développeur, envoyez un mail à : quesnel@lil.univ-littoral.fr