UNIVERSITÉ DE MOHAMMED KHEIDAR BISKRA

Faculté des Sciences et Sciences de l'Ingénieur

Département d'Informatique

MÉMOIRE

Pour obtenir

LE DIPLOME DE MAGISTER EN INFORMATIQUE

Spécialité : Intelligence Artificielle et Image

présenté par

Bachir SAÏD

Programmation visuelle Des modèles de simulation à évènements discrets

Vers la pris en compte de la dimension de groupe

Soutenue le……..
Devant le jury composé de

Pr. M.BENMOHAMMED	Président	Professeur	Université de Constantine
Dr. A.BILAMI	Examinateur	Maître de Conférence	Université de Batna
Dr. A.ZIDANI	Examinateur	Maître de Conférence	Université de Batna
Dr. B.BELATTAR	Rapporteur	Maître de Conférence	Université de Batna

Remerciements

Je tiens à remercier ici les personnes qui, par leurs conseils et leurs encouragements ont contribué à l'aboutissement de ce travail.

Mes premiers remerciements iront à ***Dr Brahim Belattar****, mon directeur de mémoire qui m'a fait bénéficier de son expérience et de ses connaissances.*

Je remercie également les membres de jury :

Monsieur ***BenMohamed Mohamed****, Professeur à l'université de Constantine (président).*

Monsieur ***Zidani Abd El Madjid****, Maître de conférence à l'université de Batna (examinateur).*

Monsieur ***Bilami Azeddine****, Maître de conférence à l'université de Batna (examinateur).*

J'adresse aussi un remerciement à ***Mr Med Caouki Babahnini*** *et à* ***Mr Hammadi Benneoui*** *(département de l'informatique) pour leur aide tout au long de ma formation.*

Enfin, je tiens à remercier ma famille, ma compagne et mes collègues à l'université de Ouargla pour le soutient continue qu'ils m'ont apporté.

CHAPITRE 6

Conclusion générale

Dans ce travail nous avons précisé comme objectif la proposition d'une approche pour la programmation visuelle des modèles de simulation avec la prise en compte de la dimension de groupe (le travail collaboratif). Alors nous avons suit des étapes dans notre étude afin d'atteindre l'objectif spécifié, nous avons présenté d'abord les fondements de la programmation visuelle et les différentes classifications, puis nous avons fait une étude détaillée sur les principes de la modélisation et la simulation et les étapes d'un projet de simulation, cette étude est enrichie par l'explication de l'approche de l'intégration de la dimension de groupe (travail collaboratif) dans les environnements de modélisation et simulation. Avons la proposition de notre approche nous avons préparé un bilan sur des travaux qui traitent la programmation visuelle dans le domaine de simulation. D'après ce que nous avons trouvé dans les ces études, nous avons proposé une approche pour la programmation visuelle des modèles de simulation avec la prise en compte de la dimension de groupe, le formalisme DEVS est choisi comme cas d'étude à cause que les modèles DEVS sont décomposable et ont une structure hiérarchique. Pour valider cette approche nous avons implémenté un prototype qui est nommé VISDEVS, VISDEVS est un environnement implémenté par le langage Java à l'aide de l'environnement JBuilder, notre prototype est caractérisé par :

- La prise en compte de toutes les étapes d'un projet de simulation.
- Il fonctionne en mode collaboratif distant via un réseau Internet, et peut être employé par n'importe quel nombre d'utilisateurs dans des différents emplacements.
- Il profite l'avantage de la programmation visuelle dans la simplification de la tâche de la programmation des modèles.
- La séparation de ces différents composants, ce qui facilite la modification et le développement, alors c'est un environnement ouvert et extensible.
- Il permet la construction des modèles réutilisables, c'est-à-dire qu'on peut réutiliser des modèles stockés comme des sous-modèles.

Limites

Notre travail est un prototype, il est implémenté pour valider l'approche que nous avons proposée, malgré que ce prototype donne des bons résultats et répond aux besoins spécifiés précédemment, il à des limites. Le formalisme DEVS est capable de représenter n'importe quel système complexe, pour cela les modèles DEVS qui doivent être simulé par *devsjava* sont réellement écrits en langage Java qui est un langage puissant, alors on a toujours besoin d'ajouter des variables et des instructions en langage Java dans les classes correspondants aux modèles DEVS générés par l'éditeur de la programmation visuelle *prog_vis*. Cette limite concerne à la nature des techniques de la programmation visuelle, jusqu'aujourd'hui la programmation visuelle n'arrive pas à remplacer la programmation textuelle mais tous les essais déversent dans l'objectif de diminuer l'utilisation de la programmation textuelle.

Perspective

Le prototype que nous avons implémenté consiste à réaliser des modèles de simulation définissent par le formalisme DEVS classique, en utilisant les techniques de la programmation visuelle et prenant en compte la dimension de groupe. Nous signalons ici que ce travail à notre connaissance est le premier qui essaye d'utilise la programmation visuelle dans la construction des modèles de simulation DEVS avec la prise en compte de la dimension de groupe.

Après l'étude que nous avons fait, nous pouvons dire que notre travail peut avoir un développement et des améliorations dans plusieurs axes :

- Ce travail peut être exploité par des experts de modélisation par le formalisme DEVS afin de définir une grande bibliothèque qui contient des modèles DEVS visuelle fréquemment utilisé.
- Le prototype doit être généraliser pour traiter tous les type de formalisme DEVS (Cell_DEVS, DSDEVS, VDEVS, …).
- Nous ponçons au future qu'il y a la possibilité d'utiliser la programmation visuelle dans la modélisation de n'importe qu'elle type de modèles connus.
- Le prototype que nous avons implémenté est un outil de modélisation, nous avons alors toujours besoin d'utiliser l'environnement DEVSJAVA pour simuler les modèles construits par notre prototype. Pour cela pensons à développer un simulateur et visualisateur équivalent à DEVSJAVA afin de pouvoir lancer la simulation des modèles directement à partir de l'éditeur de la programmation visuelle *prog_vis*.

CHAPITRE 1

Introduction générale

Aujourd'hui plusieurs travaux de recherche concernant le domaine de la simulation sont élaborés, à cause de sont profit dans de nombreux domaines scientifiques et industriels à l'analyse des systèmes complexes. La modélisation de n'importe quel système et la simulation de son comportement exige le choix des modèles de simulation adéquats, mais cela implique un certain niveau de détaille, alors qu'on risque d'avoir des difficultés lors sa réalisation. A plus pour que les résultats de la simulation soient solides, l'expérimentation des modèles nécessite un temps et des coûts importants.

La simulation par ordinateur sert à réduire quelques difficultés, mais malheureusement elle nécessite que les modèles soient programmés en utilisant des langages spécifiques pour la simulation LSs. Ce qui nécessite des efforts supplémentaires, pour la maîtrise de ces langages.

Au cours des vingt dernières années l'informatique a vu la naissance d'un nouveau domaine de recherche, c'est la programmation visuelle.

La programmation visuelle fait référence à tout système qui permet de spécifier un programme d'une façon multi-dimensionnelle, les éléments fournis par les environnements de programmation visuelle pouvant être manipulé interactivement par l'utilisateur (programmeur). La programmation visuelle est devenue aujourd'hui très utile, elle a donné une nouvelle vue au développement des environnements dans différents domaines scientifiques.

Le domaine de simulation est l'un de ces domaines, et l'intégration des concepts de la programmation visuelle et la visualisation devienne l'objectif des recherches qui lui correspondent.

Dans ce qui suit nous allons discuter une nouvelle vu sur l'intégration de ces concepts dans le domaine de simulation.

Problématique

Dans la simulation par ordinateur il y a une phase qui s'appelle la programmation du modèle, la programmation textuelle était toujours la solution utilisée pour la construction des modèles de simulation, elle est classique et a des inconvénients, tel que la difficulté de la mise en œuvre et la mise à jour, la complexité, et de plus l'incompréhensibilité de l'affichage des résultats lors de l'exécution.

Alors pour faciliter l'utilisation de la modélisation et la simulation, la programmation visuelle et la visualisation sont adoptées comme outils de construction des modèles de simulation, et de représentation du comportement des modèles durent leur simulation.

Dans un autre axe de recherche, un nouveau concept est ajouté à la simulation, c'est la dimension de groupe où l'objectif est d'avoir la possibilité de réaliser des projets de simulation en télécollaboration, c'est-à-dire les membres d'un groupe de simulation peuvent travailler en collaboration dans des emplacements géographiquement séparés afin de réaliser leurs projets. Cette approche est motivée par l'évolution de la technologie des réseaux de communication d'un côté et la nature des projets de simulation qui exige la présence d'un groupe d'experts, tandis qu'ils ne sont pas dans le même local.

Objectif

Dans ce travail nous avons comme objectif la proposition d'une approche pour la programmation visuelle de modèles de simulation à évènements discrets en prenant en compte la dimension de groupe. Nous allons alors proposer une architecture détaillée d'un environnement de modélisation et simulation et son mode de fonctionnement, en couplant les concepts de la programmation visuelle avec celle de la dimension de groupe.

Notre étude se base sur la proposition d'une approche pour la conception et la mise en œuvre d'un environnement de programmation visuelle des modèles de simulation à évènements discrets en prenant en compte la dimension de groupe.

Les différentes parties de ce mémoire sont résumées dans le paragraphe suivant.

Organisation du mémoire

Pour pouvoir atteindre l'objectif cité dans 1.2, notre étude doit recouvrir les trois axes suivants :

- Programmation visuelle,
- Simulation,
- Collaboration (dimension de groupe),

Ainsi notre mémoire sera organisé en cinq chapitres. Le premier chapitre est une introduction (introduction générale, problématique, objectif, organisation du mémoire).

Dans le deuxième chapitre les fondements de la programmation visuelle et l'état de l'art des langages de programmation visuels sont présentés.

Dans le troisième chapitre nous allons expliquer la théorie de la modélisation et la simulation puis nous présentons les caractéristiques des environnements collaboratifs de modélisation et de simulation, et les différentes fonctionnalités requises d'un collecticiel de modélisation et de simulation.

Le quatrième chapitre commence par une étude sur quelques travaux qui traitent de l'utilisation de la programmation visuelle dans le domaine de simulation en axant sur ceux qui basent sur le langage Java. Puis nous proposons une approche pour la conception d'un collecticiel de modélisation et simulation des modèles DEVS en utilisant la programmation visuelle et en prenant en compte la dimension de groupe dans toutes les étapes de l'étude de simulation.

Le cinquième chapitre est consacré à la mise en œuvre d'un prototype de collecticiel de modélisation et de simulation qui permet la programmation visuelle des modèles de simulation DEVS en mode collaboratif, ce prototype fournit les fonctionnalités requises et il est réalisé pour valider l'approche que nous avons proposée dans notre conception.

CHAPITRE 4

Conception de l'environnement de la programmation visuelle

Depuis l'apparition de la programmation visuelle, de nombreux travaux visant à exploiter ses avantages ont été lancés.

Dans le domaine de simulation plusieurs environnements de programmation visuelle sont développés afin de faciliter la programmation des modèles de simulation et de permettre aux non programmeurs de construire leurs modèles sans écrire une seule ligne de code. Mais jusqu'à ce jour ces environnements restent mono-utilisateur et ne prennent pas en compte la dimension de groupe.

Notre travail consiste donc à proposer une approche pour la mise en œuvre d'un environnement de programmation visuelle pour la simulation prenant en compte la dimension de groupe.

Dans ce chapitre nous allons présenter l'architecture de notre environnement, définir ses différents composants et expliquer son fonctionnent.

4.1. L'existant

4.1.1. JBDS

JBDS (JavaBeans Discrete Simulation)[Fukunari98a] [Fukunari98b] est un système de simulation discrète qui est basé sur la programmation orientée composants (JavaBeans), il supporte la programmation visuelle et contient une bibliothèque de composants de base pour les modèles de simulation.

Le système JBDS comporte trois éléments de base; entité, évènement, et nœuds (Figure 4.1).

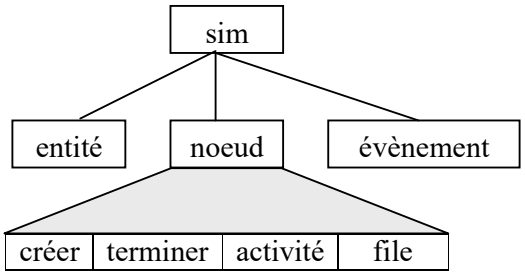

Figure 4.1 : Architecture de JBDS

Entité

Les entités sont des classes de java. Elles se déplacent à travers les noeuds et transportent des informations. Les entités sont habituellement créés, déplacées à travers les noeuds suivant un mécanisme d'évènement, puis détruites quand elles quittent le système.

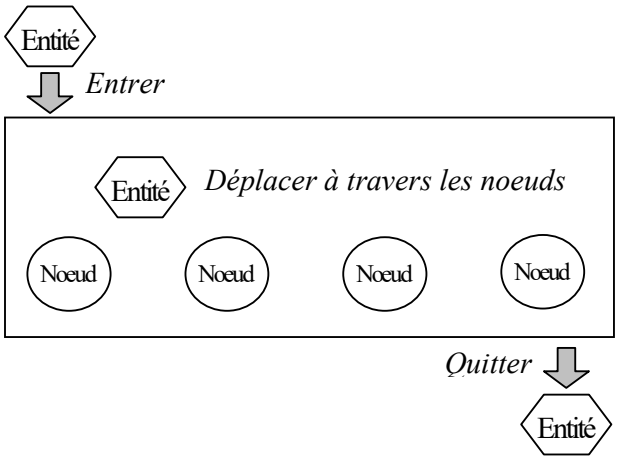

Figure 4.2 : architecture d'un système dans JBDS

Evènement

Les évènements sont des classes de java utilisés par les noeuds. Les événements dans JBDS s'occupent de la communication entre les noeuds.

Nœuds

Les noeuds sont des JavaBeans, il y a trois types différents; des noeuds de base, qui créent, gèrent la file d'attente, déplacent, et terminent des entités; un noeud de contrôle, qui gère l'ordonnancement en organisant les évènements futurs; et des noeuds de prise de décision, qui sont employés quand un modèle de simulation comporte une décision conditionnelle ou probabiliste.

JBDS utilise les concepts de CBSD (Component-Based Software Development) pour développer les modèles de simulation. L'utilisateur peut rapidement développer des modèles sans écrire un code. JBDS peut être aussi utilisé dans un environnement distribué pour développer des modèles complexes de simulation.

Voici à la fin l'application BEANBOX qui permet de construire et exécuter les modèles de simulation.

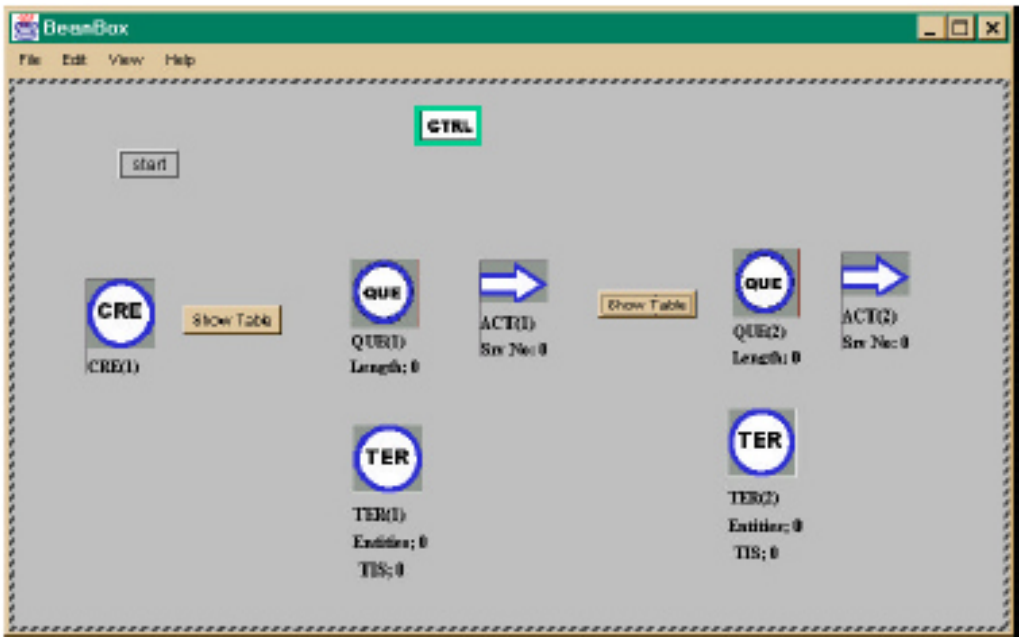

Figure 4.3 : Application de simulation

4.1.2. JSIM

JSIM [Miller98a][Miller98b][Ge98][Zhang97][Zhao97] est un environnement de simulation et animation qui est basé sur Java, il a été développé à l'université de Georgia. La dernière version de JSIM incorpore la nouvelle approche de développement de logiciel : la technologie basée sur les composants.

JSIM a trois packages de base, ***queue*** pour la gestion de la file d'attente, ***statistics*** pour la collection des informations statistiques et ***variates*** qui fournit une grande variété de variables aléatoires. La classe de variable aléatoire est une classe qui est étendue pour toutes autres variables aléatoires. Ces packages sont généralement utiles, dans et hors de la simulation. Pour supporter les approches de simulation (par évènement et par processus) JSIM contient deux packages *event* et *process*. Le package de *qds* fournit un moyen commode pour les données d'accès/génération de la simulation. Un autre package *jmodel* celui qui nous intéresse, il fournit un concepteur visuel pour le package *process*. Le modèle conçu peut être animé quand la simulation est lancée. Il permet aux utilisateurs de positionner un objet de simulation sur un canvas de construction de modèle en choisissant un bouton à partir de la barre d'outil et puis en cliquant sur un endroit sur le canvas pour placer l'objet.

Les boutons suivants sont actuellement fournis :

Server: fournit le service aux entités arrivés au noeud.
Facility: hérité de Server et ajoute une file d'attente pour retenir les entités en attente.
Signal: modifie le nombre d'unités de service dans le(s) serveur(s).
Source: produit des entités avec des temps entre arrivées aléatoires.
Sink: consomme des entités et enregistre des statistiques correspondantes.
Transport: connecte deux noeuds entre eux.
Move: transfert des noeuds dans de nouvelles positions dans le canvas.
Delete: supprime des noeuds ou des bords en cliquant sur eux.
Update: affiche / change les propriétés des noeuds choisis.
Generate: génère le code Java implémentant le modèle conçu.

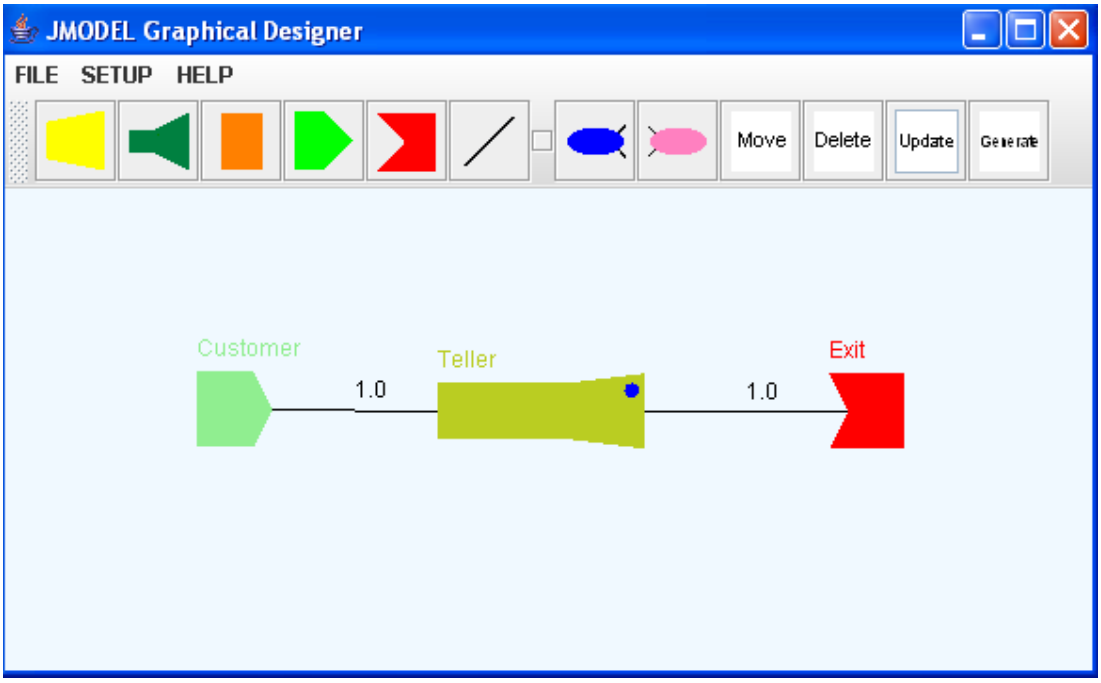

Figure 4.4 : Le concepteur graphique des modèles (JMODEL)

Avec JSIM on peut avoir une animation des modèles conçus, les conceptions de JSIM peuvent être animées en réutilisant une grande partie du code utilisé pour le concepteur visuel. Le constructeur de modèle de simulation peut tracer les mouvements des entités de simulation à travers le modèle.

4.1.3. SimBeans

SimBeans [Miroslav97] [Praehofer98] [Praehofer99a] [Praehofer99b] est un prototype de simulation conçu pour exécuter des simulations construites par des composants de (JavaBeans) et adoptant le formalisme DEVS. Ses extensions pour la simulation continue et mixte. Par conséquent, cette approche convient aux utilisateurs déjà familiers avec DEVS. Le but était de rendre des composants de simulation réutilisables dans différents contextes, adaptables à différentes applications, et extensibles pour des besoins particuliers imprévus.

Dans le prototype de simulation SimBeans, la réalisation des systèmes de simulation est basée sur les idées et les concepts suivants:

- Un ensemble de modules élémentaires sont fournis pour la modélisation et la simulation, pour l'évaluation statistique, et pour la visualisation.
- Une bibliothèque d'objets est fournie. Grâce à ces objets, des composants de modèle peuvent être adaptés afin de répondre à des exigences particulières.
- On définit des interfaces de modèle qui indiquent où et comment les composants du modèle peuvent être utilisés. Les caractéristiques de l'interface obéissent à une hiérarchie de classification permettant de définir la compatibilité entre les composants du modèle.
- Les interfaces et la classification des composants de modèle basé à l'interface sont utilisées pour définir des modèles
  génériques pour les modèles couplés qui définissent les interfaces des composants et la structure de couplage mais pas les composants eux-mêmes. Lors de la conception, ces composants génériques peuvent être configurés par l'instanciation des composants modèles qui obéissent aux interfaces.
- Les systèmes de simulation sont principalement ascendants construit par la composition et le couplage hiérarchique des composants de modèle.
- La bibliothèque des composants peut facilement être étendue pour répondre aux besoins spéciaux.
- Utilisant la bibliothèque des composants élémentaires, des environnements de simulation pour des domaines d'application particuliers peuvent être réalisés.

Selon ces idées, nous distinguons les manières suivantes pour la composition:

- **Sélection:** choisis les composants concrets de modèles dans un modèle couplé pour lequel seulement les interfaces sont indiquées d'une manière descendante.
- **Personnalisation:** adapte les composants de simulation pour répondre à des différentes exigences en utilisant les composants utilitaires, par exemple, fonctions de distribution aléatoires, stratégies de commande.
- **Couplage:** couple les modèles d'une manière ascendante hiérarchique.
- **Attachement:** attache les composants pour la sortie de la simulation, le calcul statistique, la visualisation et l'animation.

Pour son architecture SimBeans est un ensemble de composants de modèles ainsi que des composants pour des visualisations et des animations, qui peuvent être utilisées pour construire des systèmes de simulation à évènements discrets. Tous ces composants sont réalisés comme JavaBeans. La figure 4.5 montre l'architecture multi-couches de SimBeans [Praehofer99b] :

Outils de simulation d'applications spécifiques
• Cadre pour le développement d'outil de simulation

Composants de simulation d'applications spécifiques
• Par exemple machines, transporteurs, stockages, véhicules, …

Composants de base de simulation
• Composants élémentaires de modèles (par exemple processeurs, files d'attente, retards,…). • Schémas de couplage. • Stratégies de commande. • Composants de sortie (visualisation, animation…).

Noyau de simulation
• Algorithme de simulation: manipulation d'événement et intégration numérique. • Variables d'état et évènements de changement. • Collectes de données et fonctions utilitaires. • Sorties de base.

Java / JavaBeans et extension
• Mécanisme de délégation. • Manipulation de propriété asynchrone d'événement de changement.

Figure 4.5 : Architecture multi-couches de SimBeans

Couche1

La couche la plus basse est le langage de programmation Java et le modèle de composant JavaBeans. Les spécifications des Beans ne définissent pas la sémantique de l'ordre d'invocation et la synchronisation des manipulateurs d'événement multiple en entendant le même événement. Une implémentation de défaut existe pour la manipulation synchrone d'événement (classe *PropertyChangeSupport*). Le couplage des Beans est habituellement fait au moyen d'objets adaptateurs qui doivent être définis par une classe individuelle adaptateur menant à une prolifération des classes. Avec l'aide des dispositifs de la réflexion de Java, une classe *Delegate* est implémentée, qui peut fournir des événements à n'importe quel objet cible.

Couche2

La couche noyau de simulation fournit les concepts d'infrastructure et d'implémentation de simulation pour les composants de simulation. Cette couche est spécifique pour différents types de simulations, par exemple, il y a une infrastructure pour la simulation à évènements discrets, pour la simulation continue et pour la simulation combinée.

Couche3

C'est la couche principale contenant les composants élémentaires de simulation à partir des quels des systèmes de simulation sont construit. Elle est spécifique pour le domaine d'application, par exemple, il y a une bibliothèque pour la simulation orientée processus discrète classique.

Couche4

En utilisant les composants élémentaires de simulation, on peut construire des composants spécifiques d'application. Ils ont une visualisation et une interface utilisateur personnalisée qui est significative pour l'ingénieur d'application.

Couche5

Au sommet de la hiérarchie des couches il y a les systèmes ou les environnements de simulation d'application spécifique. Ils sont obtenus par accumulation des composants de la couche inférieure en tant que programmes autonomes. Ils sont spécifiques pour l'application actuelle et fournissent une interface utilisateur dans le contexte de l'ingénieur d'application. Nous distinguons entre les systèmes de simulation, qui ont un modèle simple de simulation, ils sont souvent une partie d'une plus grande application, et les environnements de simulation, qui permettent de construire et d'examiner les différentes configurations par la composition des composants d'une bibliothèque.

Pour le coté de la programmation visuelle dans SimBeans il y a des composants graphiques. Ils fournissent des blocs de base pour construire des modèles de simulation discrets grâce aux JavaBeans. Ces blocs incluent les Beans suivants:

- *ModelControl.* La barre d'outils de base pour l'expérimentation de la simulation avec l'information sur le temps courant, le temps maximal, et des boutons pour exécuter et déboguer le modèle. Il est possible d'exécuter pas à pas la simulation (événements simples ordonnés pour le traitement).
- *SimProcessGenerator.* Classe abstraite pour les Beans qui fournit la création de nouveaux processus. Chaque générateur de processus est relié à une classe des processus (descendants de la classe de *SimProcess*).
- *BulkProcessGenerator.* Produit un nombre spécifié d'instances pour la classe de processus associée.
- *RandomProcessGenerator.* Produit une séquence des processus avec une distribution spécifiée des intervalles entre leurs activations. La distribution est représentée par une classe mettant en application l'interface *distribution*.
- *ProcessObserver.* Une Listbox montrant l'état actuel de tous les processus observés. Il peut y avoir plusieurs observateurs de processus dans l'application, un pour chaque de processus.

4.1.4. VisSim

VisSim/à évènements discrets [Schwet97][Dilley03][Ptv-vis] est un environnement de modélisation que les analystes de simulation peuvent employer pour construire des modèles orientés processus pour des systèmes complexes. L'environnement utilise une interface utilisateur graphique puissante qui aide l'analyste à construire des modèles et à analyser les résultats. La facilité d'utiliser l'interface et l'ensemble compréhensif d'objets de simulation signifie que des modèles conçus des systèmes spécifiques peuvent être aisément construits.

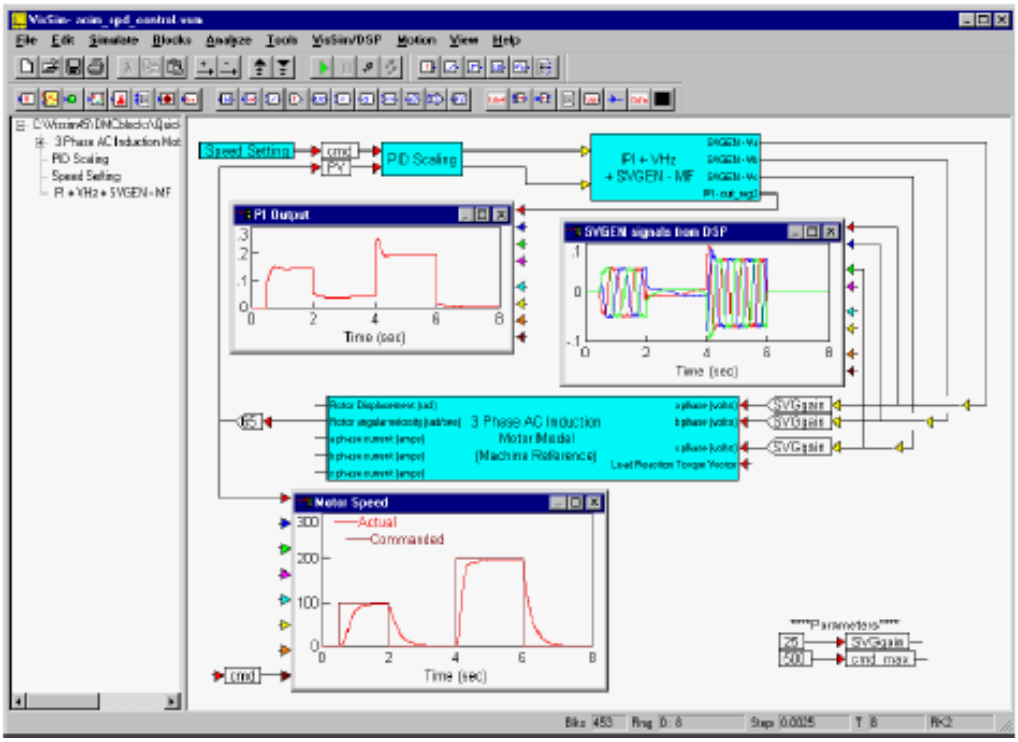

Figure 4.6: Interface de VisSim

Un modèle de simulation à évènements discrets se compose typiquement des entités actives et des ressources simulées, dans VisSim les entités actives sont appelées *tâches,* et le comportement d'une tâche est indiqué par un *graphe des tâches* . Le *graphe des tâches* indique réellement le comportement de l'ensemble des tâches semblables.

Un modèle dans l'environnement de VisSim se compose d'un certain nombre d'objets de simulation et d'un ou de plusieurs graphes de tâches. Il y a deux genres de graphes de tâches: les graphes ouverts et les graphiques fermés. Dans un graphe ouvert, le noeud *taskSource* produit une nouvelle instance de la tâche; le taux de génération de tâche est commandé par les paramètres du noeud. Les graphes ouverts contiennent souvent le noeud *taskSink*, pour rassembler (ou absorber) des tâches réalisées. Le graphique fermé contient le noeud *sourceSink*. Le noeud *sourceSink* produit un nombre de tâches fixe au début de l'exécution du modèle. Après cette première phase, des tâches entrantes dans ce noeud sont retardées pendant une période et puis émergent en tant que nouvelles tâches. Le graphe de tâches se compose d'une collection de blocs (ou de noeuds) reliés par des arcs.

Les blocs contrôlent le comportement des tâches et permettent aux tâches de simuler le comportement des entités dans le système réel modélisé. Les genres de blocs qui contrôlent le comportement de tâche incluent:

- taskDelay - retard pour un intervalle de temps d'une tâche.
- route - choisir un chemin selon des probabilités de branchement.
- computeResult - modifier les valeurs des variables
- evaluateRelation - tester la valeur d'une variable
- des bloques qui mettent en référence des objets de simulation.

Le schéma suivant illustre la spécification et l'opération d'un graphe de tâches simple. Le bloc de *taskSource* émet plusieurs fois. L'intervalle entre chaque nouvelle tâche est déterminé par les attributs du bloc qui sont indiqués dans une boite de dialogue. Dans

l'exemple, chaque nouvelle tâche voyage d'abord (visite le) bloc de taskDelay. Le temps simulé passé dans ce bloc est déterminé, pour chaque tâche, par les attributs du bloc; ces attributs sont de nouveau indiqués dans une boite de dialogue.

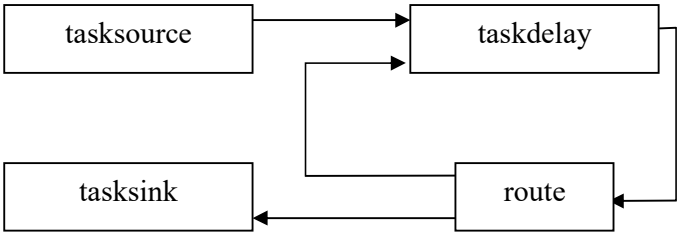

Figure 4.7 : Graphe des tâches pour VisSim

Après l'expérience d'un retard, chaque tâche visite après le bloc *route*. Ce bloc a une entrée (pour les tâches) et deux sorties pour les tâches. Une table de routage probabiliste conduit les tâches entrantes à un des connecteurs de sortie au bloc; l'i-ème entrée dans cette table indique la probabilité de sortir via l'i-ème connecteur de sortie du bloc. Dans l'exemple, la tâche peut retourner au bloc *taskDelay* (via le connecteur 0) ou se déplacer au bloc *taskSink* (via le connecteur 1). Le bloc *taskSink* absorbe ou termine chaque tâche entrant le bloc.

Voici un exemple de modèle d'un système informatique traitant des tâches informatiques.

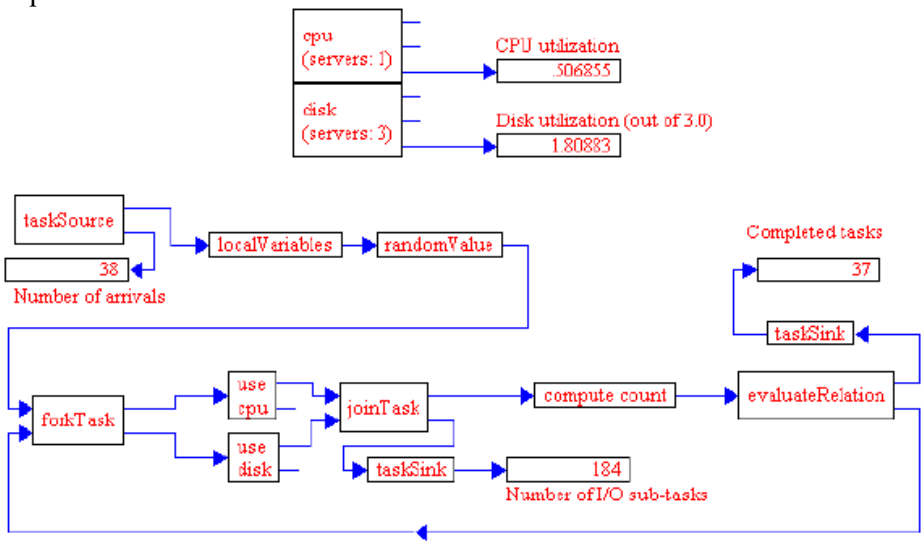

Figure 4.8 : modèle de système d'ordinateur dans VisSim

4.1.5. VSE

VSE (Visual Simulation Environment) est un environnement visuel de simulation, le développement du prototype de VSE est commencé en Virginia à Juillet 1995[Balci97a] [Balci97b] [Balci97c][Bachelet98][Orca].

L'architecture du modèle de VSE se compose d'une partie statique et une autre dynamique. L'architecture statique du modèle se compose des composants décomposés hiérarchiquement [Balci97a] [Balci97c]. L'architecture dynamique du modèle se compose des objets dynamiques. Un objet dynamique est une entité qui se déplace physiquement ou logiquement d'un point à un autre dans un modèle. Un objet dynamique peut être décomposé

en hiérarchie des composants semblables à la décomposition de l'architecture statique du modèle.

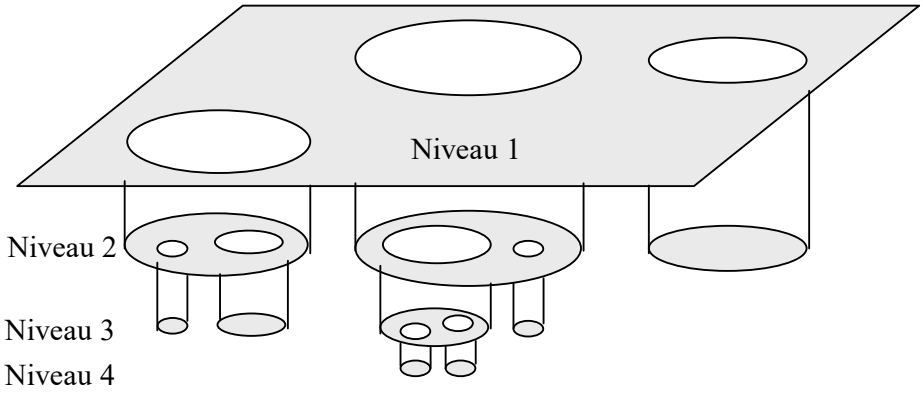

Figure 4.9 : La décomposition hiérarchique d'un modèle

VSE fournit deux genres de composants: shallow (peu profond) et deep (profond). Un composant shallow est un composant qui n'a aucune décomposition. Un composant deep est un composant qui a une disposition (ou une décomposition). Les composants shallow représentent les noeuds feuilles (les noeuds qui ne sont pas encore décomposés) de la décomposition hiérarchique. Les composants deep représentent les noeuds de la hiérarchie qui sont décomposés.

VSE se compose de quatre outils :

VSE Editor: Nous laisse construire notre modèle de simulation graphiquement employant le paradigme orienté-objet, avec l'héritage, le passage de messages, et l'encapsulation.

VSE Simulator : Fournit l'animation et nous laisse exécuter des expériences avec notre modèle de simulation.

VSE Output Analyzer : Nous laisse analyser statistiquement nos données de sortie de simulation.

VSE Teacher : Utilise le navigateur Web par défaut et nous laisse apprendre VSE en regardant les clips vidéo et par la navigation par hypermédia des informations techniques, et d'aide.

Un modèle dans VSE est graphiquement structuré et de façon hiérarchique. Nous pouvons intégrer une image graphique dans VSE Editor dans différents formats comprenant EPS et TIFF. L'image peut représenter décore de notre modèle, ou une partie, sur laquelle des objets peuvent être animés. Nous pouvons aussi redimensionner et repositionner l'image dans le VSE Editor.

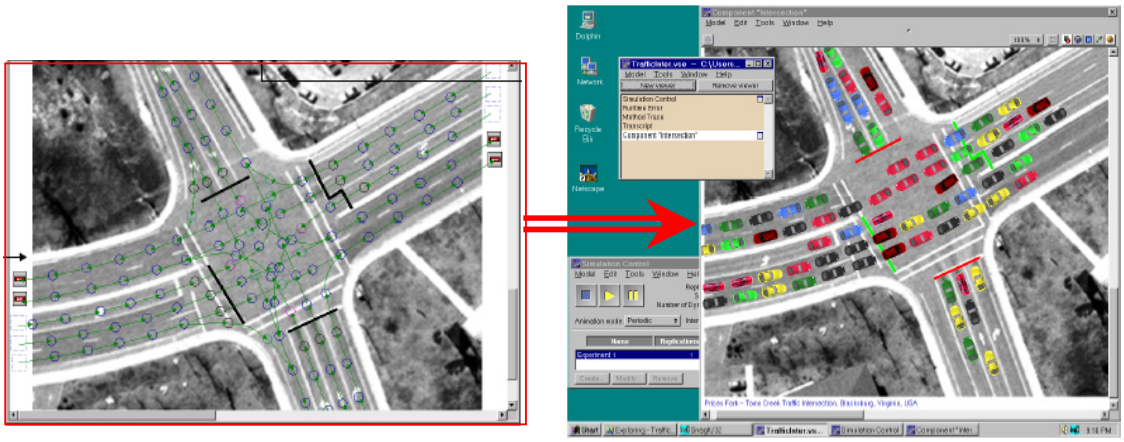

Figure 4.10 : Fenêtre de VSE Editor

La représentation graphique d'un composant de modèle de VSE peut être construite en utilisant n'importe laquelle des techniques suivantes [Balci98]:

Diagramme: Une représentation diagrammatique d'un processus logique peut être utilisée comme un arrière plan d'un composant de modèle sur lequel l'exécution logique peut être animée. Le processus logique peu représenter l'exécution d'un programme machine, le flot d'informations dans une organisation, ou les activités dans un processus d'affaires. Beaucoup de types de diagrammes peuvent être utilisés [Balci98]. La visualisation diagrammatique permet au modélisateur de représenter des activités et des procédures complexes facilement, et de communiquer la logique associée aux autres d'une façon concise, précise, et compréhensible.

Dessin: Un arrière plan de composant du modèle peut se composer des dessins créés en utilisant un logiciel de dessin professionnel tel que Adobe Illustrator, CorelDraw, et Free-Hand. Un tel arrière plan se fonde sur la connaissance et le talent du créateur en produisant un visuel significatif.

Icône: La composition d' arrière plan basé sur des Icônes est généralement utilisée par les produits de logiciels de simulation des domaines-spécifiques (par exemple, fabrication, santé, ...). Différents objets et tâches dans un domaine particulier sont représentés par des icônes. Le modélisateur construit un arrière plan de composant du modèle en choisissant des icônes et en les reliant pour montrer des interactions potentielles. La même fonctionnalité peut facilement être fournie en VSE. De tels objets iconiques peuvent être créés dans la fenêtre des modèles de VSE Editor comme des composants de modèle réutilisables et nous pouvons les réutiliser en les draguant depuis une palette d'icônes.

Carte: Une carte géographique, militaire, ou autre peut être utilisée comme un arrière plan de composant du modèle sur laquelle l'animation peut être conduite. Les cartes nous permettent de représenter une région dans les formes, les tailles, et les rapports corrects.

Peinture: Un arrière plan de composant du modèle peut se composer de peintures créées en utilisant un logiciel professionnel de peinture tel que Adobe Photoshop, Paint Shop Pro, et Painter.

Photographie: Une photographie vaut mieux que mille mots parce qu'elle est concise, précise, et claire pour la représentation d'un système complexe. Un composant de système complexe peut être photographié, la photographie peut être scannée, l'image scannée peut être retouchée, et l'image photographique peut être utilisée comme un arrière plan de composant modèle sur laquelle l'animation peut être conduite.

Schéma: Un arrière plan de composant du modèle peut se composer de dessins créés en utilisant un logiciel professionnel de conception assistée par ordinateur (CAO) tel

qu'AutoCAD et DesignCAD. Le dessin d'une conception de système peut être utilisé comme un arrière plan de composante du modèle sur lequel l'animation peut être conduite.

4.1.6. JDEVS

JDEVS est un environnement de modélisation et de simulation qui est développé par J.B.Filippi à l'université de Corsica à 2003 [Batti] [Philippi03a] [Philippi03b] [Philippi03c].JDEVS a été développé pendant plus d'une année pour servir de cadre expérimental aux techniques de modélisation des systèmes naturels.
Il permet le développement et l'exécution des modèles de simulation à évènements discrets, d'usage universel, orienté objet, orienté composant et des SIG de manière visuelle. L'implémentation des exemples de modèles montre que cet environnement expérimental peut être utilisé pour résoudre tous les problèmes complexes de simulation à évènements discrets mais il est approprié particulièrement à la simulation des systèmes naturels utilisant les réseaux de neurones artificiels.

JDEVS est basé sur le formalisme DEVS, il est développé en utilisant le langage Java. La boite à outils de JDEVS se compose de cinq modules indépendants, qui peuvent interagir ensemble.

Noyau de modélisation et de simulation

Le noyau de modélisation et de simulation est une implémentation simple en Java du formalisme DEVS (Ziegler 1990), contenant les modèles atomiques et couplés.

Interface utilisateur graphique

En utilisant l'interface utilisateur graphique, l'utilisateur peut graphiquement créer, compiler, lier et stocker les modèles atomiques et couplés, déboguer le modèle résultant et exécuter la simulation.

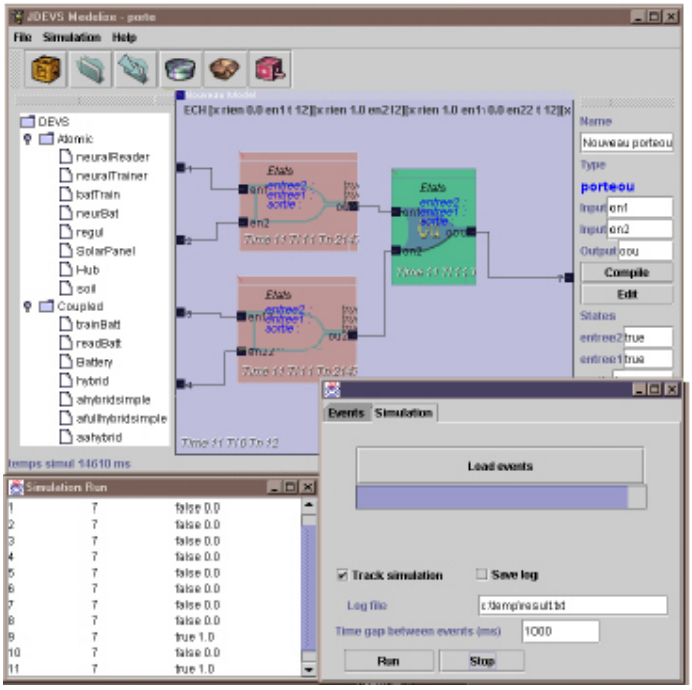

Figure 4.11 : Interface de modélisation et de simulation des bloques hiérarchiques par JDEVS

Le GUI permet également une modélisation distribuée, si différents modélisateurs travaillent sur des sous-modèles couplés et les stocke dans la même bibliothèque.

Bibliothèque des modèles de DEVS

Le but de la bibliothèque est de fournir un stockage et une réutilisabilité faciles des modèles. La conception de logiciel peut être vue comme base de données orientée-objet. En plus des liens structuraux, la bibliothèque garde le lien d'héritage et d'abstraction entre les modèles. Les modèles stockés dans la bibliothèque s'appellent les modèles de "context-out " (habituellement le code source). Pour rechercher un modèle, il est mis " dans le contexte ". Pendant cette phase l'état du modèle est placé de nouveau à l'état qu'il a quand il a été stocké. Plusieurs scénarios de simulation peuvent créer plusieurs modèles de " context-in " de la même description de modèle de " contexte-out ".

L'implémentation de la description de bibliothèque dans JDEVS a résultat un module dans le GUI. Ce module présente des modèles selon son domaine et sous-domaine, tous sont classifiés dans un arbre comme architecture.

Interconnexion de SIG (Système d'Information Géographique)

Pour créer des modèles cellulaires qui reproduiront un certain morceau de terre pour des modèles de propagation, nous avons besoin de définir une manière d'exporter des données du SIG et d'une procédure qui initialisera les cellules selon les données spatiales rassemblées de SIG.

Comme la simulation est en fait conduite, la carte n'est jamais entièrement mise à jour pendant l'exécution. La sortie de la simulation est un ensemble d'événements qui représente seulement ce qui a changé sur la carte avec le temps. Ceci réduit considérablement la taille de fichier d'enregistrement et accélère l'exécution, mais ce format ne peut pas être sorti directement au GIS.

Panneaux cellulaires de simulation

Ce module permet à l'utilisateur d'exécuter (et déboguer) la simulation d'un modèle cellulaire. L'utilisateur peut directement interagir avec la simulation, comme il peut envoyer des évènements en utilisant la souris. Les sorties du panneau de simulation cellulaire sont des cartes chronométrées qui sont importées de nouveau dans le SIG.

L'utilisateur peut interagir sur temps d'exécution de la simulation, avec un clique sur la carte il est possible de choisir une cellule et d'envoyer un événement spécifique. Un panneau de la simulation 3D a été également développé pour permettre une meilleure visualisation des phénomènes. Cet outil est particulièrement utile pour visualiser des flux d'écoulement de l'eau ou des modèles cellulaires tridimensionnels.

4.1.7. Synthèse

Nous avons cité quelques environnements visuels de modélisation et de simulation, la plupart sont basé sur le langage Java. L'objectif général de ces environnements est de rendre facile la construction des modèles de simulation et de fournir une visualisation plus proche du comportement des systèmes réels lors de l'exécution.

Nous remarquons que certains environnements ne supportent pas purement la programmation visuelle tel que JDEVS, VSE, alors que d'autres sont spécifiques pour un genre de système tel que JDEVS (systèmes naturels). On trouve des environnements qui peuvent être distribués sur un réseau tel que JBDS et JDEVS pour développer des modèles complexes, mais on ne trouve pas des environnements visuels de simulation qui prennent en compte la dimension de groupe (collaboration).

Ce bref bilan nous permet de tirer profit des travaux traitant de l'utilisation de la programmation visuelle dans la modélisation des systèmes et l'utilisation de la visualisation et l'animation dans l'exécution de la simulation.

Pour concevoir notre système nous allons nous inspirer de ces travaux en essayant d'introduire la notion de collaboration pour avoir à la fin une architecture d'un environnement de programmation visuelle des modèles de simulation basé sur le formalisme DEVS et qui prend en compte la dimension de groupe.

4.2. Le formalisme DEVS

B.P. Zeigler [Zeigler95] [Zeigler98] [Zeigler05] proposa un formalisme qui l'a nommé DEVS (Discrete Event System Specification) pour permettre la formalisation de modèles modulaires et hiérarchiques. Ce formalisme est basé sur la théorie des systèmes ; il permet une représentation formelle de modèles susceptible de manipulations mathématiques comparables aux équations différentielles pour les systèmes continus.

Il y a deux types de modèle DEVS, modèle DEVS atomique et modèle DEVS couplé. Un modèle atomique DEVS AM est une structure :

$AM = < X,S,Y,\delta_{int},\delta_{ext},\lambda, ta >$ Avec :

- X : $\{(p, v)|(p \in$ ports d'entrée, $v \in Xp)\}$: Ensemble de ports et de valeurs d'entrée pour la réception d'activations externes,
- Y : $\{(p, v)|(p \in$ ports de sortie, $v \in Yp)\}$: Ensemble de ports et de valeurs de sortie pour l'émission d'événements de sortie,
- S : L'ensemble des états séquentiels internes,
- $\delta_{int} : S \rightarrow S$ La fonction de transition interne qui place le modèle dans l'état suivant après le temps renvoyé par la fonction d'avancement du temps,
- $ta : S \rightarrow \mathbb{R}+$: La fonction d'avancement du temps qui renvoie le temps de vie de l'état courant (temps jusqu'à la prochaine transition interne),
- $\delta_{ext} : Q \times X \rightarrow S$ La fonction de transition externe qui programme les changements d'états en fonction d'activations d'entrées,
- $\lambda : Q \rightarrow Y$: La fonction de sortie qui génère des événements de sortie juste avant la transition interne.

Interprétation :

- $Q = \{(s, e)|(s \in S, 0 < e < ta(s))\}$ est l'ensemble total des états du modèle.
- e représente le temps écoulé depuis la dernière transition, et s l'ensemble partiel d'états pour la durée de $ta(s)$ en absence d'activation externe.
- δ_{int} : Le modèle étant dans un état s, à t_i il passera dans l'état s', $s' = \delta_{int}(s)$, en l'absence d'activation externe pendant la durée de $t_i + ta(s)$.
- δ_{ext} : Lorsqu'une activation externe se manifeste, le modèle étant dans un état s depuis le temps e passe en $s' = \delta_{ext}(s,e,x)$.
- L'état suivant dépend du temps passé dans l'état courant.
- A chaque changement d'état e est remis à 0.
- λ : La fonction de sortie est exécutée avant la transition interne, avant l'émission d'une sortie le modèle est dans un état passager (transient).
- Un état avec un temps de vie infini est un état passif, sinon, c'est un état passager (transient). Si un état s est passif, le modèle ne peut évoluer qu'avec l'apparition d'activations d'entrée.

Chaque état possible s ($s \in S$) a un *avancement de temps* associé et calculé par la *fonction de l'avancement de temps ta(s)* ($ta(s) : S \rightarrow \mathbb{R}^+$). L'*avancement de temps* est un nombre réel non négatif indique combien de temps le système reste dans un état donné en l'absence des événements d'entrée.

Ainsi, si l'état adopte la valeur s_1 au temps t_1, après $ta(s_1)$ unités du temps (c.-à-d. au temps $ta(s_1) + t_1$) le système fait une *transition interne* passant à un nouvel état s_2. Le nouvel état est calculé en tant que $s_2 = \delta_{int}(s_1)$. La fonction δ_{int} ($\delta_{int} : S \rightarrow S$) s'appelle la *fonction de transition interne*.

Quand l'état passe de s_1 à s_2 un événement de sortie est produit avec la valeur $y_1 = \lambda(s_1)$. La fonction λ ($\lambda : S \rightarrow Y$) s'appelle la *fonction de sortie*. Les fonctions ta, δ_{int} et λ définissent le comportement autonome d'un modèle de DEVS. Quand un événement d'entrée arrive l'état change instantanément. La nouvelle valeur d'état dépend non seulement de la

valeur d'événement d'entrée mais également de la valeur précédente d'état et du temps écoulé depuis la dernière transition. Si le système va à l'état s_3 au temps t_3 et alors un événement d'entrée arrive au temps t_3+e avec la valeur x_1, le nouvel état est calculé en tant que $s_4 = \delta_{ext}(s_3, e, x_1)$ (notant que $ta(s_3) > e$). Dans ce cas-ci, on dit que le système fait une *transition externe*. La fonction δ_{ext} (δ_{ext}:$S\times \mathbb{R}^+ \times X \rightarrow S$) s'appelle la fonction de *transition* externe.
Aucun événement de sortie n'est produit pendant une transition externe.

Un modèle DEVS couplé CM peut se décrire sous la forme :
$CM = < X,Y,D, \{M_d \in D\}, EIC, EOC, IC >$
- *X* : Ensemble de ports et de valeurs d'entrée pour la réception d'activations externes,
- *Y* : Ensemble de ports et de valeurs de sortie pour l'émission d'événements de sortie,
- *D* : L'ensemble des composants qui lui sont attachés (couplés ou atomiques),
- M_d : Le modèle DEVS pour chaque $d \in D$,
- *EIC* : L'ensemble des liens d'entrée qui connectent les entrées du modèle couplé à une ou plusieurs des entrées des composants qui lui sont attachés.
- *EOC* : L'ensemble des liens de sortie qui connectent les sorties d'un ou plusieurs des composants qui lui sont attachés, aux sorties du modèle.
- *IC* : L'ensemble des liens internes qui connectent les composants qui sont attachés, à *CM* entre eux.

Dans un modèle couplé, un port de sortie d'un modèle $M_d \in D$ peut être connecté à l'entrée d'un autre $M_d \in D$ mais pas directement à lui-même. Les modèles couplés comme atomiques sont autonomes et peuvent être stockés séparément. La structure interne d'un modèle couplé peut être cachée pour créer des composants de plus haut niveau.

Le formalisme DEVS présente une séparation explicite entre la modélisation et la simulation : un modèle DEVS est directement simulable dans le contexte d'un cadre d'expérimentation donné.

La simulation de modèles DEVS est dirigée par les événements qui activent les transitions d'états des modèles. La figure 4.12 présente le fonctionnement d'un modèle atomique en rapport à ces activations.

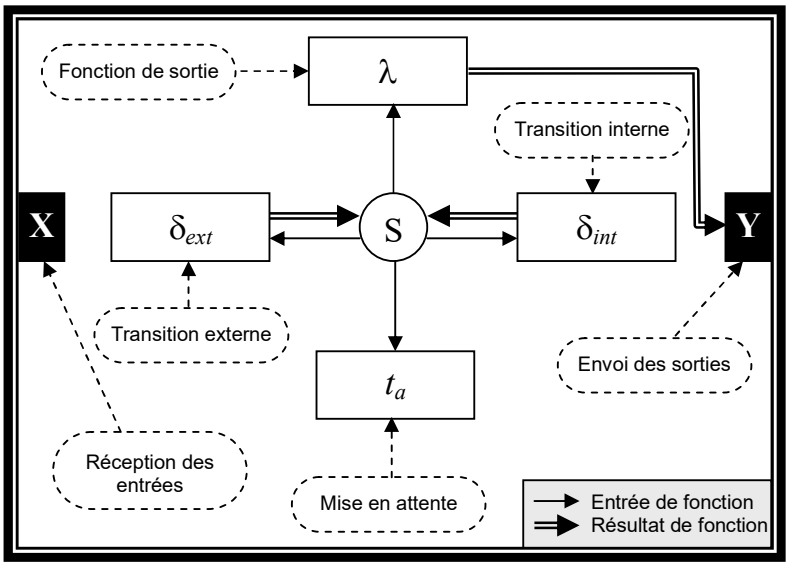

Figure 4.12 : Fonctionnement d'un modèle atomique

DEVS est très proche de la théorie générale des systèmes et se présente comme formalisme unificateur pour la spécification des systèmes. Il présente aussi l'avantage d'être *fermé sous composition*. Un formalisme est défini comme *fermé sous composition* si n'importe

quelle composition obtenue par couplage de composants spécifiés par le formalisme est elle-même spécifiée par le formalisme. Les équations différentielles et les machines séquentielles sont connues pour être *fermées sous composition*. La signification d'une telle propriété est qu'elle facilite la construction hiérarchique des modèles par l'application récursive des procédures de couplage [Zeigler02][Zeigler04].

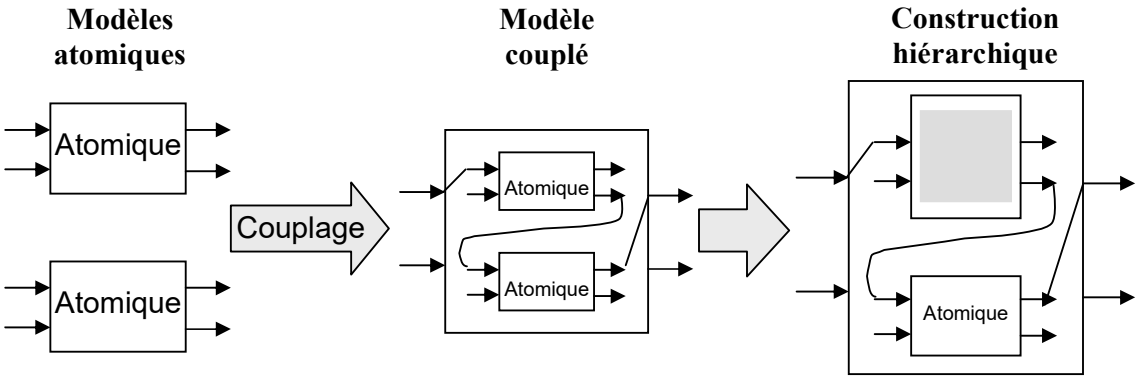

Figure 4.13 : Composition d'un modèle couplé

Le couplage entre différents composants est indiqué en énumérant les connexions pour les décrire. Une *connexion interne* implique un port d'entrée et un port de sortie correspondant à différents modèles. Dans le contexte du couplage hiérarchique, il y a également des connexions des ports de sortie des sous-systèmes aux ports de sortie du réseau qui s'appellent *connexions de sortie externes* et des connexions des ports d'entrée du réseau aux ports d'entrée des sous-systèmes qui s'appellent *connexions d'entrée externes*. La figure 4.14 montre un modèle DEVS couplé *N* ce qui est le résultat de couplage des modèles *Ma* et *Mb*. Là, le port de sortie 2 du *Ma* est relié au port d'entrée 1 du *Mb*. Cette connexion peut être représenté par [(*Ma*,2),(*Mb*,1)]. Les autres connexions sont [(*Mb*,1),(*Ma*,1)],[(*N*,1),(*Ma*,1)],[(*Mb*,1),(*N*,2)], etc. Selon la propriété de fermeture, le modèle *N* peut être également employé en temps que DEVS atomique et il peut être couplés à d'autres modèles atomiques ou couplés.

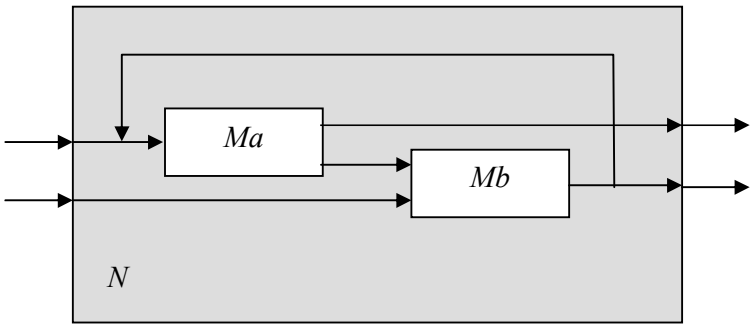

Figure 4.14 : Un modèle DEVS couplé

La théorie de DEVS utilise une structure formelle pour représenter les modèles couplés de DEVS avec des ports. La structure inclut les sous-systèmes, les connexions, les entrées et les sorties du réseau et la fonction *tie-breaking* pour gérer la présence des évènements simultanés.

4.3. Architecture de l'environnement

La conception d'un système est faite après la spécification des besoins, et la précision des objectifs attendus de ce dernier.

Alors à partir de nos besoins et objectifs nous allons définir les fonctions de l'environnement. Notre environnement est conçu pour permettre aux non programmeurs de réaliser des projets de simulation passant par toutes ses étapes et pour permettre aussi la collaboration entre plusieurs participants. Dans la phase de construction des modèles les utilisateurs n'auront pas besoin d'écrire des programmes, ceci est rendu possible grâce à la programmation visuelle qui sera intégré dans l'environnement. La construction d'un modèle se résumera à une série de "glisser déposer" et clique sur la souris et remplissage des champs vides. L'environnement doit aussi permettre de visualiser la simulation des modèles construits.

4.3.1. Architecture globale

Nous nous sommes basés sur le modèle DEVS classique [Zeigler95][Zeigler98] [Zeigler05] comme un cas d'étude. DEVS a eu une implémentation au niveau de ACIMS Arizona [Acims] en utilisant le langage Java(TM) , l'environnement est appelé DEVSJAVA [Acims][Zeigler05].

Ainsi, pour la conception de notre environnement nous nous sommes basé sur les concepts utilisés dans DEVSJAVA. Notre environnement se divise en quatre unités principales, Interface utilisateur, Unité de programmation visuelle, Unité de simulation et visualisation et Gestionnaire de collaboration.

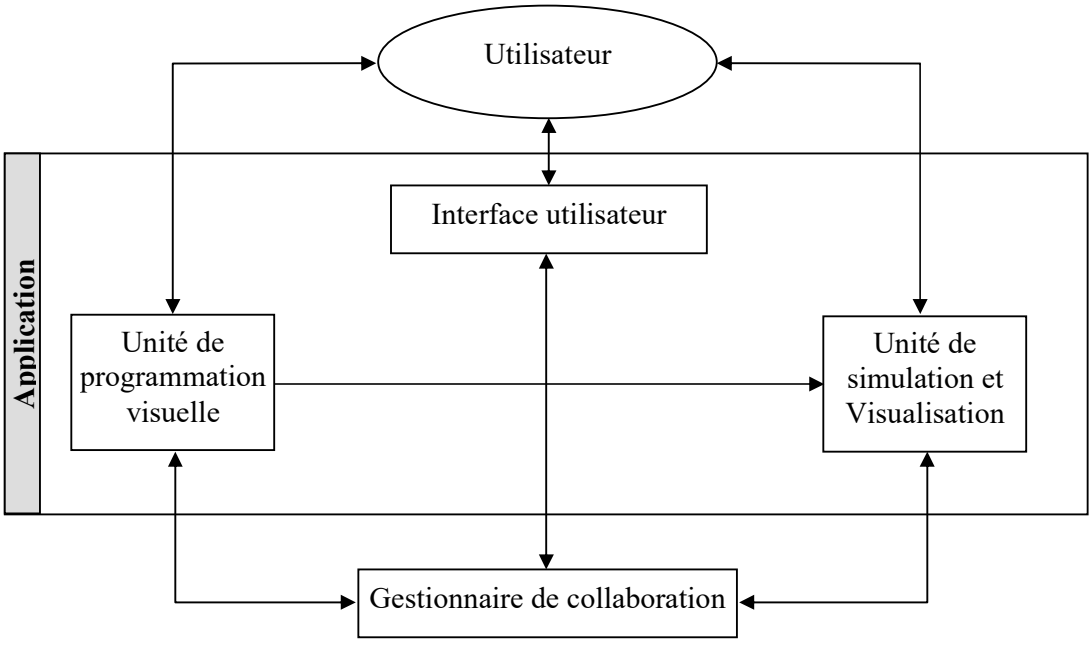

Figure 4.15 : Architecture globale de l'environnement

Le schéma dans la figure 4.15 montre les différents composants de l'environnement, ils sont chargés des fonctions principales qui répondent aux besoins décris précédemment.

Les utilisateurs peuvent travailler en mode collaboratif via une interface utilisateur graphique (GUI), cette interface permet aux utilisateurs de facilement utiliser des différentes fonctionnalités d'un *groupeware* (chat, tableau blanc, ……..).

Le deuxième composant c'est l'unité de programmation visuelle, elle permet la construction des modèles de simulation DEVS d'une manière graphique. Nous savons que les

approches classiques exigent un effort important dans la phase de programmation, par contre notre approche rend la construction des modèles aisée. A l'aide des outils disponibles dans l'éditeur l'utilisateur peut dessiner le modèle sous forme d'un ensemble de blocs connectés les uns aux autres. A la fin le modèle construit sera transformé automatiquement en un programme qui sera envoyé à l'unité de simulation et de visualisation.

Le troisième composant est l'unité de simulation et visualisation des modèles DEVS, elle permet l'exécution des modèles construit par l'unité de programmation visuelle. Le principe est l'association à chaque sous modèle atomique un processus simulateur qui l'exécute, le simulateur envoie des évènements à l'entrée de sous modèle et reçoit des évènements de sortie, pour la simulation de tout le modèle, on doit construire un arbre de simulation dont les nœuds sont des simulateurs et des coordinateurs, à chaque sous-modèle atomique un simulateur est associé et à chaque sous-modèle couplé un coordinateur est associé, la racine de l'arbre est un coordinateur principal. Les résultats de la simulation sont retournés sous forme visuelle (visualisés). Nous allons donner plus de détails sur les composants et les fonctions de cette unité à la suite.

Le dernier composant est le gestionnaire de collaboration, qui est le point de liaison entre les différentes téléapplications. Dans le cas mono- utilisateur, un seul utilisateur travaille (une seule entrée), via l'interface. Il fait appelle au composant de programmation visuelle, construit son modèle sous format graphique qui sera exécuté par l'unité de simulation et visualisation et il peut voir les résultats de la simulation. Le rôle de gestionnaire de collaboration et de gérer les tâches précédentes en mode collaboratif c-à-d plusieurs utilisateurs partagent les mêmes composants et construisent simultanément le même modèle et voient en même temps l'exécution et la visualisation de la simulation.

L'environnement que nous avons conçu est un système qui sera exécuté au niveau d'une machine terminale qui appartient à un réseau, alors pour avoir un collecticiel on doit exécuter notre système sur différentes machines, sur une de ces machines le serveur est exécuté. Les participants sont les utilisateurs entrant dans la session du collecticiel, ces derniers peuvent communiquer et travailler en collaboration à l'aide de gestionnaire de collaboration associé à chacune des applications. Le serveur qui est exécuté sur une seule machine crée la session de collaboration et gère l'inscription de chaque client (participant) dans la session. Le serveur contrôle la communication entre les différents participants.

La figure suivante montre l'architecture générale du collecticiel son fonctionnement :

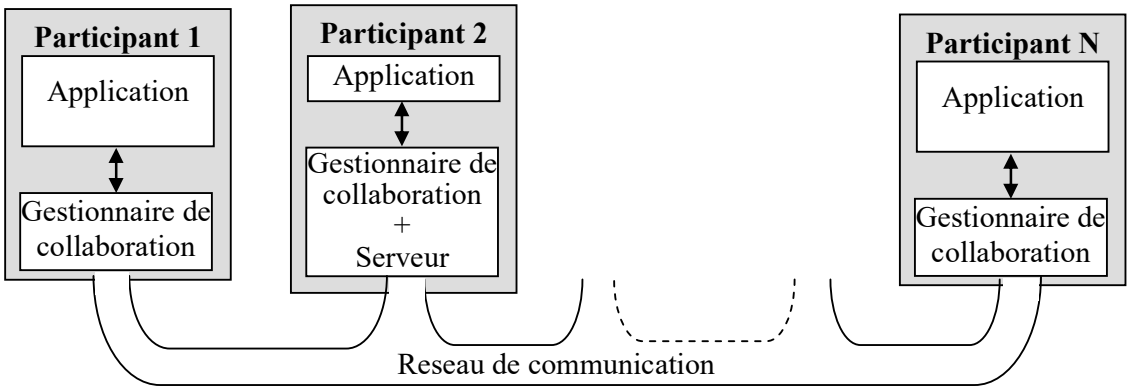

Figure 4.16 : Le collecticiel dans un réseau de communication

Dans ce qui suit nous allons détailler chaque composant de l'environnement (architecture et fonctionnement) afin de bien éclaircir notre approche.

4.3.2. Architecture détaillée de l'environnement

4.3.2.1. Interface utilisateur

Ce composant est une interface qui permet à l'utilisateur de faire l'appel aux autres composants, l'interface utilisateur fournit aussi des outils de communication permettant la télécollaboration :

- Un tableau blanc.
- Une fenêtre de chat.
- Un outil de conversation vocale.
- Un éditeur de texte.

Chacun de ces outils fonctionne en collaboration de la même façon que dans les autres applications de collecticiel. La figure ci-dessous montre les différentes parties de l'interface utilisateur :

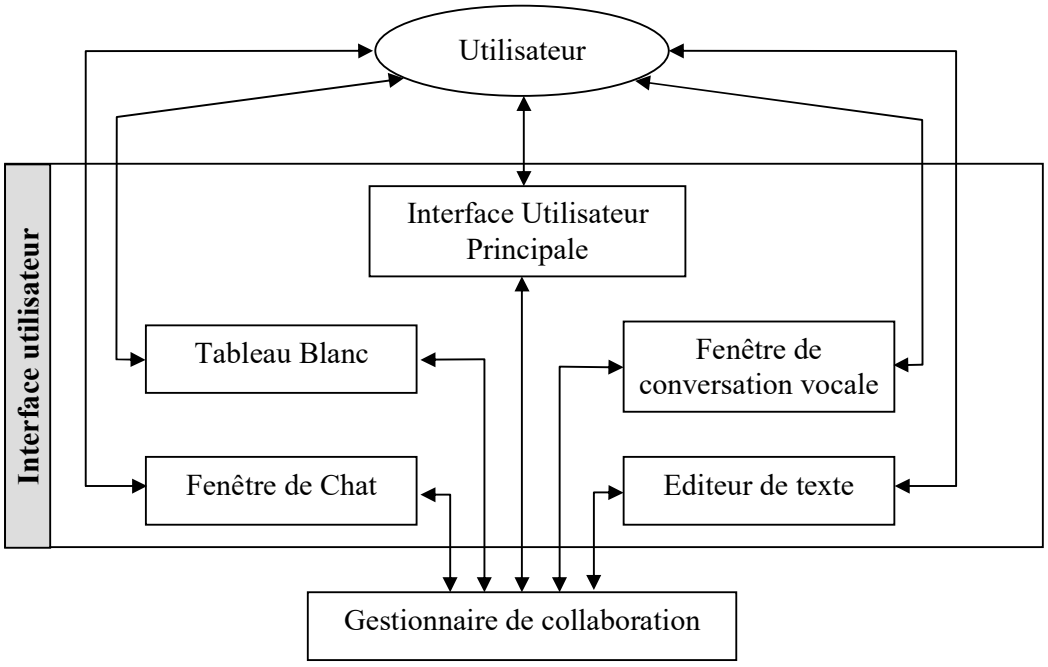

Figure 4.17 : L'interface utilisateur

4.3.2.1.1. Interface utilisateur principale

A l'aide de l'interface utilisateur principale l'utilisateur peut :

- *Exécuter le serveur de collecticiel :* puisque le serveur ne peut s'exécuter que sur une seule machine alors un des utilisateurs doit exclusivement exécuter le serveur à partir de sa propre interface utilisateur principale.
- *S'inscrire dans la session de collaboration* : pour s'inscrire, un utilisateur doit se connecter au serveur, enregistrer son nom. Le serveur garde le nom et l'adresse de ce nouveau utilisateur.
- *Faire appel aux autres composants* : à partir de cette interface l'utilisateur peut appeler les autres composants (unité de la programmation visuelle, unité de simulation et de visualisation, tableau blanc, fenêtre de chat, fenêtre de conversation vocale et éditeur de texte),

la remarque importante ici est qu'il y a une indépendance entre ces composants, ceci implique qu'un utilisateur n'est pas obligé de mettre au point tous les composants, l'utilisateur peut ne fait appel qu'aux composants nécessaires à effectuer sa tâche. Quand un utilisateur fait l'appel,

le composant aura un enregistrement dans la session de collaboration. Il est préférable que - avant de commencer le travail en collaboration - tous les utilisateurs doivent s'inscrire dans la session et faire appel aux composants exigés par leurs taches respectives.

Les fonctions principales de cette interface sont :

- *appelServeur()* : faire appel au serveur.
- *inscrireSession()* : spécifier les informations (nom de l'utilisateur,mot de passe), puis enregistrer l'utilisateur à la session de travail.
- *appelTBlanc()* : faire appel au Tableau Blanc.
- *appelChat()* : faire appel à la fenêtre de Chat
- *appelFCVocale()* : faire appel à la fenêtre de conversation vocale.
- *appelETexte()* : faire appel à l'éditeur de texte partagé.
- *appelEPVisuelle()* : faire appel à l'éditeur de la programmation visuelle.
- *appelSimVis()* : faire appel à l'unité de simulation et visualisation.

4.3.2.1.2. Tableau blanc

Le tableau blanc est un outil qui est intégré dans la plupart des collecticiels (universels ou spécifiques) à cause de son intérêt dans la conception des projets (voir le chapitre précédent).

Le tableau blanc est un espace de dessin qui permet à un utilisateur d'exploiter une palette de formes et de couleurs et dessiner sur le tableau blanc. Quand un utilisateur dessine dans son propre tableau, les tableaux des autres utilisateurs sont mis à jour, c'est-à-dire le même dessin apparaît dans tous les tableaux.

L'idée de tableau blanc est issue du principe de tableau blanc utilisé dans les conférences ou lors de conception des projets, où les membres de groupe partagent le tableau blanc pour que chacun puisse expliquer ses idées dans le même tableau.

Dans notre environnement les membres de groupe doivent discuter et utiliser beaucoup de brouillon pendant la phase de conception des modèles de simulation, le tableau blanc est alors intégré dans notre environnement.

4.3.2.1.3. Fenêtre de chat

Le chat est un outil très important dans les collecticiels, il est utilisé dans la plupart des collecticiels. L'importance du chat est évidente dans notre cas. Supposons par exemple qu'un membre de groupe veut envoyer une petite information (mot de passe, adresse, mot d'encouragement, ……) à d'autres membres, il aura alors besoin d'utiliser le chat.

Pour cela nous avons intégré une fenêtre de chat dans notre environnement. A l'aide de cette fenêtre chacun des utilisateurs peut envoyer quelques phrases à un autre utilisateur et peut aussi les envoyer à tous les utilisateurs. Au contraire de la messagerie électronique, les messages envoyés dans le chat arrivent en mode synchrone.

La fenêtre de chat contient deux zones de texte, dans la première l'utilisateur peut composer des messages et il reçoit dans l'autre zone les messages arrivés, à chaque message est associé le nom de l'émetteur. Avant d'utiliser cette fenêtre l'utilisateur doit entrer son nom et l'enregistrer. Avant d'envoyer une message il doit spécifier le récepteur (un utilisateur ou tous les utilisateurs).

4.3.2.1.3. Fenêtre de conversation vocale

La conversation vocale permet aux utilisateurs de communiquer et de bien exposer leurs différentes idées, et aussi pour que le chef de projet puisse facilement diriger son groupe de travail à distance. Nous avons estimé donc qu'il est nécessaire d'intégrer une fenêtre de conversation vocale dans notre environnement.

Le principe de la fenêtre de conversation vocale est analogue au principe de téléphone normal, mais dans la fenêtre de conversation l'utilisateur peut parler avec un utilisateur et il le peut avec tous les utilisateurs. L'utilisateur s'il veut faire une conversation doit entrer son nom et spécifier le nom de l'appelant puis il fait l'appel.

4.3.2.1.4. Editeur de texte partagé

Les membres d'un groupe de développement de projets de simulation ont besoin de rédiger ensemble des rapports sur les étapes de projet. Pour cette raison nous proposons l'intégration d'un éditeur de texte partagé à l'environnement de programmation visuelle.

Le principe de l'éditeur de texte que nous avons intégré est différent des autres composants, l'utilisation de l'éditeur de texte ne sera autorisée que pour un seul utilisateur à la fois, alors il est verrouillé pour les autres utilisateurs. Si l'utilisateur courant de l'éditeur modifie le contenu du texte, tous les autres éditeurs seront mis à jour simultanément.

4.3.2.2. Unité de la programmation visuelle des modèles

Un modèle de simulation DEVS est un programme écrit en un langage évolué tel que (Java, C++, …), ce programme sera exécuté lors de la simulation. Alors pour qu'un programmeur construit un modèle il doit écrire de nombreuses lignes de code.

Comme nous avons vu précédemment, n'importe quel modèle du formalisme DEVS a une structure hiérarchique, c'est-à-dire qu'un modèle DEVS est décomposable en sous-modèles qui peuvent être aussi décomposables, alors nous trouvons que la *programmation orientée composants* est le paradigme le plus adapté à l'implémentation [Zeigler05].

Un composant est, tout simplement, une partie d'un programme écrit dans un langage de programmation cette partie et est indépendante. Le composant peut être réutilisable par d'autres programmes. Un modèle est alors un ensemble de composants réutilisables regroupés afin de composer ce modèle. La figure 4.18 montre la structure logicielle d'un exemple de modèle DEVS.

L'aspect visuel de la programmation orientée composants est un bénéfice important au niveau de l'unité de la programmation visuelle. L'idée générale de l'éditeur visuel des modèles est : une surface de construction plus une palette contenant des modèles vides, atomiques et couplés. Afin de construire un modèle l'utilisateur doit construire les sous-modèles.

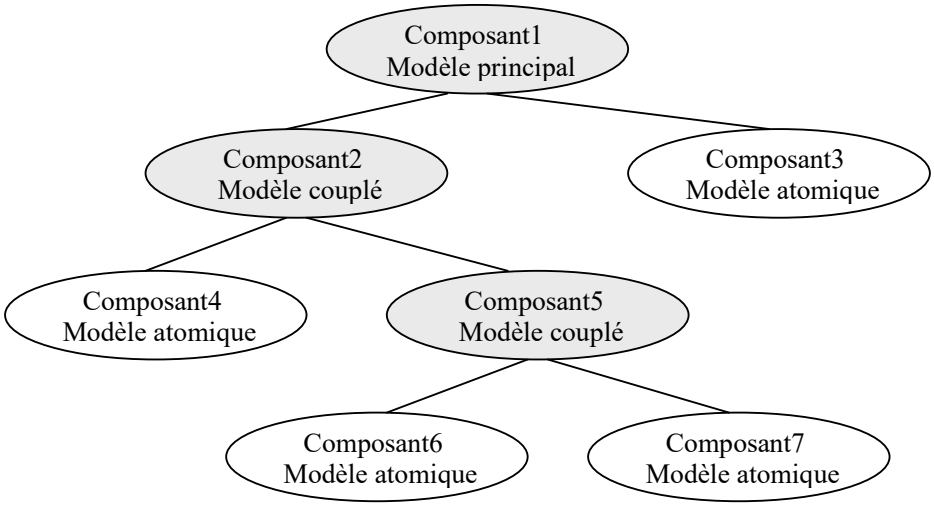

Figure 4.18 : Structure logicielle d'un modèle DEVS

La première étape est la spécification des entrées et des sorties du modèle L'étape suivante est la construction des sous-modèles, il y a deux types de sous-modèles, les modèles DEVS atomiques et les modèles DEVS couplés. La construction d'un sous-modèle atomique exige à l'utilisateur de définir le nom de sous-modèle, les entrées, les sorties et les propriétés (l'ensemble des états S). Pour construire un sous-modèle couplé l'utilisateur doit suivre les mèmes étapes de la construction du modèle principal, alors la programmation visuelle d'un modèle DEVS est une tâche récursive. Le sous-modèle créé (atomique/couplé) est apparu sur la surface, l'utilisateur peut le déplacer à n'importe quelle position de la surface.
Une fois l'utilisateur a défini et placé tous les sous-modèles, il va ensuite faire connecter les portes des sous modèles, les unes avec les autres comme il est montré dans la figure suivante.

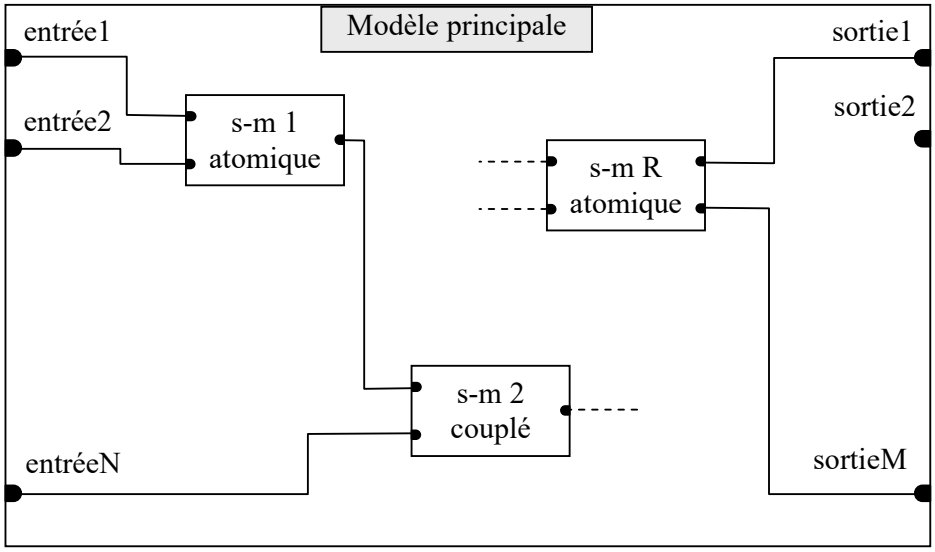

Figure 4.19 : Représentation visuelle d'un modèle DEVS

Après que l'utilisateur définisse le modèle principal l'unité de programmation visuelle génère automatiquement les fichiers de code source, un fichier source pour chaque sous-modèle et pour le modèle principal, puis le modèle soit sera compilé afin de préparer a l'exécution, soit il sera ajouté à une bibliothèque des modèles. Les modèles stockés dans la bibliothèque seront réutilisés lors de la construction d'autres modèles.

Après la spécification de nos besoins et après la révision de quelques travaux reliés à ce sujet nous avons proposé les différents composants de l'unité de la programmation visuelle et leurs fonctionnements. La figure 4.20 illustre les différents composants de l'unité de la programmation visuelle.

4.3.2.2.1. Les composants

Editeur graphique de la modélisation

L'éditeur graphique de la modélisation est une interface qui permet à l'utilisateur de concevoir d'une manière visuelle un modèle de simulation DEVS. Il se compose d'un panneau de dessin et d'une palette contenant les outils de conception. L'éditeur graphique fournit un ensemble de fonctions nécessaires, les fonctions principales sont :

- *ajouterEntree(),* pour définir une nouvelle entrée au modèle/sous-modèle.
- *ajouterSortie(),*pour définir une nouvelle sortie au modèle/sous-modèle.
- *ajouterAtomique(),* créer un modèle atomique dans un modèle couplé.
- *ajouterCouple(),*créer un modèle couplé à l'intérieur d'un modèle couplé.

- *ajouterPropriete(),*définir un nouvel état de l'ensemble *S*.
- *ajouterConnection(),* faire une connexion entre une source et une destination.
- *editerModele(),* permettre l'ajout des lignes de code au fichier source.

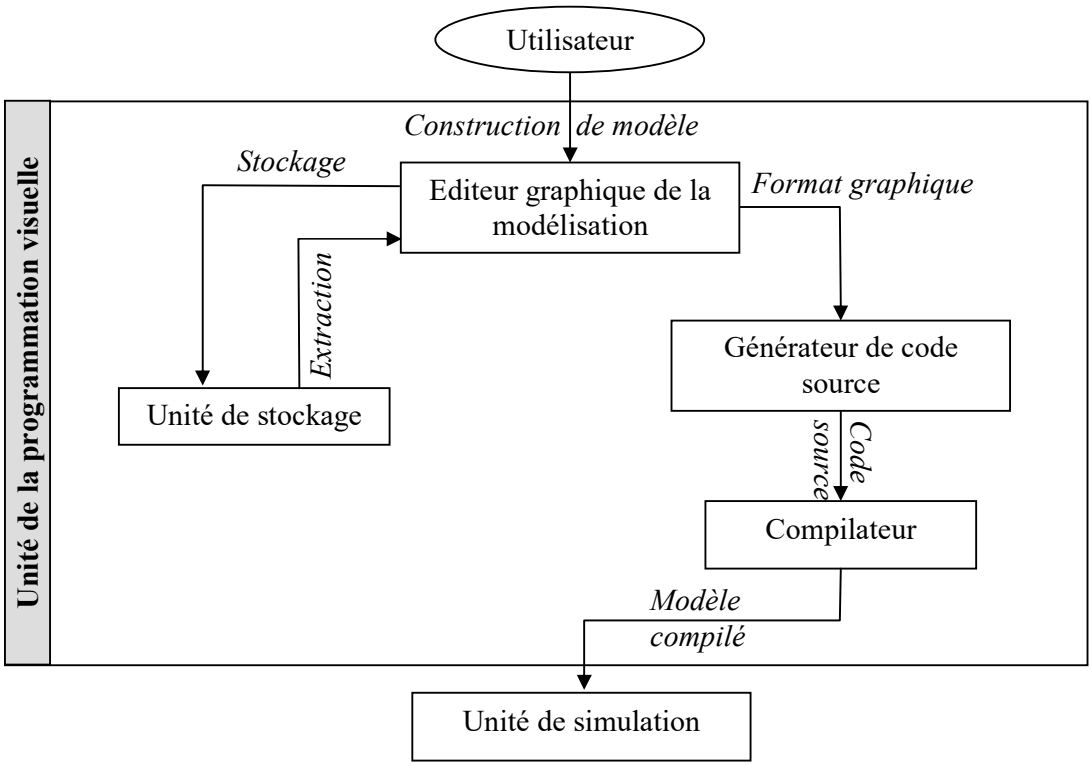

Figure 4.20 : Les composants de l'unité de la programmation visuelle

Unité de stockage

L'unité de stockage est le composant qui gère l'opération de stockage et d'extraction des modèles de simulation. Tous les modèles construits seront stockés dans une bibliothèque, l'opération de stockage doit conserver toutes les informations graphiques et comportementales du modèle. Une fois l'utilisateur aura besoin d'utiliser un de ces modèles l'unité de stockage l'extrait, l'utilisateur peut à la suite manipuler le modèle chargé dans l'éditeur graphique comme un sous-modèle.

L'unité de stockage fournit de fonctions principales :

- *stockerModele()* : stocker le modèle dans une bibliothèque, sous une format qui sauvegarde toutes les informations graphiques.
- *extraireModele()* : charger le modèle stocké dans l'éditeur graphique.

Générateur de code source

Comme nous avons dit précédemment, un modèle est à la fin un code source écrit dans un ou plusieurs fichiers. Le générateur de code source joue le rôle de programmeur. Après que l'utilisateur construit son modèle à l'aide de l'éditeur graphique, le générateur de code source transforme la représentation visuelle du modèle en un code source, le code source est écrit par un langage évolué.

Compilateur

L'unité de la programmation visuelle doit contenir un compilateur de langage évolué, ce compilateur sera intégré dans l'unité de la programmation visuelle, après que le modèle soit compilé il sera alors prêt à l'exécution, le rôle de l'unité de la programmation visuelle des modèles est terminé.

4.3.2.2.2. Fonctionnement

A l'aide de l'éditeur graphique de la modélisation l'utilisateur construit son modèle de simulation aisément, ce modèle peut être atomique et peut être couplé, il crée alors un ensemble de composants qui représentent les sous-modèles (atomiques/couplés), ces composants ont des propriétés visuelles (forme, couleur, taille, position,….), et des propriétés non visuelles (informations internes). Chaque composant est relié avec d'autres composants via un ensemble de portes (d'entrée et de sortie), les portes sont conçues aussi comme des composants et ils portent des informations spécifiques.

Le travail commence, l'utilisateur développe son modèle à l'aide de l'éditeur graphique. Une fois la programmation visuelle se termine, l'utilisateur aura la main soit d'enregistrer le modèle à l'aide de l'unité de stockage, soit de générer le code source du modèle à l'aide de générateur de code source. Le code source ne doit porter aucune information visuelle sur le modèle, à cause de ça on a besoin de définir une forme de stockage qui peut garder la structuration du modèle (relation de dépendance entre les sous modèles) et toutes les informations visuelles. Après la génération de code source (programme), le programme sera compilé afin de générer le byte code, puis le code généré sera près et passe à la phase de l'exécution et la visualisation.

Pendant la première phase, la programmation visuelle, l'utilisateur peut ajouter à son modèle des modèles stockés dans la bibliothèque, ils sont considérés comme des sous-modèles couplés.

En mode collaboratif les utilisateurs dans une session peuvent travailler en collaboration afin de construire leur modèle, quand un utilisateur édite le modèle, les éditeurs des autres utilisateurs sont mis à jours (des modifications sont apparus dans tous les éditeurs). A la fois un seul utilisateur a le droit d'utiliser l'éditeur graphique, alors la programmation visuelle est aussi un travail exclusif, c'est-à-dire si l'utilisateur en cour utilise son propre éditeur aucun autre ne peut utiliser son éditeur que si l'utilisateur en cour lui donne l'accord.

4.3.2.3. Unité de simulation et visualisation

Avec l'unité de simulation et visualisation les modèles qui sont construits à l'aide de l'unité de la programmation visuelle seront exécutés en visualisant les résultats.

Cette unité fournit deux fonctions, la simulation et la visualisation. La simulation est l'exécution d'un modèle par le calcul de l'avancement du temps, l'injection des évènements aux entrées et la prise des évènements des sorties. La visualisation est l'affichage de tous les changements des contenus de modèle lors de la simulation et l'affichage des résultats obtenus en une façon visuelle.

4.3.2.3.1. La simulation

Afin d'interpréter le modèle, l'ordinateur a besoin d'un simulateur abstrait. Ce concept a été défini par Zeigler [Zeigler05].

Le simulateur abstrait est un dispositif conceptuel capable d'interpréter la dynamique des modèles DEVS. Il prend un modèle à évènement discret et exécute l'expérience de simulation. Ainsi le modèle peut être aisément transféré dans un programme de simulation

exécutable.

Le simulateur abstrait pour DEVS a une structure hiérarchique qui reflète la structure du modèle (figure 4.21). Il se compose de deux types d'objet - les *simulateurs* et les *coordinateurs*. À chaque modèle atomique nous associons un simulateur, et à chaque modèle couplé un coordinateur est associé. Pour lancer les cycles de simulation un coordinateur racine (root) est employé. Ils sont attachés au coordinateur global du réseau.

Tandis que la simulation a lieu, des messages sont passés entre les simulateurs et les coordonnateurs. De ce fait, il n'est pas important, si un message est passé vers un coordonnateur ou un simulateur, tous les deux sont capables de traiter les mêmes genres de messages. Comme il est montré dans la figure suivante, le simulateur abstrait travail avec le passage de messages :

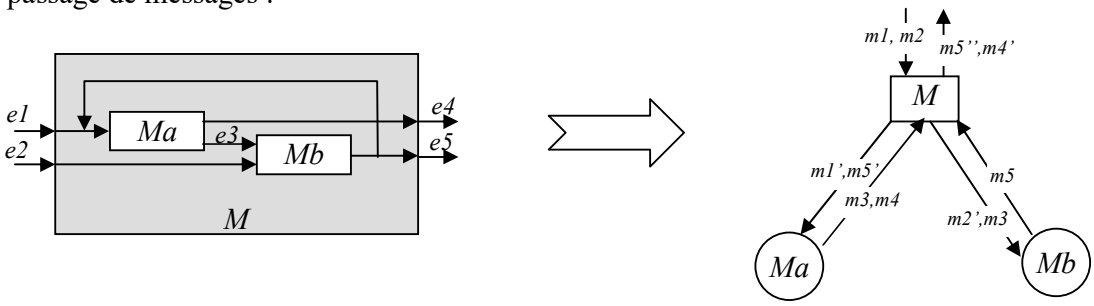

Figure 4.21 : L'envoie de messages dans un simulateur abstrait d'un modèle couplé

Les évènements qui arrivent à simulateur/coordinateur ou qui se produisent d'un simulateur/coordinateur sont portés dans des messages. Un message contient plusieurs informations (évènement *ei, destination, numéro de port, type de port*). Par exemple *m1'(e1,Ma,1,in), m4(e4,M,1,out),…*Dans le cas de *m3* qui se produit de *Ma* et arrive à *Mb*, il doit passer au coordinateur *M.*

Le coordinateur envoie des messages à ses fils par conséquent ils exécutent les fonctions de transition. Quand un simulateur exécute une transition, il calcule son prochain état et -quand la transition est interne- il envoie la valeur de la sortie à son coordinateur parent. Dans tous les cas, l'état de simulateur coïncidera avec l'état de son modèle atomique DEVS associé.

Quand un coordinateur exécute une transition, il envoie des messages à quelques uns de ses fils ainsi ils exécutent leurs fonctions de transition correspondantes. Quand un évènement de sortie s'est produis d'un fils, le coordinateur envoie un message à son coordinateur parent portant la valeur de la sortie.

Chaque simulateur ou coordinateur a un variable local *tn* qui indique le temps de l'occurrence de sa prochaine transition interne. Dans les simulateurs ce variable en utilisant la fonction *ta* du modèle atomique correspondant. Dans les coordinateurs, il est calculé comme le *tn* minimum de ses fils. Donc, le *tn* du coordinateur racine est le temps de l'occurrence du prochain évènement dans le système. Alors le coordinateur racine regard seulement à ce temps, avance le temps global *t* à cette valeur et alors il envoie un message à son fils ainsi il exécute la prochaine transition (il répète ce cycle jusqu'à la fin de la simulation).

Une des propriétés les plus intéressantes est l'indépendance entre les différents simulateurs associés aux différents modèles atomiques. A cette manière, quand un modèle atomique a son temps d'avancement met à une grande valeur (ou infini), il n'affecte pas quoi que ce soit au reste des modèles et la simulation ne fait aucun calcul avec.

4.3.2.3.2. La visualisation

Le simulateur d'un modèle est une hiérarchie des simulateurs et coordinateurs, l'exécution de la simulation effectuée par le calcul de temps d'avancement et le passage des messages entre les simulateurs et les coordinateurs associés aux modèles atomiques et couplés.

Les éléments importants qu'un utilisateur doit suivre sont : *l'avancement de temps global, le temps restant pour avoir la prochaine transition interne, le changement d'état de chaque modèle atomique et la production des évènements*. La visualisation se base alors sur ces quatre éléments. Lors de l'exécution de la simulation le visualisateur de la simulation affiche le comportement de la simulation sur le panneau qui contient le modèle construit par l'utilisateur.

Pendant la simulation le visualisateur affiche l'avancement du temps, l'état de chaque sous-modèle atomique et le temps de l'occurrence de la prochaine transition interne *tn* sur les blocks qui représentent les sous-modèles atomiques, si une transition interne est effectuée le visualisateur affiche le nouvel état, quand un message est envoyé il affiche une petite boite qui montre le contenu du message, cette boite démarre de la sortie du sous-modèle source, se déplace sur la ligne de l'interconnexion et arrive à la fin au sous-modèle destinataire.

4.3.2.3.3. Architecture de l'unité de simulation et visualisation

On trouve dans cette unité une indépendance entre la simulation et la visualisation. Pendant la simulation le comportement du modèle simulé est visualisé. On peut diviser cette unité en deux couches, couche de simulation et couche de visualisation selon la figure suivante :

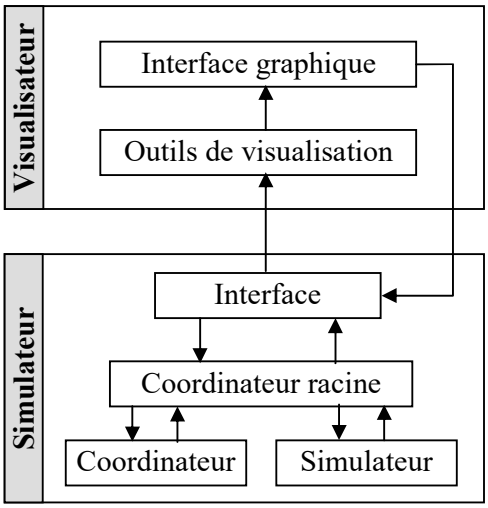

Figure 4.22 : Unité de simulation et visualisation

L'unité de simulation et visualisation est divisée en deux couches, couche de haut niveau (visualisateur) et couche de bas niveau (simulateur). Dans le visualisateur il y a une interface graphique utilisateur. Cette interface qui contient un panneau de visualisation permet à l'utilisateur de charger le modèle dans le panneau de la visualisation, d'injecter les évènements dans les entrées du modèle et de lancer la simulation. Le visualisateur à des outils qui permettent de traduire les effets de la simulation et les visualiser sur le panneau (production d'évènements, incrémentation du temps, changement d'état).

Pour le simulateur, une structure de simulateurs et coordinateurs sera associée au modèle. Un simulateur est généré pour chaque sous-modèle atomique et un coordinateur pour chaque sous-modèle couplé au début de la simulation. Le simulateur contient une interface,

cette interface traduit les commandes reçues de l'interface graphique surtout dans l'exécution pas à pas de la simulation et envoie les effets de la simulation aux outils de visualisation pour qu'elles soit affichés sur le panneau.

Le coordinateur racine est celui qui gère la simulation du modèle. Il lance pour chaque modèle couplé un coordinateur et pour chaque modèle atomique un simulateur, puis il met à jour son interface avec le visualisateur dans chaque modification.

4.3.2.3.4. Fonctionnement en mode collaboratif

En mode collaboratif l'unité de simulation et visualisation suit presque le même principe des autres composants de l'environnement. Après la construction du modèle il sera chargé dans le panneau de visualisation. L'initialisation de la simulation peut être faite par n'importe quel utilisateur, par exemple si un utilisateur injecte les évènements dans les entrées du modèle, les évènements seront injectés sur le visualisateur de tous les utilisateurs. Si un autre utilisateur exécute la simulation, elle sera exécutée pour tous les utilisateurs, l'exécution de la simulation ne sera pas distribuée (chaque sous modèle est exécuté dans une machine), mais la simulation de modèle complet sera exécutée seulement dans la machine de ce dernier utilisateur, ce qui se passe est quant un effet est produit les visualisateurs de tous les utilisateurs seront mis à jours. Comme ça la même simulation est suivie par tous les membres de groupe en même temps.

4.3.2.4. Gestionnaire de collaboration

Le gestionnaire de collaboration est le composant qui dirige le travail en collaboration, il fournit des objets qui permettent la communication et le partage de données entre les différents utilisateurs. Alors c'est à l'aide de ce gestionnaire que les autre composants de l'environnement peuvent être employés en mode collaboratif.

4.3.2.4.1. Principe de fonctionnement

Comme nous les avons détaillé, tous les composants de l'environnement ayants une interface utilisateur graphique (GUI). Les GUIs sont basés sur les évènements, quand une action est performée une tâche sera effectuée (un sous-programme sera exécuté).

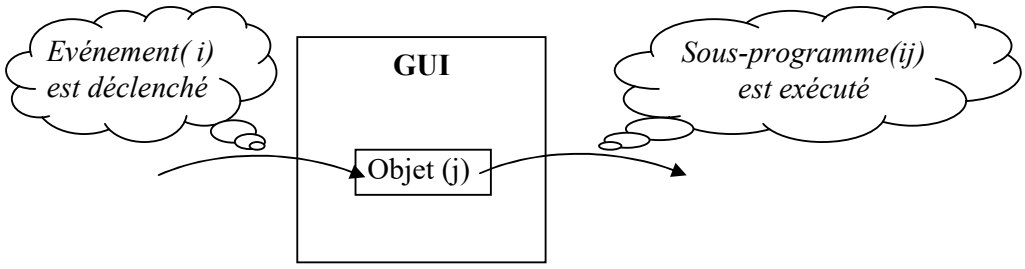

Figure 4.23 : Gestion des évènements

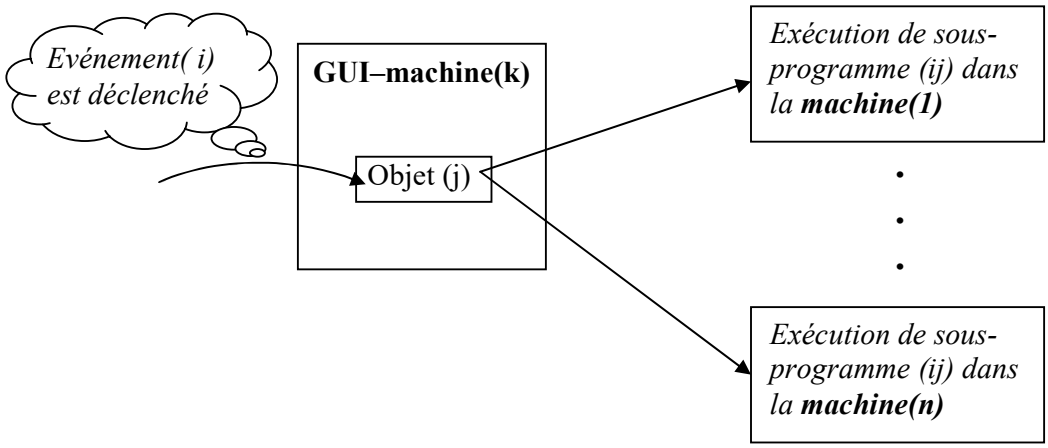

Figure 4.24 : Gestion des évènements en mode collaboratif

En mode collaboratif, on a plusieurs utilisateurs (clients) qui exécutent le même système dans des différentes machines. On doit avoir après l'inscription de tous les utilisateurs dans la session du travail collaboratif que si un des utilisateurs déclenche un *événement(i)* sur un *objet(j)* de son propre interface , les systèmes dans toutes les machines auront une exécution de *sous-programme(ij)*.

L'idée en globale pour avoir ce fonctionnement est simple. Si un événement est déclenché sur un objet de l'interface d'un client, le sous-programme correspondant sera exécuté au niveau de sa machine, au même temps le gestionnaire de collaboration reçoit un message de commande, alors il l'envoie à tous les autres clients. Ce message sera reçu par les gestionnaires de collaboration des autres clients, ce message contient les mêmes paramètres de l'exécution de sous-programme qui correspond à l'événement déclenché. La figure suivante illustre ce qui se passe quand un événement est déclenché.

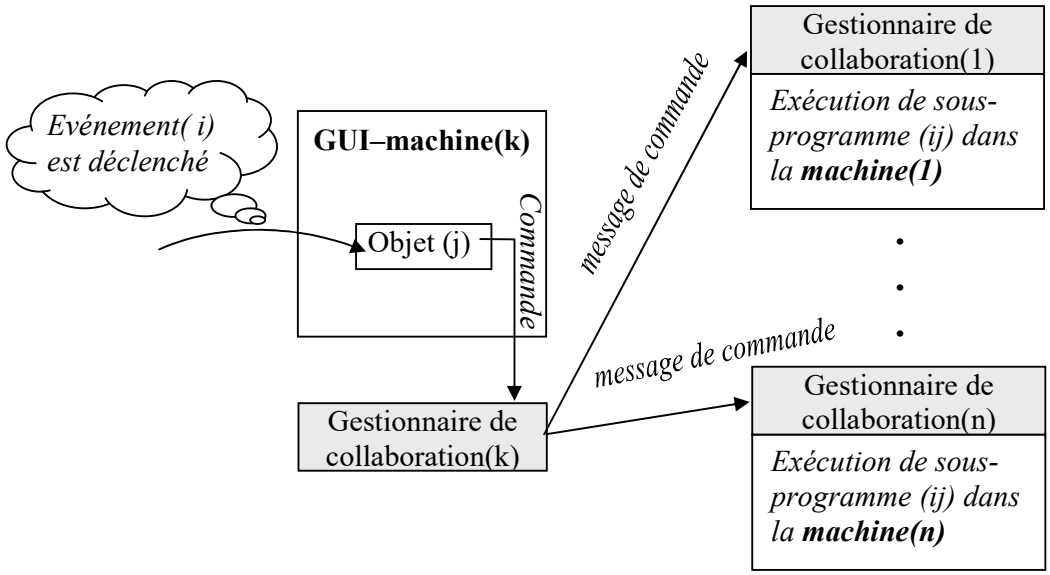

Figure 4.25 : Gestion des évènements par envoie de messages

4.3.2.4.2. Architecture du gestionnaire de collaboration

Pour pouvoir gérer la communication et le partage de données, le gestionnaire de collaboration doit fournir des outils capables et suffisants à recouvrir ce travail.

Nous avons proposé alors une architecture du gestionnaire de collaboration et aussi comment il doit fonctionner. Pour vérifier le travail collaboratif l'environnement doit avoir un serveur qui crée et dirige la session de collaboration. Chaque gestionnaire de collaboration dans les différentes machines à son propre serveur, mais un seul serveur qui sera exécuté.

Nous voulons aussi que les différents composants auront une indépendance l'ors de l'exécution, c'est à dire qu'un utilisateur peut employer chaque composant de sa propre application indépendamment des autres composants. Si un évènement arrive à un objet d'un composant, les actions ne seront exécutées qu'au niveau des composants similaires dans les différentes applications, par exemple si un utilisateur envoie un message de chat il ne sera reçu que par les fenêtres de chat des autres utilisateurs. Cela implique qu'on à besoin de concevoir une canal de communication pour chaque type de composant.

L'architecture du gestionnaire de collaboration est illustrée dans la figure 4.26. Le premier composant qui sera mis au point dans l'environnement est le serveur, il crée après son exécution la session de travail collaboratif. Cette session recouvre les applications de tout l'environnement, elle contient un ensemble d'objets (outils) qui assurent la communication et le partage de données (clients, canaux, consommateurs de messages).

Chaque composant d'une application à son ensemble d'objets propre qui lui assure la communication avec les composants similaires des autres applications de l'environnement.

- *Client :* on associe à chaque composant un client, il sera son interface dans la session de travail. Pour qu'un composant soit un membre de la session il devra créer son propre client afin de pouvoir s'inscrire à la session de travail, ce composant ne pourra aussi se connecter au serveur et se communiquer qu'à l'aide de son propre client.

- *Consommateur :* le consommateur est un objet important, il est responsable de recevoir les messages et de les traduire en commandes, informations et données. Le consommateur est une boite de réception des messages, il se trouve à l'intermédiaire entre le client et le canal de communication.

- *Canal :* le canal est l'objet qui est responsable de porter les messages. Si un client veut envoyer un message à d'autres clients il utilise le canal de communication, le client compose son message et lui associe son identificateur et les identificateurs des clients destinataires, puis il met le message dans le canal. Ce message ne sera à la suite reçu que par les consommateurs des clients destinataires spécifiés dans le message.

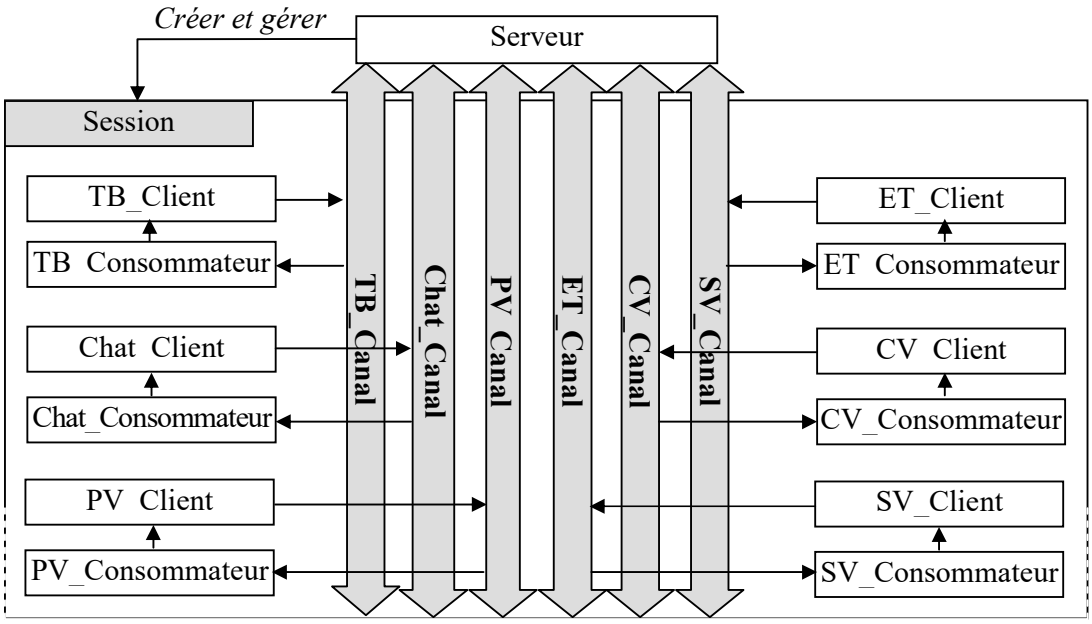

Figure 4.26 : Architecture du gestionnaire de collaboration

4.3.2.4.3. Fonctionnement

Nous avons présenté dans le paragraphe précédent les différents composants du gestionnaire de collaboration. Dans ce paragraphe nous allons montrer comment ces composants fonctionnent pour gérer un travail collaboratif.

Le serveur qui est le premier composant exécuté crée tous les objets nécessaires à la gestion de collaboration. Il commence par la création de la session, une fois la session est créée le serveur sera prêt à recevoir les inscriptions des différents clients. Comme nous avons proposé, les membres de la session ne sont pas les applications mais les composants des applications. Alors on peut avoir dans la même application des composants inscrits dans la session et des autres composants hors de la session.

Ensuite, le serveur crée un canal de transfère de données pour chaque type de composant, et chaque client doit joindre le canal pour pouvoir l'utiliser à la communication. Les clients associés aux composants similaires dans l'environnement doivent partager le même canal, un message envoyé ne pourra être reçu que par les clients joints au canal utilisé à la transmission.

Selon l'architecture du gestionnaire de collaboration présentée, la collaboration est atteinte par l'interaction de tous les objets. Le travail commence après l'inscription de tous les membres dans la session, c'est-à-dire les clients sont connectés au serveur. Une fois un utilisateur met à jours un des composants de son application, le client associé au composant envoie un message de commande, ce message doit contenir les informations suivantes :

Client source	**Clients destinataires**	**Ligne de commande**

Figure 4.27 : structure d'un message

- ***Client source*** : identificateur du client qui a envoyé le message.
- ***Clients destinataires*** : identificateurs des clients qui doivent recevoir le message.

- ***Ligne de commande*** : une chaîne de caractères qui contient des données, actions et les paramètres des actions, qui sont concaténés en une chaîne.

Le message envoyé sera transmit à travers le canal associé au client. Généralement le message doit être reçu par tous les clients qui joignent le canal (multicast), mais quelques fois le client source spécifie des clients destinataires, alors ils seront les seuls qui peuvent recevoir le message (broadcast).

Dès que le message soit envoyé, les clients raccordés avec le canal peuvent recevoir le message. A chaque client est associé un consommateur de messages, c'est le consommateur qui reçoit directement les messages. On pose la question « pourquoi un consommateur de messages ? ». La réponse est que les messages reçus ont besoin d'un traitement pour séparer les informations et traduire les commandes en actions, en plus le client peut recevoir plusieurs messages à la fois, donc le consommateur trie ces messages et les traduit message par message.

Prenant l'exemple de Tableau Blanc. Si l'utilisateur N° i trace une ligne de p1(x1,y1) vers p2(x2,y2) avec l'épaisseur 3 et la couleur rouge, alors cette ligne sera tracée dans le tableau blanc de l'utilisateur, et le message suivant est composé :

TB_Client i	**Tous**	**Tracer_Ligne x1 y1 x2 y2 3 rouge**

Figure 4.28 : message correspond au traçage d'une ligne

En fin la sortie de la session est effectuée par déconnection des clients du serveur, le client déconnecté sera disjoint de tous les objets du gestionnaire de collaboration.

4.4. Conclusion

Dans ce chapitre nous avons étudié quelques travaux qui traitent la programmation visuelle dans le domaine de la modélisation et la simulation. Nous avons utilisé les mêmes principes afin de concevoir l'unité de la programmation visuelle, et l'unité de la simulation et la visualisation. Les autres utilités du collecticiel (tableau blanc, chat, éditeur partagé, audio conférence) sont conçues en basant sur l'étude faite dans le chapitre précédent qui concerne les fonctionnalités d'un collecticiel de simulation.

Ce qui est ajoutée aux travaux précédents est la dimension de groupe, c'est-à-dire nous avons proposé une approche prend en compte la dimension de groupe dans la phase de la programmation visuelle des modèles de simulation. Cette approche traite alors la construction des modèles de simulation visuellement par un groupe en télécollaboration (les membres de groupe travaillent dans des emplacements géographiquement séparés).

Les utilitaires sont ajoutés afin d'étendre l'environnement et pour qu'il soit capable de traiter toutes les étapes d'un projet de simulation en mode collaboratif.

Dans le chapitre suivant nous allons détailler l'implémentation d'un prototype d'environnement de programmation visuelle des modèles de simulation de formalisme DEVS, prenons en compte la dimension de groupe. Le prototype implémenté est pour valider l'approche proposée et détaillée dans ce chapitre.

CHAPITRE 3

Modélisation, simulation et collaboration

La simulation est un domaine vaste et très intéressant à cause de ses avantages dans l'étude et l'analyse des systèmes réels. Plusieurs travaux de recherche sont menés afin de simplifier et augmenter la puissance des environnement de modélisation et simulation.

Les études sont arrivées à développer des environnements permettant aux experts participant à un projet de simulation de travailler en collaboration dans toutes ces étapes sans être dans la même situation (à distance souvent), ces environnements sont appelés « Environnements Collaboratifs de Simulation » et qui prennent en compte la dimension du groupe.

Dans ce chapitre nous allons présenter les fondements de la modélisation et la simulation et des notions sur la collaboration et les Environnements Collaboratifs de Simulation.

3.1. Modélisation et simulation

3.1.1. Modélisation

3.1.1.1. Système

Comme définition d'un système nous retenons celle donnée par l'AFCET (Agence Française de la cybernetique) [Belattar93] et qui s'énonce comme suit :

"Un système est une entité complexe traitée (eu égard à certaines finalités) comme une totalité organisée, formée d'éléments et de relations entre ceux-ci, les uns et les autres étant définis en fonction de la place qu'ils occupent dans cette totalité et cela de telle sorte que son identité soit maintenue face à certaines évolutions."

La complexité d'un système provient du fait qu'il peut comporter beaucoup de sous-systèmes interconnectés et pouvant avoir des objectifs en conflit. L'analyste doit disposer de méthodes et d'outils capables de représenter ces systèmes, sans recourir à une simplification grossière, origine d'une perte d'information ; ni à la construction de modèles complexes, inexplicables par la suite à cause d'un manque de documentation, de problèmes de validation des modèles, ou de portabilité de programme.

Exemple :

Une usine de production constitue un très bon exemple de système dont les parties pourraient être :

- main d'oeuvre
- service achats
- service approvisionnement
- service de gestion de stocks
- service Fabrication
- service Ventes
- service Ordonnancement de la production
- service Administratif

Chacune de ses parties a ses lois (règles de fonctionnement) et est partiellement

indépendante des autres mais aussi fonction des autres et réagit sur elles. La connaissance des lois d'interaction et leur application à l'ensemble va influer directement sur le but à atteindre (rentabilité, investissement, bénéfice,...) par le système. L'identification des frontières du système avec son environnement (milieu) permettent de délimiter la portée du système.

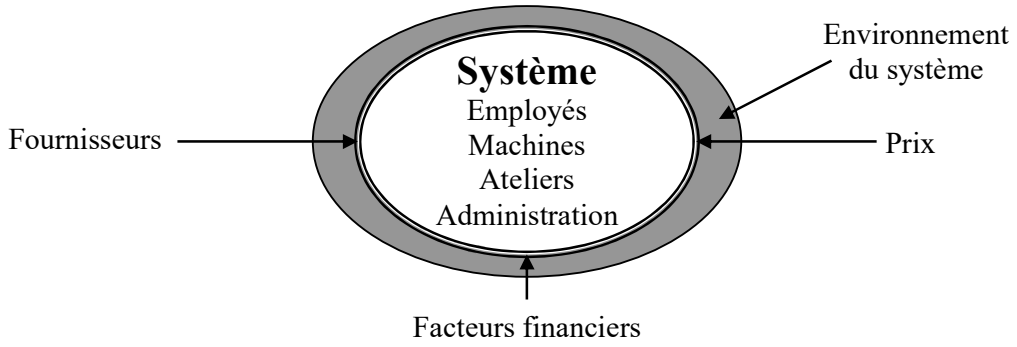

Figure 3.1 : Le système et son environnement

Les frontières d'un système peuvent être physiques; cependant il serait plus juste de parler des relations de cause à effet entre un système et son environnement. Certains facteurs externes peuvent influencer le système. Si ces facteurs contrôlent entièrement la dynamique du système , il n' y a pas d'intérêt à conduire des expérimentation avec ce système et il faudra redéfinir le système. Si par contre ces facteurs contrôle partiellement la dynamique du système, On peut alors soit :
- élargir le système de façon à les inclure
- les ignorer (si on juge que leurs effets sont négligeables)
- les considérer comme des entrées du système [Belattar93].

3.1.1.2. Modélisation de système

La modélisation consiste à construire une représentation simplifiée d'un système appelée généralement « modèle ». Un modèle est donc une représentation d'un système (réel ou imaginé) dont le but est d'expliquer et de prédire certains aspects du comportement de ce système [Belattar93].

Il y a des différentes manières pour diriger l'étude d'un système. La première doit expérimenter avec le système lui-même. Cependant, c'est rarement possible. La raison de cela peut être que c'est techniquement impossible. Par exemple, le placement des sondes dans le système peut les perturber trop. Alors le coût d'une telle expérience peut être très élevé. L'autre raison peut être que le système n'existe pas ou il est partiellement inachevé.
Dans ce cas-là, la création des prototypes est possible, mais généralement chère. En conclusion, la manière d'étude d'un système est de lui faire un modèle. C'est une substitution et simplification du système, qui doit être moins coûteuse que des modifications sur le système ou sa création. Logiquement, un modèle est fondé sur l'information recueillie pour le but et sur des hypothèses faites pour simplifier la prévision du système.
D'ailleurs il y a des différentes natures des modèles.

3.1.1.3. Types de modèles

La figure 3.2 présente une classification des différents genres de modèle [Belattar93]. Les modèles sont d'abord séparés en modèles " physiques " et " mathématiques ". D'une part les modèles physiques sont des systèmes concrets basés sur une certaine analogie avec le système original. Par exemple, un système et un modèle de lui ont des propriétés semblables dans quelques contextes. D' autre part, les modèles mathématiques sont des abstractions en utilisant des notations symboliques. Des notations peuvent être équations mathématiques aussi bien que n'importe quelle autre forme de formalisme, habituellement fournie par une analyse ou une méthode de conception.

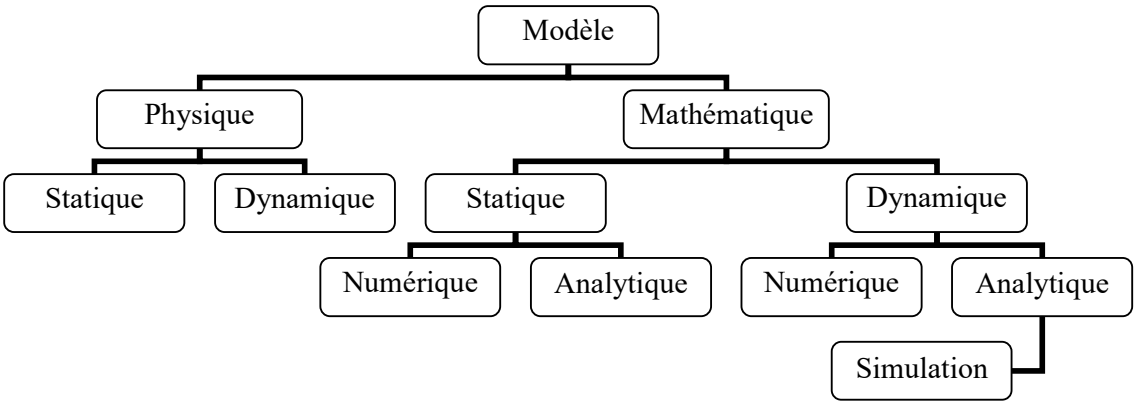

Figure 3.2 : Les types de modèle

Une autre distinction est faite entre les modèles "statiques" et "dynamiques". Dans un modèle statique, la notion de temps est non considérée. L'intérêt est l'état du système quand il arrive à un équilibre, tandis que les modèles dynamiques introduisent explicitement le facteur de temps, l'intérêt est l'évolution de l'état du système. Par exemple, le modèle de la molécule illustre la notion du modèle physique statique, alors que le modèle électrique RLC de ressort représente un modèle physique dynamique. De même façon, les équations qui représentent un système peuvent introduire le variable de temps ou non. Ceci distingue respectivement les modèles mathématiques statiques et dynamiques. Alors, une troisième distinction est faite pour les modèles mathématiques, elle dépend de la façon avec la quelle ils sont résolus. On distingue deux catégories de méthode "analytiques" et "numériques". Les méthode analytique consistent dans l'utilisation d'un raisonnement déductif de théorie mathématique pour résoudre un ensemble d'équations et fournir une solution exacte, tandis que les méthodes numériques soient basées sur des procédures de calcul pour approcher les solutions. Les modèles de simulation sont considérés comme modèles numériques.

3.1.1.4. Des concepts utilisés dans la modélisation

Il y a quelques principes pour construire un modèle [Bachelet98] :

- Construction de blocs: Le système peut être décomposé en blocs considérés eux même comme des systèmes. Habituellement, ils sont appelés "sous-systèmes". Pour un sous-système, les autres sous-systèmes deviennent son environnement.
- Pertinence: Seulement les détails pertinents d'un système qui doivent être modelisés. Les détails non pertinents ne font que compliquer le modèle. Le choix des détails à

introduire dépend directement au but de l'étude et de l'effet de ces détails dans la modélisation.

- Précision: La précision de l'information introduite dans le modèle doit être prise en considération, une bonne précision doit être fournit pour les informations clés du modèle. On trouve deux difficultés: quels sont les informations clé du système et comment vérifier le résultat de la simulation?

3.1.2. Simulation

La simulation représente la méthode de résolution des modèles mathématique. Simulation désigne imitation de fonctionnement du système réel à travers un modèle [Banks00], on peut dire aussi que La simulation est l'étude du comportement dynamique d'un système, grâce à un modèle que l'on fait évoluer dans le temps en fonction des règles bien définies, à des fins de prédiction [Belattar93]. Dans le cas d'une simulation assistée par ordinateur, la simulation d'un système exige que le modèle doit être traduit en un formalisme compréhensible par l'ordinateur, la simulation revient à l'exécution sur ordinateur d'un programme écrit dans un langage spécifique.

3.1.2.1. Quand et pourquoi simuler ?

Quand le modèle est définit, on doit choisir une méthode pour le résoudre. Quelquefois, on peut utiliser des méthodes analytiques, mais malheureusement, les problèmes résolus par ces techniques sont limités. Les méthodes analytiques impliquent beaucoup de simplifications du système original, ce qui limite la complexité des systèmes étudiés.

D'autres fois, la résolution analytique est virtuellement possible mais prend un long temps de calcul. Pour ces raisons, la simulation parait souvent la meilleure solution pour résoudre les problèmes concernant des systèmes très complexes. La simulation présente alors beaucoup d'avantages par rapport à d'autres méthodes.

3.1.2.2. Terminologie

Nous donnons ci-après une liste de termes utilisés dans le domaine de la simulation.

· ***Les Entités*****:** Ce sont les composants (objets) passifs du système, qui subissent des opérations (clients par exemple).

· ***Les Ressources*****:** Ce sont les composants (objets) actifs du système, qui réalisent des opérations (serveurs par exemple).

· ***Les Attributs*****:** Ce sont les caractéristiques, les propriétés des objets (entités et ressources), les attributs peuvent changer leur valeur au cours de la simulation.

· ***L'Etat du Système*** : Il est déterminé par la valeur des attributs des objets, ainsi que les relations entre ces objets, à un moment donné. L'évolution du système dépend des changements d'état du système.

· ***Les Evénements*** : Ce sont les point de changement d'état d'un système elle sont caractérisées par une date (temps) d'apparition.

· ***Les Activités*** : Une activité est une tâche qui peut inclure un ensemble d'opérations. Chaque activité commence et se termine par un événement.

· ***Les Processus*** : Un processus est le regroupement d'une séquence d'événements chronologique dans lequel ces événements se produiront.

Si on prend comme exemple un service banquière, au niveau d'un guichet le serveur est une ressource, le client est une entité. Pour qu'un client passe au guichet il doit suivre le processus suivant :

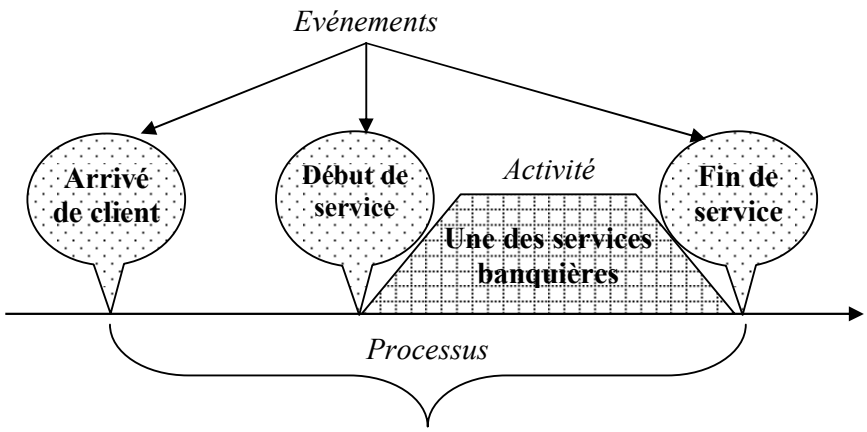

Figure 3.3:Un processus décrivant un service banquière

3.1.2.3. Etapes d'un projet de simulation

Plusieurs présentations décrivant les étapes d'une étude de simulation sont proposées, on va choisir celui de Banks, Carson, Nelson, et Nicol [Banks00]. Un projet de simulation doit passer par les étapes illustrées dans le schéma suivant:

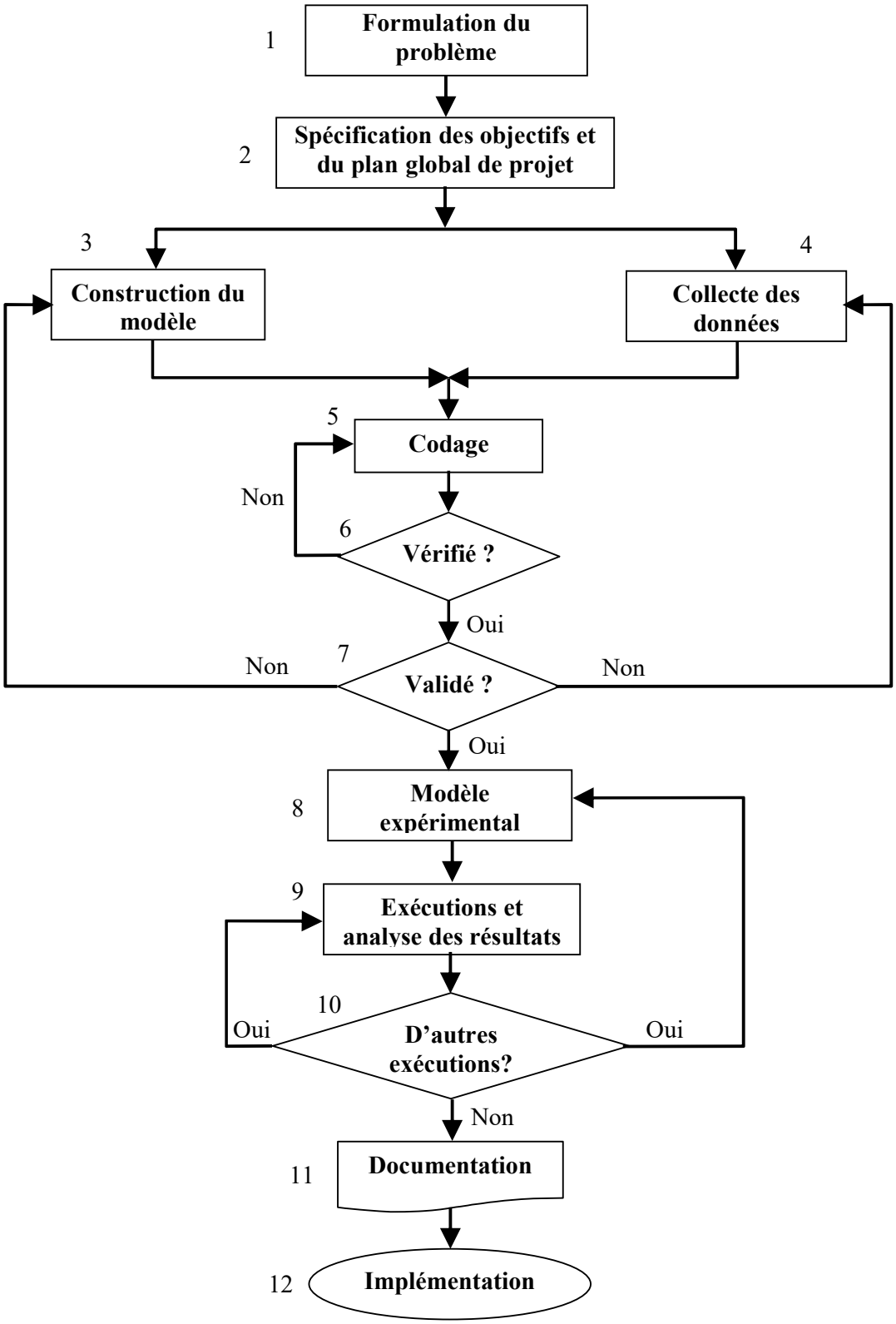

Figure 3.4 : Les Etapes d'une étude de simulation [Banks00]

3.1.2.3.1. Formulation du problème

Toute étude de simulation commence par une présentation du problème faite par le client (celui qui a le problème à résoudre). Puis l'analyste (de simulation) doit faire l'extrême attention pour assurer que le problème soit clairement compris. Si un rapport sur le problème est préparé par l'analyste, il est important que le client concorde avec la formulation. Même avec toutes ces précautions, il est possible que le problème devra être reformulé tant que l'étude de simulation progresse.

3.1.2.3.2. Spécification des objectifs et du plan global du projet

Une autre manière d'énoncer cette étape est de "préparer une proposition". Cette étape devrait être accomplie indépendamment de l'analyste et du client, entant qu'un consultant externe ou interne.

Les objectifs indiquent les questions qui doivent être répondues par l'étude de simulation. Le plan du projet devrait inclure le rapport des divers scénarios qui seront examinés. Les plans pour l'étude devraient être indiqués en termes du temps qui sera demandé, du personnel qui sera employé, des besoins matériels et des logiciels si le client veut exécuter le modèle et conduire l'analyse, les phases dans l'investigation, le rendement à chaque phase, le coût de l'étude ,...etc.

3.1.2.3.3. Construction du modèle

Le système réel à l'étude est abstrait par un modèle conceptuel, des séries de relations mathématiques et logiques concernant les composants et la structure du système. Il est recommandé que le modèle initial soit simple et qu'il augmente jusqu'à ce qu'un modèle de complexité appropriée ait été développé.

La construction d'un modèle trop complexe s'ajoutera au coût de l'étude et au temps pour son accomplissement sans augmenter la qualité de la sortie. La participation de client augmentera la qualité du modèle résultant et augmentera la confiance de client en son utilisation.

3.1.2.3.4. Collecte des données

Une fois le problème formulé et les objectifs visés identifiés, il faudra établir un inventaire des besoins en données sur le système réel et le soumettre au client. Dans le meilleur des cas, le client dispose déjà des données nécessaires qu'il conserve dans le format adéquat et que l'analyste n'aura qu'à utiliser.

Cependant dans la pratique, ce cas est vraiment très rare. En effet, très souvent le client pense disposer de toutes les données requises mais en réalité lorsque l'analyste prendra compte de celles-ci, elles s'avéreront très différentes de celles dont il a besoin.

3.1.2.3.5. Codage du modèle

Dans cette étape le modèle conceptuel construit dans l'étape 3 est codé dans une forme reconnaissable par ordinateur (un modèle opérationnel) en utilisant un langage de simulation parmi ceux disponibles.

3.1.2.3.6. Vérifié ?

Elle concerne le modèle opérationnel (programme). Il s'agit de s'assurer que le modèle s'exécute sans erreurs. La vérification est primordiale même pour des modèles de taille réduite car ces derniers peuvent aussi comporter des erreurs bien qu'ils soient très petits comparés à des modèles de systèmes réels par essence complexes. Il est fortement

recommandé que l'analyste de simulation attende jusqu'à ce que le modèle entier soit complet pour commencer le processus de vérification. En outre, l'utilisation d'un contrôleur d'exécution interactif, ou le débogueur, est fortement encouragée comme un aide au processus de vérification.

3.1.2.3.7. Validé ?

La validation est la détermination que le modèle conceptuel est une représentation précise du système réel. Le modèle peut-il être substitué au système réel pour les buts de l'expérimentation? S'il y a un système existant, on l'appelle le système base, alors la manière idéale de valider le modèle est de comparer son rendement à celui du système base. Malheureusement, il n'y a pas toujours un système base. Il y a beaucoup de méthodes pour effectuer la validation.

3.1.2.3.8. Modèle expérimental

Pour chaque scénario qui doit être simulé, les décisions doivent être faites au sujet de la longueur de l'exécution de simulation, le nombre d'exécutions (également appelées les répliques), et de la façon de l'initialisation, comme il est exigé.

3.1.2.3.9. Exécutions et analyse des résultats

Les exécutions de production, et leur analyse suivante, sont employées pour estimer les mesures de performance pour les scénarios qui sont simulés.

3.1.2.3.10. D'autre exécutions?

Cette étape est basée sur l'analyse des exécutions qui ont été accomplies, l'analyste de simulation détermine si les exécutions additionnelles sont nécessaires et si les scénarios additionnels doivent être simulés.

3.1.2.3.11. Documentation

La documentation est nécessaire pour de nombreuses raisons. Si le modèle de simulation va être employé encore par les mêmes ou les différents analystes, il peut être nécessaire de comprendre comment le modèle de simulation fonctionne. Ceci permettra la confiance en modèle de simulation de sorte que le client puisse prendre des décisions basées sur les analyses. En outre, si le modèle doit être modifié, ceci peut être considérablement facilité par une documentation adéquate. Le résultat de toutes les analyses devrait être rapportés clairement et avec concision. Ceci permettra au client de réviser la formulation finale, les alternatives qui ont été adressées, le critère par lequel les systèmes alternatifs ont été comparés, les résultats des expériences, et les recommandations de l'analyste éventuelles.

3.1.2.3.12. Implémentation

L'analyste de simulation agit en tant que journaliste plutôt qu'avocat. Le rapport préparé dans l'étape 11 se tient sur ses mérites, juste l'information additionnelle que le client emploie pour faire une décision. Si le client a été impliqué tout au long de la période d'étude, et l'analyste de simulation a suivi toutes les étapes rigoureusement, alors la probabilité d'une implémentation réussie est augmentée.

3.1.2.4. Simulation continue et discrète

Selon la nature de système modélisé on doit choisir le type de simulation, nous avons dit précédemment que les systèmes sont classifiés en systèmes continus et systèmes discrets. De même façon on trouve des modèles de simulation continus, discrets ou bien combinés '*mixtes*'. La différence c'est que dans les modèles continus le système est décrit par des variables '*variables d'état*' qui changent de valeur d'une manière continue (dans n'importe quel instant les variables peuvent avoir un changement de valeur), mais dans les modèles discrets l'axe de temps est découpé en intervalles 'Δt' tel qu'une variable ne peut changer sa valeur qu'à des points $p=n* \Delta t$. On peut avoir aussi une combinaison entre la simulation continue et discrète c'est la simulation mixte où on trouve des variables continus et des autres discrets.

3.1.3. Simulation à événements discrets

3.1.3.1. Les événements discrets et le temps

Comme c'était mentionné précédemment, les changements d'état de la simulation discrète, par opposition à la simulation continue, ont lieu de manière discrète dans le temps. Ces changements correspondent à des événements dans la simulation qui peuvent arriver dans un intervalle régulier de temps (gestion du temps dirigée par horloge) ou guidée par des événements asynchrones (gestion du temps dirigée par événement).

Il convient cependant de donner des précisions sur la notion de temps. Le temps de la simulation est un temps virtuel qui progresse de manière discrète et indépendamment du temps réel (continu). Bien entendu, une simulation n'a d'intérêt que si ce temps virtuel s'écoule plus rapidement que le temps réel, sauf si la simulation ne peut pas être remplacée par un essai réel (ce qui reste le cas pour de nombreux problèmes de physique quantique)[Campos00].

3.1.3.2. Les grandes approches de modélisation

Pour construire un modèle de système on doit choisir une des trois grandes approches (world views) [Belattar93][Bachelet98] suivantes :

- *Approche par événement (Event-scheduling world view),*
- *Approche par activités (Activity Scanning world view),*
- *Approche par interaction de processus (process-interaction world view).*

La combinaison entre les deux approches *par événement* et *par activités* est proposée dans une approche appelée *Approche de trois phases*.

3.1.3.2.1. Approche par événement

Dans cette approche, un système est modélisé en définissant les changements qui ont lieu aux instants d'occurrence des événements. Le concepteur doit donc déterminer les événement qui peuvent changer l'état du système et de développer ensuite la logique associée avec chaque type d'événement. La simulation est alors effectuée en exécutant la logique associée à chaque événement selon une séquence ordonnée en fonction des dates d'occurrence de ces événements. C'est pour cela que cette approche est qualifiée d'approche par 'planification d'événement' car les temps des événements futurs sont codés explicitement dans le modèle, comme ils sont planifiés pour se produire au fur et a mesure que la simulation progresse.

3.1.3.2.2. Approche par activités

Cette approche est très populaire mais moins efficace et moins naturelle que l'approche par événements, par conséquent elle n'a pas été très adoptée.

Elle est très similaire à la programmation logique par règles de production : ***Si*** *condition(s)* ***Alors*** *Action(s)*. Le concepteur décrit les activités dans lesquelles les objets du système s'engagent et prescrit les conditions qui causent le début ou la fin d'une activité. Les événements de début ou de fin d'une activité ne sont pas planifiés par le concepteur mais ils sont initiés à partir des conditions spécifiés pour l'activité.

Au fur et à mesure que le temps de la simulation progresse, les conditions de début ou de fin d'une activité sont examinées (scrutées). Si ces conditions sont satisfaites alors les actions appropriées sont exécutées.

Afin de s'assurer que toutes les activités sont prises en compte, il est nécessaire de scruter toutes les conditions à chaque progression du temps. Ainsi, avec cette approche, on ne répertorie plus les différents types d'événements mais les différents types d'activités possibles avec leurs conditions de début et de fin.

C'est une approche gourmande en temps de calcul puisque à chaque progression du temps, des tests sont faits même pour les activités qui ne peuvent pas débuter ou finir. Elle est en général réservée aux cas où les activités ont une durée non connue à priori.

3.1.3.2.3. Approche par interaction de processus

Avec cette approche le concepteur concentre aux entités du système et plus précisément à leur cycle de vie. Elle est la plus intuitive est utilisée par beaucoup de langages de simulation (Simula , GPSS, SIMAN, ...). En fait, le concepteur va décrire les étapes de la vie d'une entité *'processus'*. Un processus est une séquence d'activités et événements, la progression de processus représente l'évolution de l'entité. Il peut être suspendu, signifie que l'entité reste en attente d'un événement pour continuer sa évolution.

Généralement, il est associé avec le fait que pour effectuer une activité, une entité a besoin des ressources. Dans cette approche l'avancement du temps ressemble à la première approche, par le prochain événement.

3.1.3.2.4. Approche des trois phases

Une combinaison de deux approches *par événement* et *par activités* est proposée dans une approche appelée *trois phases* [Bachelet98]. Les événements sont considérés comme des activités avec une durée égale à zéro. Avec cette définition les activités seront séparées en deus groupes B et C. Les activités de type-B sont ceux qui sont planifiés, préprogrammés et correspondent à l'approche *par événement.* Ceci permet l'avancement du temps du prochain événement et une liste des événements planifiés est maintenue. Les activités de type-C sont ceux qui sont conditionnelles en certaines conditions qui soient vrais et qui correspondent à l'approche *par activités.*

Donc, la technique consiste à l'avancement du temps à l'événement plutôt, utilisant la liste des activités de type-B planifiées. Toutes les activités de type-B produisant à ce temps sont exécutées et alors, les conditions sont vérifiées pour exécuter éventuellement les activités de type-C. Alors, le temps est encore incrémenté au prochain événement de la liste des activités de type-B, et ainsi de suite. La combinaison de ces deux approches permet la modélisation des problèmes de ressources plus complexes.

3.1.3.3. Les composants d'un modèle de simulation à événement discret

Les modèles de simulation a **événements discrets** ont tous plusieurs composantes en communs et l'organisation logique de ces composantes permet d'aider le codage, la mise au point, et la mise à jour du programme traduisant le modèle de la simulation. En particulier, les composantes suivantes se retrouvent dans la plupart des **modéles de simulation à**

événements discrets adoptant l'approche par ‘‘événement’’:
- **L'état du système:** défini par une collection de variables d'état décrivant l'état du système à un temps particulier.
- **L'horloge de la simulation:** variable indiquant la valeur courante du temps de la simulation.
- **La liste des événements:** Une liste qui contient les dates d'occurrence des événements avant avoir lieu dans le futur.
- **Des compteurs statistiques:** variables utilisées pour collecter des informations statistiques sur les performances du système.
- **La routine d'initialisation :** sous programme servant a initialiser le modèle de simulation au temps zéro.
- **Routine de Gestion du temps (contrôle du temps):** sous programme qui détermine le prochain événement a partir de la liste des événements et initialise l'horloge de simulation avec la date d’occurrence de cet événement courant.
- **Routines associées aux événements:** sous programme qui met à jour l'état du système quand un type particulier d'événement se produit (il y a une routine d'événement pour chaque type d'événement).
- **Bibliothèque des routines utilitaires:** ensemble de sous programmes utilisés pour générer des variables aléatoires identifiées comme faisant partie du modèle de simulation.
- **Générateur des résultats:** sous programme qui calcule des estimations (a partir des compteurs statistiques) des mesures de performance désirées et génère en fin de simulation un rapport.
- **Le programme principal:** chargé d’initialiser l’état du système au début de la simulation; toutes les variables utilisées au cours de la simulation. Il invoque la routine de gestion du temps pour déterminer le prochain événement devant avoir lieu et passe le contrôle à la routine associée a cet événement. Il contrôle aussi si la fin de la simulation est atteinte et invoque dans ce cas le générateur de résultats [Belattar93].

3.1.3.4. Les étapes d'un programme de simulation

L'exécution d'un programme de simulation peut être séparé en trois étapes. La première phase est la génération du modèle. Les entités du modèle sont crées, initialisées et les premiers évènements peut être planifiés, selon l'approche considérée. De cette manière le modèle est préparé pour être exécuté. Alors le modèle est exécuté, le temps est incrémenté, les évènements se produisent, les activités sont exécutés jusqu'à le temps atteint une limite donnée ou bien jusqu' à ce qu’il n'y a plus d'activité à exécuter. Pendant l'exécution, des données sont rassemblées. En conclusion, des résultats sont retournés sous forme de rapport.

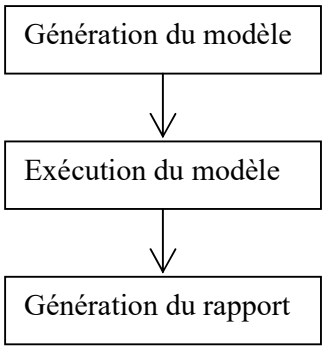

Figure 3.5 : Les étapes d'un programme de simulation

3.2. Simulation et technologie de collaboration

Durant la cycle de vie d'un projet de simulation, beaucoup de personnes sont sollicitées pour l'alimenter en données ou en expertises afin qu'il aboutisse (statisticien, ingénieur, programmeur, client, etc). Mais malheureusement, il est rare de trouver un cadre possédant à lui seul des connaissances dans des domaines variés (Probabilité et Statistique, Modélisation, Programmation, etc) sans avoir recours aux connaissances d'un certain nombre de collaborateur.

Tout projet de simulation implique donc de près ou de loin un groupe de personnes qui travaillent en collaboration.

3.2.1 Le travail collaboratif

Beaucoup de termes utilisés portent sur la notion de groupe ou travail collaboratif :
-*TCAO* : Travail Collaboratif (coopératif) Assisté par Ordinateur, *CSCW* en anglais.
- *Collecticiel* : Une application collective (multi-utilisateurs), *Groupware* en anglais, c'est un terme appelé aux systèmes logiciels spécialisés assistants le travail collaboratif d'un groupe ou d'une équipe[Knave03].
- *Teamwork* : travail d'équipe.
- *Collaboratoire* : Un laboratoire virtuel multi-utilisateur qui permet aux chercheurs séparés géographiquement de travailler ensemble [Cristea98].

Les termes les plus utilisé sont *TCAO* et *Collecticiel,* le collecticiel est une application qui supporte le travail coopératif ou collaboratif, il assiste un groupe d'individus impliquées dans une tâche commune. Le terme *TCAO* (*Travail Coopératif Assisté par Ordinateur*) a été introduit pour désigner tous les travaux et toutes les applications développées pour des groupes d'utilisateurs [Owezarski96]. Le but de ces applications n'est pas seulement de prendre en compte les interactions entre une personne et un ordinateur, mais également les interactions entre un groupe de personnes à travers d'un réseau d'ordinateurs. Ici, il n'y a pas uniquement un dialogue homme/machine mais également des échanges entre des entités (individus ou agents) via un système informatique. Pour construire tels environnements, il est essentiel de bien définir et comprendre les notions induites par le travail coopératif.

3.2.1.1 Quelque notions

Il y a plusieurs notions concernant une application coopérative :

- **Le groupe et l'individu** : chaque participant est un individu personnalisé qui appartient à un ou plusieurs groupes de participants. Un groupe est un ensemble d'individus travaillant sur un même domaine.
- **Les rôles** : dans chaque groupe et à un instant donné, chaque participant joue un rôle [Tarpin97][Ben-Atallah97]. Ce rôle est caractérisé par l'ensemble des droits et des devoirs du participant vis-à-vis de ses partenaires et des données partagées. Il peut évoluer au cours du temps comme il peut être constitué de plusieurs rôles élémentaires.
- **Les vues** : chaque participant peut avoir sa propre perception (vue) des entités manipulées collectivement. La notion de vue inclut à la fois des choix de représentation, et des choix d'interaction [Tarpin97]. En fait, une vue peut être considérée comme un *filtre bidirectionnel* sur un ensemble d'entités partagées : elle définit la *représentation* des entités vers l'utilisateur et autorise ou interdit les *accès* à ces entités. Une vue peut être publique (accessible par d'autres utilisateurs), privée (accessible seulement par le propriétaire) ou semi-privée (publique pour un sous-groupe ou bien intégrant quelques éléments publics et d'autres privés, etc.).
- **Le WYSIWIS** (What You See Is What I See) : il a pour but d'assurer la *rétroaction du groupe*, c'est-à-dire de permettre à chaque participant de voir ce qu'un autre participant voit

ou fait. Cette notion est, en fait, modulable. Lorsque la vision est vraiment identique chez plusieurs participants, on parle de WYSIWIS strict, mais que lorsque la vision d'une même situation est différente, on parle de WYSIWIS relâché[Tarpin97][Ben-Atallah97].

3.2.1.2 Paramètres d'un collecticiel

Modes d'interaction : collecticiels synchrones/asynchrones

Les collecticiels offrent plusieurs possibilités pour échanger des informations entre les participants. Un collecticiel est dit *synchrone* lorsque les informations sont échangées selon un mode temps réel. Dans ce cas, les acteurs sont avertis en temps réel des actions menées sur les ressources partagées.

Le mode d'interaction synchrone nécessite que les membres du groupe soient présents en même temps pour effectuer le travail coopératif. Les systèmes de téléconférence sont des exemples représentatifs de ce mode de fonctionnement. Ce mode permet le partage immédiat des informations.

Par contre les collecticiels asynchrones, au contraire, correspondent aux applications coopératives dont le support de communication est asynchrone, de type courrier électronique. Le mode d'interaction asynchrone ne nécessite pas la co−présence des différents participants. Ce mode est fréquemment employé dans l'édition coopérative des documents. Le mode asynchrone permet le partage séquentiel des informations.
La classification des collecticiels en fonction du mode d'interaction ne recouvre pas complètement l'ensemble des tâches coopératives. L'édition coopérative représente un exemple de collecticiels qui peuvent être à la fois synchrone et asynchrone.

Cohérence

Dans les collecticiels synchrones, le mode WYSIWIS strict définit une *cohérence forte* du contexte partagé. Selon ce mode, tous les acteurs ont une vue identique des données partagées.

La *cohérence faible* peut être engendrée par le mode WYSIWIS relaxé selon l'aspect du temps. Une telle relaxation donne dans le cas extrême, où aucune hypothèse n'est faite sur le mode d'interaction, un fonctionnement asynchrone.

Cependant, le mode WYSIWIS relaxé selon l'aspect visuel n'affecte pas la cohérence du contexte partagé. Même si les acteurs n'ont pas la même vue des informations, l'état de ces informations est identique pour tous.

En conclusion, dans les collecticiels, la cohérence est plutôt liée au mode d'interaction qu'aux différentes variantes du mode WYSIWIS.

Granularité

La *granularité* est une caractéristique spatiale qui s'oppose au mode d'interaction, elle est fréquemment associée aux droits d'accès, aux données attribués, et aux utilisateurs du collecticiel. La granularité définit l'unité d'information (le mot, le paragraphe, le document, ...) qui peut être accédée simultanément par plusieurs utilisateurs.

Une granularité fine associée à un mode d'interaction synchrone définit un travail *fortement couplé*. Dans le cas contraire, une granularité *forte* associée à un mode asynchrone définit un travail *faiblement couplé*.
Par exemple, un éditeur de texte permettant l'accès simultané à un même mot permet une granularité plus fine qu'un éditeur de texte qui ne permettant pas l'accès au document d'un utilisateur à la fois.

Dans le cas des collecticiels qui utilisent des politiques à *jeton*, si celui−ci est attribué pour accéder tout un document, nous serons donc dans le cas d'une granularité *forte*.

Droits d'accès, droit de parole, politiques à jetons

Les *droits d'accès* associés à un utilisateur sont dans une partie constitués des attributions qui lui sont conférées par son rôle, et en partie par les droits qui peuvent lui être confiés pendant le déroulement du travail coopératif. Ces droits, dynamiquement attribués, sont communément appelés *droits de parole*.

Dans les collecticiels, le droit de parole est échangé entre les participants à l'aide de *politiques à jeton*. Il s'agit de différents protocoles qui permettent aux participants de se communiquer une information de contrôle désignant un *jeton* logiciel.

En général, les politiques utilisent un seul jeton. Ce procédé permet de garantir un accès exclusif aux informations partagées. Toutefois, dans certaines applications, plusieurs jetons peuvent coexister, c'est souvent le cas des applications de jeux virtuels où *m* utilisateurs parmi *n* (*m* δ *n*) peuvent modifier le contexte partagé.

Les politiques de passage du jeton sont regroupées en deux classes. Celles−ci sont respectivement fondées sur :

- la désignation explicite ; Un acteur se charge de passer explicitement le jeton à un autre membre du groupe. Celui−ci peut être l'ancien détenteur du jeton ou un acteur à qui revient la responsabilité d'organiser le travail et de coordonner les interventions des participants,
- le passage implicite ; Dans ce cas les acteurs n'interviennent pas explicitement dans l'affectation du jeton. L'application se charge de désigner le détenteur du jeton en fonction de l'ordre d'arrivée des demandes pour l'acquisition du jeton (e.g. ordre FIFO), ou en fonction d'une quelconque priorité instaurée entre les participants.

Gestion dynamique de groupes : protocoles de connexion/déconnexion de participants dans un collecticiel

Le service de gestion de groupe dans un collecticiel fournit principalement des protocoles permettant : la connexion de nouveaux membres et la déconnexion d'anciens membres.

La participation dynamique de participants figure parmi les fonctions spécifiques fournies par un collecticiel synchrone. Cette fonction marque le passage d'un utilisateur d'un environnement de travail privé à un environnement de travail coopératif.

Afin de garantir la flexibilité d'utilisation, le collecticiel doit permettre à des retardataires de se connecter pendant le déroulement du travail. La complexité de cette opération consiste à, fournir à un nouvel arrivant une vue "cohérente" du contexte partagé et à informer les membres du collecticiel de l'arrivée d'un nouveau membre.

De même, le collecticiel doit gérer le départ de certains membres qu'il soit volontaire ou résultant d'une panne quelconque du réseau ou des sites.

En résumé, deux contraintes régissent l'exécution des fonctions de participation et de déconnexion. Elles sont respectivement spatiales et temporelles :

- cohérence du contexte partagé : pour les collecticiels adoptant un mode de fonctionnement WYSIWIS, à l'issue de la phase de connexion, le nouvel arrivant doit avoir une vue identique du **contexte partagé**. Dans ce cas, le protocole garantit une cohérence forte,
- temps de réponse : le traitement des demandes de connexion ou de déconnexion doit être asynchrone. le protocole de connexion doit garantir les **temps** de réponse faibles aussi bien pour le demandeur de la connexion que pour les acteurs du collecticiel.

Remarques

La cohérence du contexte n'est pas absolue, elle est liée au rôle, qui est conférés à l'utilisateur au moment de son adhésion au groupe. Par exemple, si un utilisateur n'est autorisé qu'à

partager une application particulière du collecticiel en mode WYSIWIS, le service de gestion de groupe ne lui fournit que le contexte de l'application en question.
La fonction de déconnexion marque la sortie d'un acteur du groupe de participants.
Le protocole de déconnexion doit garantir un fonctionnement normal du collecticiel pendant cette phase. En général, le protocole doit s'assurer que le participant n'a pas de jeton, ou qu'il n'occupe pas un rôle qui peut entraver le déroulement du travail (par exemple, le rôle de président doit être cédé avant de quitter la session).
Les opérations de connexion/déconnexion dynamiques sont accomplies par des protocoles qui sont souvent choisis en fonction du type du travail coopératif.
Dans certains collecticiels, la connexion d'un utilisateur peut être soumise à l'accord d'un *président*. Dans d'autres applications telles que les téléconférences simulant par exemple un grand séminaire public, la connexion d'un nouvel arrivant est complètement transparente aux acteurs du collecticiel.

Session

Une session permet d'associer : un groupe d'acteurs, un contexte partagé, des rôles, des protocoles d'accès, et des protocoles de connexion/déconnexion, afin de définir selon la dimension *espace* un travail coopératif. La définition complète du travail nécessite la donnée du mode d'interaction adopté. Souvent, le terme *session* est employé pour définir une durée d'exécution d'un collecticiel synchrone.

3.2.1.3. Classifications

Les collecticiels peuvent être catégorisés en plusieurs taxonomies :

3.2.1.3.1. Taxonomie espace−temps

La taxonomie la plus reconnue est celle fondée sur les critères *espace−temps* [Knave03][Renevier04][Tarpin97] [Husman04][Lorcy00][Vapillon00]. Il s'agit de classer les collecticiels dans un repère à deux dimensions (ou bien une matrice) espace−temps. L'axe *espace* désigne la distance géographique séparant les acteurs, alors que l'axe *temps* définit le mode d'interaction (synchrone/asynchrone).

	Co-situé	**A distance**
Synchrone (même temps)	salle de conférence	Editeur partagé, chat
Asynchrone (temps différents)	Forum électronique, Calendriers électroniques partagés	Email, systèmes de messages

Figure 3.6 : Classification Temps-Espace

Cependant, la taxonomie *espace−temps* marque quelques insuffisances pour représenter l'ensemble des collecticiels. Par exemple, les collecticiels d'édition coopérative présentent un mode d'interaction "variable" synchrone/ asynchrone. En résumé, les limitations de la taxonomie *espace−temps* sont multiples :
– la classification ne permet pas de placer distinctement tous les collecticiels dans l'une des quatre classes identifiées,
– le critère espace ne parait pas fondamental dans la classification des collecticiels,
– plus d'informations sur la nature et le fonctionnement sont généralement requises pour une classification.

Une autre classification très intéressante est fondée sur la nature du collecticiel.
Plus précisément, il s'agit de grouper les collecticiels en tenant compte des fonctions qu'ils offrent. Selon cette approche, cinq classes de collecticiels sont identifiées[Knave03][Ben-Atallah97][Husman04].

3.2.1.3.2. Taxonomie fonctionnelle

Messageries électroniques

La messagerie électronique est un outil inspiré du courrier postal. Le large diffusion de ce type d'application a fait du courrier électronique l'application du collecticiel la plus répandue. La facilité d'utilisation apportée par le courrier électronique, engendre des problèmes de surcharge généralement provoqués par la fréquence élevée des réceptions des courriers. Plusieurs travaux intéressés aux aspects de présentation et de structuration des courriers visant à alléger la tâche de l'utilisateur.

Caractéristiques :

Mode d'interaction : asynchrone,
résolution de conflits : pas de conflits,
granularité : message,
participation : implicite,
communication : données.

Éditeurs de documents partagés

Ces applications permettent aux utilisateurs ; de partager un ensemble de documents, de les éditer, d'en modifier le contenu et de visualiser les opérations des autres intervenants.
Il existe plusieurs collecticiels d'édition coopérative existent qui se distinguent essentiellement par le mode d'interaction qu'il fournissent. Certains collecticiels privilégient un travail découplé dans lequel chaque participant a la responsabilité d'un fragment de document.
Le mode d'interaction fournit par l'application est synchrone. Les ensembles d'*items* manipulés dans un texte peuvent être privés, partagés ou publics. L'application n'offrent aucun verrouillage sur les données et une granularité d'accès très fine. La résolution des conflits pouvant survenir pendant la coopération et à la charge des utilisateurs.

Caractéristiques :

Mode d'interaction : synchrone,asynchrone,synchrone/asynchrone,
résolution des conflits : politiques à jetons, libre accès,
granularité : correspondant à des fragments (asynchrone), fine (synchrone),
participation : nécessaire pour le cas synchrone,
communication : données et contrôle.

Téléconférences assistées par ordinateurs

Les réunions de travail et les conférences assistées par ordinateur sont des collecticiels axés sur la communication synchrone. Ces applications ont la lourde tâche de remplacer les réunions autour d'une table, devant un tableau ou un rétro−projecteur.

La particularité des applications de téléconférence est qu'elles mettent à la disposition des utilisateurs des canaux de communication audio/vidéo. Cependant, elles se distinguent par le support technologique utilisé pour l'acheminement du son et de la vidéo.

Les premières applications de téléconférence utilisaient des canaux de communication

analogiques annexes au support informatique.

La seconde génération d'applications de téléconférence présente un environnement du travail complètement intégré. Elle sont souvent équipées du moyen de communication audio/vidéo numériques. Ce type d'application fournit un environnement coopératif (en anglais *Desktop Conferencing*).

En général, il s'agit des fenêtres partagées servant de, tableaux de présentation, d'éditeurs ou d'applications de dessin.

Caractéristiques :

mode d'interaction : synchrone, WYSIWIS strict.
résolution des conflits d'accès : politiques à jetons.
granularité : grande (application partagée).
participation : dépend de la nature des applications (peu importante pour les canaux de communication).
communication : données et contrôle.

Systèmes supports d'aide à la décision de groupe

Ces systèmes permettent d'assister un groupe d'utilisateurs pour les discussions et facilitent l'obtention d'un consensus, soit sur des sujets généraux, soit dans un domaine spécifique (création de document, génie logiciel).
CoNex (*Coordinator and Negociation support for eXpert in design applications*) [Ben-Atallah97] est un environnement expérimental qui offre un support coopératif utilisable dans le cadre de l'ingénierie du logiciel. L'application fournit un éditeur d'arguments pour assister aux négociations entre les concepteurs et les analystes (système de conférence pour l'échange de messages informels). Elle fournit également un éditeur de contrats qui permet aux analystes d'attribuer les tâches à programmer et de responsabiliser les programmeurs. D'autres applications sont issues du domaine de l'intelligence artificielle distribuée et des systèmes multi−agents. Les applications flot de travail, appelée *Workflow* en anglais, assistent et organisent un travail de groupe pour la création de documents au sein d'une entreprise.
Le système permet d'organiser les interventions des acteurs et se charge de véhiculer le document entre les différents points d'un circuit d'élaboration de documents.

Caractéristiques :

Mode d'interaction : asynchrone.
résolution des conflits : mode découplé,
granularité : grande (le document),
participation : peu importante,
communication : données.

Environnements de développement partagé (EDP)

Ils s'agissent d'environnements logiciels qui permettent à une équipe de développeurs de partager des ressources communes (fichiers sources, fichiers exécutables) organisées en versions en fonction de leur date de soumission. Ces environnements sont avec les applications *Workflow* les seuls exemples caractéristiques de collecticiels asynchrones.

Caractéristiques :
Mode d'interaction : asynchrone.
résolution des conflits : mode découplé,
granularité : grande,
participation : implicite,
communication : données.

3.2.1.4. Les fonctionnalités dans un environnement collaboratif

Dans un environnement collaboratif on peut trouver les fonctionnalités suivantes [Knave03][Husman04] :

Chat

C'est une communication basée sur la conversation écrite. Un participant écrit une ou plusieurs phrases à un autre, l'autre participant reçoit ce qui est écrit et répond de sont coté de la même façon.

Le *chat* est caractérisé par deux zones de texte, la première pour la lecture et la deuxième pour l'écriture.

Communication Audio/Vidéo

La *communication audio* permet au membre du groupe de faire une conversation vocale directe, elle permet alors à chacun de comprendre mieux ses collègues.

La *communication vidéo* est plus efficace, a cause des gestes (surtout avec les mains) qui caractérisent ce type de communication. Au plus il y a la possibilité d'exposer une séquence vidéo avec des annotations au même temps.

Partage d'applications

Le *partage* d'une *application* est la distribution de cette dernière à plusieurs utilisateurs qui pourront simultanément voir et éditer les mêmes données.

Il y a deux types d'applications partagées. Le premier type, est une application multi-utilisateurs et spécialement distribué, il s'exécute indépendamment sur les terminaux de tous les utilisateurs et les mêmes données sont traitées avec synchronisation.

Le deuxième type est une application mono-utilisateur qui sera partagée en utilisant un logiciel qui permet de la partager.

Tableau blanc partagé

Le *tableau blanc partagé* est un cas spécial des applications partagées. C'est un espace de dessin qui est partagé par tous les membres du groupe, alors chacun peut dessiner et tout le monde va voir ce qui est dessiné en même temps.

Editeur partagé

L'*éditeur partagé* est un autre type d'application partagée, qui est utilisé pour l'édition, en collaboration des documents texte. Un éditeur partagé permet aux utilisateurs d'éditer simultanément le même document où tout le monde peut voir les changements en temps réel.

Messagerie électronique (Email)

Aujourd'hui l'*Email* est devenu l'outil le plus utilisé pour communiquer via les réseaux (Inter/Intra) net. C'est une communication de type multipoints, on peut envoyer ou recevoir plusieurs messages en même temps.

La *messagerie électronique* est utilisée dans les environnements collaboratifs pour avoir une communication asynchrone.

Tableau de messages

L'idée du *tableau de messages* est inspirée du monde réel. Le principe est que les messages envoyées seront postées dans le tableau et peuvent en suite être lu par chacun avec

l'accès à ce tableau. C'est aussi une communication en mode asynchrone.

Partage des fichiers

Le partage des fichiers permet aux utilisateurs de partager et distribuer des documents et d'autres fichiers entre eux. Le partage des fichiers est souvent un espace partagé dont les fichiers peuvent être téléchargés et puis consultés par chacun.

Le partage des fichiers est une fonctionnalité en mode asynchrone.

3.2.2. Simulation et collaboration

Comme nous avons cité dans la section 3.1, les applications du Collecticiel visent à lier des personnes géographiquement dispersées, en les laissant accomplir des tâches spécifiques en collaboration via Internet. Le processus de développement de simulation prend beaucoup de connaissances et d'expériences de plusieurs domaines et disciplines. Une équipe de développement de simulation a besoin des personnes avec des connaissances de développement de simulation, des connaissances de domaine, des connaissances économiques aussi bien que des décideurs.

Ces personnes sont souvent dispersées géographiquement. La question posée est comment avoir la possibilité que ces derniers travaillent en collaboration via Internet sur un projet de simulation.

3.2.2.1. Environnement collaboratif de simulation

Un environnement collaboratif de simulation est un environnement distribué qui permet à un groupe d'experts d'effectuer ensemble les étapes d'un projet de simulation où les membres sont situés dans des régions géographiquement séparées [Knave03] [Husman04]. Le but est de faciliter le processus de développement de simulation dans ces circonstances.

Pendant toutes les phases du processus de simulation le groupe doit communiquer. Ainsi, une bonne communication dans le groupe est la clé pour un projet réussi. Ceci est non seulement limité à des interactions synchrones (en temps réel), mais également valide pour une communication asynchrone.

3.2.2.2. Activités collaboratifs et fonctionnalités

Dans un environnement collaboratif de simulation, plusieurs activités sont nécessaires pour le développement de simulation. Grâce aux travaux de Knave et Stackenland [Knave03] et de Husman [Husman04], nous distinguons les différentes activités et les fonctionnalités exigeantes.

Pour pouvoir comprendre les besoins d'une simulation dans l'environnement collaboratif nous devons d'abord regarder le flot du travail dans un projet de simulation et essayer d'identifier les différentes tâches impliquées et comment elles seraient accomplies en profitant des fonctionnalités d'un environnement collaboratif distribué.

La première phase de simulation, *définition du problème*, implique de faire des réunions pour convenir sur la définition et la dimension du problème. Les membres de l'équipe doivent pouvoir juger aux réunions formelles et informelles afin de discuter le problème et parvenir à un accord sur la définition. Cette activité exige une *communication audio/vidéo* pour permettre à chacun des membre de discuter et donner son opinion, elle exige de plus le *partage des fichiers,* un archive contenant des informations identifiant le système doit être distribué aux membres de groupe. Après la discussion, un document contenant la décision prise doit être distribué puis stocké. Un environnement collaborative de simulation doit avoir un espace pour le stockage des différents documents.

Continuant à la deuxième phase, *création du modèle conceptuel*, on doit analyser le système physique et créer un modèle conceptuel. Premièrement les données sur le système

physique doivent être rassemblées, *collecte des données*, distribuées, étudiées et discutées, ensuite, le processus de la création du modèle commence, où les idées doivent être produites et discutées entre les membres. Ici les membres de l'équipe devront être capables de s'exprimer facilement et clairement. En outre, les illustrations et les descriptions des modèles qui sont discutés ont besoin d'être stockés de telle manière qu'ils puissent être accédés par tous les membres du groupe. Dans cette phase s'il y a un outil de conception il doit être partagé, donc ; c'est le *partage d'application*. Pour qu'un participant pouvoir facilement et rapidement décrire et distribuer ses idées il a besoin de *tableau blanc partagé.*

Dans la troisième phase le modèle conceptuel est traduit en *modèle d'ordinateur (codage),* on a besoin alors d'un environnement de travail partagé, pour créer le modèle d'ordinateur. Des outils de simulation sont employés pour créer et exécuter le modèle numérique de simulation, ceci exige aux membres de groupe de créer en collaboration le modèle d'ordinateur en utilisant ces mêmes outils. Avant d'avancer à la phase d'expérimentation le modèle d'ordinateur doit être examiné pour vérifier qu'il se conforme au modèle conceptuel. Ceci exige que la simulation soit exécutée pour pouvoir comparer le comportement de l'ordinateur à ce qui est prévu du modèle conceptuel. Afin de faire ceci le groupe doit exécuter le modèle d'ordinateur et analyser les résultats. La programmation du modèle implique un outil spécifique de programmation qui doit être partagé en donnant la main à tous les membres pour écrire, modifier et exécuter le code ,donc, c'est le *partage d'application*.

La quatrième étape est l'*expérimentation*, où le modèle d'ordinateur est combiné avec des données expérimentales et exécutés. Maintenant les membres du groupe doivent conjointement pouvoir exécuter les simulations et analyser les résultats de ces simulations. Encore le groupe ici a besoin d'un environnement de travail partagé, mais cette fois c'est pour exécuter des simulations et pour valider et analyser des résultats de simulation. Indépendamment de la façon dont la simulation est exécutée, en analysant les résultats, le groupe doit regarder les résultats, discuter les conclusions qui peuvent être tirées et vérifier si le modèle doit être changé. Les résultats doivent également être validés pour s'assurer que le modèle est conforme au système physique. On a besoin d'un outil qui affiche les résultats de l'exécution, généralement les résultats sont donnés sous forme des fichiers, alors on a besoin du *partage des fichiers.* De plus pour pouvoir discuter ces résultats et expliquer les idées on a besoin de *tableau blanc partagé.*

La cinquième et dernière phase est la *documentation*, les résultats et les conclusions des expériences sont documentées. Cette étape exige au groupe d'analyser les résultats et donner des conclusions. En documentant alors le projet de simulation, toutes les phases du projet doivent être documenté comme les résultats et les conclusions. La phase de documentation exigera au groupe de conjointement créer, réviser et éditer les documents. De plus les documents doivent être stockés dans une manière qu'ils sont accessibles par tous les membres du groupe. Cette phase exige un *éditeur partagé* pour pouvoir rédiger les documents en collaboration et exige le *partage de fichiers* pour distribuer ces documents à tous les membres.

La communication audio/vidéo est utilisé à chaque fois que les membre désire discuter, le *chat* peut être utilisé comme moyen de communication quand il y a des difficultés pour la communication audio/vidéo.

Dans certaine session de collaboration les membres ne sont pas nécessairement tous participés, mais ils restent en collaboration asynchrone et ils peuvent accéder les documents en mode lecture, de plus ils peuvent utiliser le *tableau de messages* pour envoyer des idées ou des questions.

Finalement nous trouvons que les fonctionnalités impliquées dans un environnement collaboratif de simulation sont :

- Communication audio/vidéo.
- Partage de fichier.
- Partage d'application.
- Editeur partagé.
- Tableau blanc partagé.
- Tableau de message.

3.3. Conclusion

Nous avons expliqués les principes de la combinaison entre la simulation et le travail de groupe (collaboration) et nous avons spécifie les caractéristiques d'un environnement collaboratif de simulation. Nous avons vu aussi que les fonctionnalités qui caractérisent les environnements collaboratifs ajoutent une nouvelle dimension a l'étude de simulation et la rend plus efficace.

Nous avons aussi cité qu'à une phase de développement d'un projet de simulation sauf les spécialistes concernés qui peuvent participer. Bien plus la phase de programmation du modèle seules les programmeurs, qui eux même trouvent des difficultés à l'écriture de code, sont concernés.

En conclusion pour faciliter la programmation des modèles d'une part et pour permettre à tous les membres du groupe de se joindre à la phase de programmation on a besoin d'introduire les concepts de la programmation visuelle.

Alors dans le chapitre suivant nous allons proposer une approche pour la conception d'un environnement de programmation visuelle des modèles de simulation qui supporte le travail de groupe.

CHAPITRE 2

Programmation visuelle et langages visuels

Pour les humains, l'utilisation des images était et reste toujours la façon la plus simple pour l'expression et la communication. L'humain apprend souvent la vie par les images et il pense en formant des images dans son esprit. Les recherches sur la programmation visuelle considèrent cette perception visuelle humaine pour concevoir des systèmes et des langages de programmation qui utilisent des images ou des symboles graphiques, et pour un but éloigné la permission aux non spécialiste de pouvoir programmer sans avoir besoin de maîtriser les techniques de la programmation classique. Ces recherches sont favorisées par l'évolution de la technologie des écrans graphiques (couleurs, résolution,…).

Plusieurs chercheurs ont établi l'avantage des méthodologies graphiques par rapport aux méthodologies textuelles. Le domaine de la programmation visuelle regroupe des recherches sur les systèmes graphiques, les langages de programmation et les interactions homme-machine. Les études sont faites sur les représentations graphiques, sur les paradigmes, sur les caractéristiques du langage et sur son utilisation dans des différents domaines. Nous abordons donc un vaste domaine, et pour l'étudier nous allons au préalable préciser la signification du terme programmation visuelle.

2.1. Définitions

2.1.1. Programmation visuelle

La définition la plus générale de la programmation visuelle est celle donnée dans [Chang90]: *c'est l'utilisation d'expressions visuelles (icônes, dessins, entrées gestuelles,...) dans le processus de la programmation*. Cette définition est adoptée par d'autres auteurs [Rava02]. *La programmation visuelle se rapporte au développement de logiciel où des notations graphiques et des composants logiciels manipulables en mode interactif sont principalement employés pour définir et composer des programmes. Le but de la programmation visuelle est de mettre en valeur la compréhensibilité des programmes et de simplifier la programmation elle-même. Les environnements de programmation visuelle fournissent donc des éléments graphiques ou iconiques qui peuvent être manipulés interactivement par l'utilisateur en fonction d'une grammaire spatiale spécifique pour construire un programme.*

Pour Shu [Shu88], *programmation visuelle signifie utilisation des représentations graphiques significatives dans le processus de programmation. Elle précise ensuite qu'il y a plusieurs aspects au sein de ce processus, et que la programmation visuelle peut s'appliquer à chacun. Les différents aspects sont :*
– le langage et l'environnement utilisés
– l'écriture même du programme
– déterminer si l'ordinateur a effectué ce qui était prévu

– l'affichage des données au cours de l'exécution.
– ...

Pour M.Burnett [Burnett99], *La programmation visuelle c'est programmer dans plus d'une dimension elle est utilisée pour porter la sémantique. Les exemples de telles dimensions additionnelles sont l'utilisation des objets multidimensionnels, l'utilisation des relations spatiaux, ou l'utilisation de la dimension de temps pour spécifier la relation sémantique "avant-après ". Chaque objet ou relation multidimensionnelle, potentiellement significative, est une token (juste comme dans des langages de programmation textuels traditionnels chaque mot est une token) et la collection d'un ou plusieurs tokens donne une expression visuelle. On peut mentionner des exemples d'expressions visuelles utilisées dans la programmation visuelle et qui incluent des diagrammes ; les esquisses à main levée, les icônes, ou les démonstrations des actions effectuées par des objets graphiques. Quand une syntaxe du langage de programmation (sémantiquement significative) inclut des expressions visuelles, on dit qu'il est un langage de programmation visuel (LPV).*

Selon Myers [Myers90], *le système de programmation visuelle se rapporte à n'importe quel système qui permet à l'utilisateur de spécifier un programme dans un mode bidimensionnel (ou plus)*. Bien que cette définition soit très large, les langages textuels conventionnels ne sont pas considérés comme bidimensionnels, étant donné que les compilateurs ou interpréteurs les traitent en flots unidimensionnels.

2.1.2. Langages de programmation visuels

Un langage de programmation visuel est un langage qui manipule des informations visuelles, qui permet une interaction visuelle, ou qui permet de programmer à l'aide d'expressions visuelles.[Rava02]

C'est une représentation illustrée par des entités conceptuelles et des opérations, elle est essentiellement un outil qui permet aux utilisateurs de composer des expressions iconiques ou visuelles [Chang95]. Les sémantiques des LPV sont définies par des représentations graphiques significatives [Shu88].

Un langage de programmation visuelle a les propriétés suivantes:

1. Il décrit des objets spécifiques (structures de données, objets graphiques, etc...) et leur comportement,
2. il utilise des éléments lexicaux visuels (graphiques) et éventuellement textuels,
3. il a une syntaxe spatiale et, au moins, une partie de la sémantique du langage est déterminée par la position et les relations spatiales des éléments lexicaux.

2.1.2.1. Caractéristiques des langages visuels

Les LPVs sont habituellement employés avec des environnements visuels.
Ceci suggère qu'un LPV doive se caractériser par des attributs de sa syntaxe et son environnement. Certaines des caractéristiques communes des LPVs sont [Bany00]:

1- Simplicité.
2- Concret.
3- Description explicite des liaisons.
4- Feedback visuel immédiat.
…….

Les LPVs ont pour but de fournir la simplicité en cours de la création et l'édition des programmes en réduisant le nombre de concepts utilisés pour construire un programme. Les LPVs vérifient ceci en enlevant plusieurs abstractions complexes et difficiles qui sont habituellement employées dans des langages de programmation textuels. Par exemple,
la plupart des LPVs soulagent les programmeurs de définir des variables pour le flux de données implicite dans les langages textuels.
Concrétion signifie « utilisation des valeurs spécifiques, plutôt qu'une description des valeurs possible ».

Les LPVs utilisent une visualisation explicite pour exprimer les liaisons entre les éléments du programme. Par exemple, les organigrammes dans un LPV montrent clairement le flot de contrôle du programme, et les diagrammes du flot de données décrivent les dépendances de données entre les opérations.
Le feedback visuel immédiat fournit la réponse instantanée aux changements faits à un programme. A chaque fois le programmeur fait un changement, le système met à jour automatiquement son affichage. Par exemple, dans des bilans le changement d'une valeur propage immédiatement des changements aux cellules dépendantes.

Les LPVs visent alors, à améliorer la manière avec laquelle les programmeurs expriment des données et des algorithmes dans un programme et la manière d'édition et de modification des programmes. Les LPVs aident les programmeurs à comprendre mieux la logique de leurs programmes par la description explicite des liaisons. Ceci est gras à l'aspect visuel des LPVs.

L'un des buts primaires du des LPVs spécifique est de permettre à des experts et des scientifiques de résoudre les problèmes complexes à leurs domaines d'expertise en programmant sans nécessairement être des programmeurs experts. Dans ce cas-ci le programmeur peut se concentrer sur la solution du problème plutôt que l'ajustage de la solution au langage.

2.1.3. Environnements de programmation visuelle

Les environnements de programmation visuelle portent sur l'aspect visuel des systèmes de programmation [Fost95]. C'est dans le contraste direct avec d'autres environnements de programmation graphiques basés sur des modèles mathématiques tels que MathCad, Mathematica... etc. Le terme EPV a été utilisé dans plusieurs contextes, parfois pour toutes les interfaces utilisateur graphiques (GUIs).Cependant, un EPV peut-être mieux appliqué aux systèmes graphiques où des icônes qui représentent les tâches d'un programme sont reliées dans un organigramme ou un réseau.

L'utilisation des environnements de programmation visuels a crû rapidement durant ces dix dernières années. Littéralement des centaines de produits académiques et commerciaux ont été développées pour l'utilisation dans le prototypage d'applications, la visualisation de données, le contrôle de matériel, la simulation, et le traitement des signaux. Les EPVs sont souvent catégorisé par les domaines d'application, tel que la conception logique, la simulation, et le traitement d'image. Cependant du point de vue langages de programmation, ils peuvent également être catégorisés selon les rôles de la programmation ou la fonctionnalité [Fost95]. Par exemple, considérant les différences entre la programmation visuelle, la visualisation de programme, et la visualisation des données. Quelques EPVs fournissent chacune des trois fonctionnalités, alors que d'autres se concentrent principalement sur un aspect.

2.2. Classifications

Il existe beaucoup d'approches différentes de programmation visuelle, vraisemblablement à cause de la grande variété des systèmes existants dans ce domaine. Nous présentons pour cela deux classifications. La première propose un découpage de l'ensemble de processus de programmation (classification de Nan C.Shu), alors que la seconde ne s'attache qu'aux langages visuels (classification de Shi-Kuo Chang).

2.2.1. La classification de Shu

Selon Shu [shu88][Shu89], l'évolution de la programmation visuelle se fait selon deux axes. Le premier correspond à l'utilisation des techniques graphiques et des systèmes de désignation pour fournir un environnement visuel permettant de :
– créer et exécuter des programmes,
– retrouver et afficher des informations ou des données,
– concevoir et comprendre des logiciels.
Le second axe étant la conception des langages visuels afin de :
– traiter des informations visuelles (images),
– supporter des interactions visuelles,
– programmer avec des expressions visuelles.

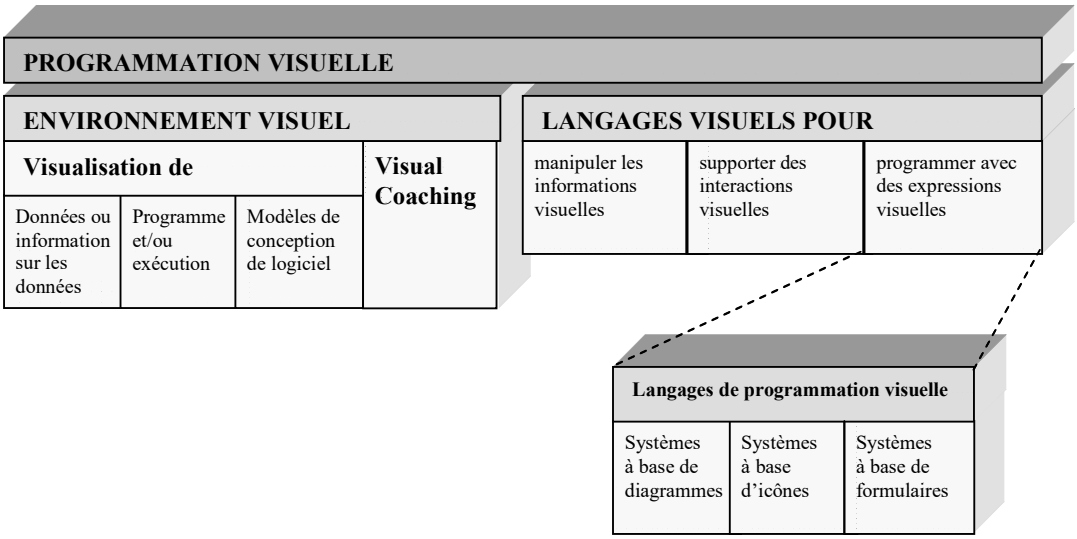

Figure 2.1 : Classification de Shu [shu88][Shu89]

2.2.1.1. Les environnements visuels

Dans le cas de la *visualisation de données ou d'informations sur les données*, nous parlons donc, d'un environnement visuel pour la représentation des informations. Cela signifie que les informations ou les données sont stockées dans des bases de données traditionnelles, mais présentées à l'utilisateur sous une surface graphique. On peut ensuite se déplacer sur cette surface, ou bien zoomer sur une partie pour obtenir plus de détails.
Un environnement visuel peut aussi être utilisé entant qu'un support graphique à la *visualisation des programmes*. On peut aussi représenter graphiquement certains états critiques et résultats. Par contre, les programmes sont écrits en langages textuels. L'objectif

est d'utiliser des représentations graphiques pour simplifier la tâche de développement et de test.

Dans le cas de la *visualisation des modèles de conception*, le but est de fournir un environnement de développement auquel l'ensemble du cycles de vie est présenté sous une forme graphique aux personnes qui conçoivent, utilisent ou maintiennent des logiciels.
On peut également utiliser la programmation visuelle pour réduire l'écart entre le processus mental de résolution du problème et la programmation effective. Dans de tels environnements, l'utilisateur peut voir des effets visuelles qui correspondent à ses instructions, au fur et à mesure de la construction du programme. En effet, il spécifie le programme en interagissant directement avec certaines données, et le système se contente de reproduire ce comportement avec d'autres données. L'utilisateur n'a pas à se soucier d'une syntaxe particulière. C'est pourquoi on utilise le terme *visual coaching* pour caractériser cette catégorie.

Avant d'aller plus loin, il faut remarquer que ces quatre catégories ont les caractéristiques suivantes en commun :
– Elles fournissent toutes un environnement visuel qui autorise des nouvelles voies en matière d'interaction homme-machine.
– Aucune ne peut être considérée comme une nouvelle approche au niveau langage. C'est cette seconde caractéristique qui permet de faire la distinction entre les deux principales composantes de la programmation visuelle, à savoir les environnements visuels et les langages visuels.

2.2.1.2. Les langages visuels

Les langages visuels peuvent être classifiés en trois catégories.
Dans la première catégorie, on a des *langages de manipulation des informations visuelles*. Typiquement, ce sont des langages textuels qui autorisent des références directes aux images. Il faut préciser que seule l'information manipulée est graphique, le langage lui reste textuel.

Dans la second, on utilise de plus en plus d'icônes ou d'objets graphiques comme moyen de communication avec l'ordinateur. Cependant, cette communication ne peut exister qu'à travers des logiciels capables de spécifier le comportement de ces représentations. Cela justifie le développement des langages capables de définir, créer et manipuler des icônes. Shu parle alors sur des *langages qui supportent des interactions visuelles*. Les langages de cette catégorie supportent, en général, des représentations et des interactions visuelles, mais le langage reste aussi textuel.

La dernière catégorie permet aux utilisateurs de programmer avec des expressions visuelles (multi-dimentionnelles). On les appelle les *langages de programmation visuelle*. Cette catégorie peut se subdiviser en :
– langages à base de graphes ou de diagrammes,
– langages à base d'icônes,
– langages à base de tableaux ou de formulaires.

2.2.1.2.1. Langages à base d'icônes (iconiques)

Pour les langages à base d'icônes, les icônes ou les images peuvent représenter soit les objets, soit la construction du programme.
Le langage de programmation visuelle HI-VISUAL utilise les icônes pour représenter des objets qui existent dans le monde réel (des objets manipulés par le programme). Il n'y a pas d'icône pour les fonctions ou les procédures. Au lieu de cela, les opérations sont spécifiées par une combinaison d'objets en les mettant les uns sur les autres.

2.2.1.2.2. Langages à base de diagrammes

Les langages diagrammatiques sont représentés par des graphes ou des figures géométriques. Les noeuds des graphes indiquent des objets et les arcs représentent la liaison entre ces objets. Les graphes conviennent aux langages basés sur le paradigme flot de données ou flot de contrôle [RAVA02], et ils sont largement utilisés. La sémantique des arcs qui relient les noeuds d'un graphe dépend du paradigme employé par le langage. Avec le paradigme de flot de données, les arcs transmettent les données à traiter. Prograph est un langage basé sur le flot de données.

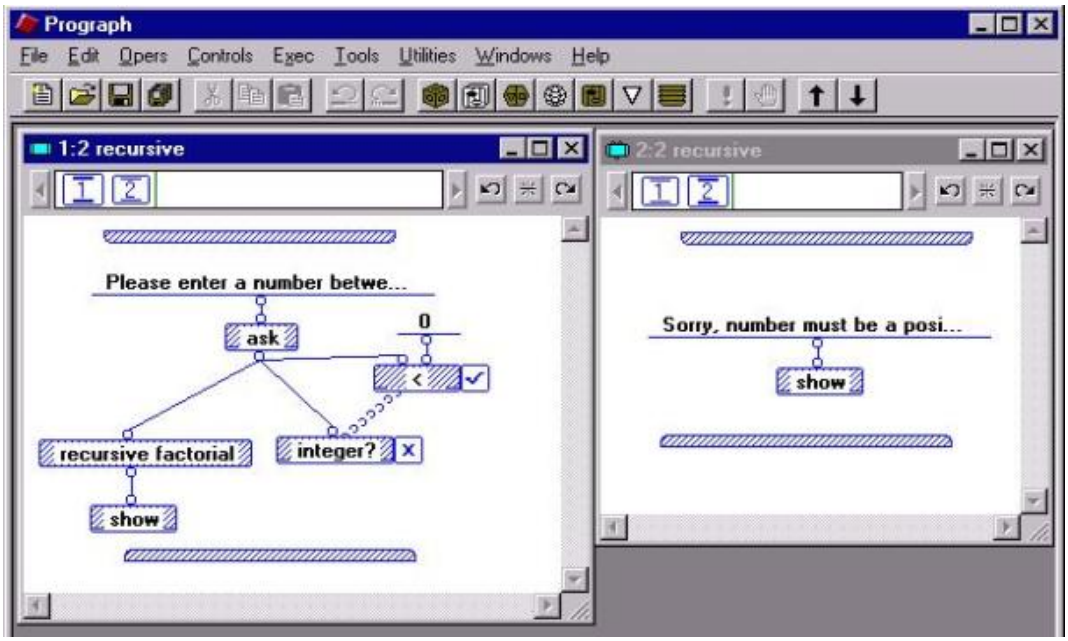

Figure 2.2 : Calcul récursif de factorielle en Prograph [RAVA02]

Pour les langages basés sur le paradigme flot de contrôle, les arcs portent des signaux de commande qui activent l'exécution des noeuds destinataires.
Pour les langages qui utilisent des figures géométrique (cercle, rectangle, triangle, ...), leurs sémantiques sont exprimées par l'intermédiaire de leurs propriétés géométriques ou de leurs relations spatiales topologiques.

2.2.1.2.3. Les langages à base de formulaires

Les langages à base de formulaires ou de tableurs représentent les programmes sous formes d'un ensemble de formulaires. Comme pour un tableur classique, les données sont représentées dans les cellules et les formulaires expriment les programmes (les calculs). Chaque cellule peut contenir une formule composée de constantes ou des références à d'autres cellules.

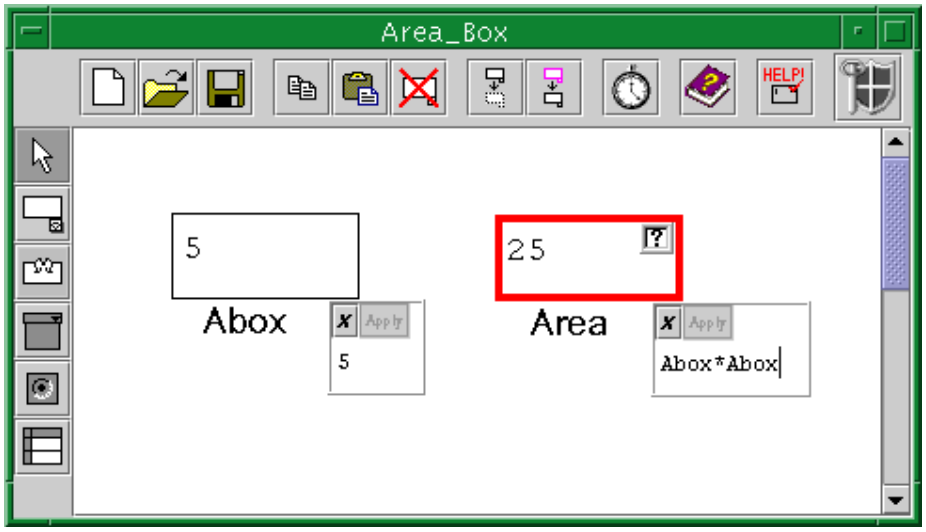

Figure 2.3 : Programme en Forms/3 [Burnett94b]

Forms/3 [Burnett94b] est un langage déclaratif de programmation visuelle qui utilise les métaphores de cellules et de formules des tableurs. Les programmes sont faits par la définition des formules sur une cellule qui peut être référencée par d'autres formules. Les cellules peuvent être organisées en groupe appelé 'formulaire', c'est un mécanisme de base d'abstraction de données. Un formulaire peut utiliser une image (graphisme). Le programmeur crée un nouveau programme dans Forms/3 en créant un nouveau formulaire, en lui ajoutant des cellules, et en indiquant les formules. La Figure 2.3 représente un programme qui calcul le carré d'un nombre.

2.2.2. Le découpage de Chang

Chang propose une classification légèrement différente des langages visuels, qui est basée sur deux critères : – la représentation des objets manipulés,
– les constructions du langage.

Dans [Cha87], il considère deux types d'objets : les objets logiques auxquels on rattache une représentation visuelle et les objets avec une représentation visuelle inhérente auxquels on rattache une interprétation logique. Pour ce qui est de la construction du programme, elle peut se faire de deux manières : linéaire ou spatiale. La combinaison de ces deux critères produit quatre catégories de langages visuels qui sont présentées dans la figure 2.4

Type de langage	**Objets manipulés**	**Constructions du langage**
Langages qui supportent l'interaction visuelle	Objets logiques avec représentation visuelle	Structures linéaires
Langages de programmation visuelle	Objets logiques avec représentation visuelle	Structures spatiales
Langages de traitement de l'information visuelle	Objets visuels avec une interprétation logique	Structures linéaires
Langages iconiques de traitement de l'information visuelle	Objets visuels avec une interprétation logique	Structures spatiales

Figure 2.4 : Classification des langages visuels selon Chang [Cha87]

Cette classification est compatible avec celle de Shu (présentée précédemment). Chang ajoute cependant une quatrième catégorie qui correspond aux langages visuels manipulant des objets visuels dans un domaine à deux dimensions. Il introduit aussi le concept d'icône généralisée qui selon lui est à la base des langages visuels. Une icône généralisée est soit une icône objet, soit une icône processus. Une icône objet se compose de deux parties notées (*Xm,Xi*). *Xm* est la partie logique (le sens) et *Xi* est la partie graphique (l'image). Certains objets (comme par exemple les structures de données) ont un sens logique, mais non graphique : (*Xm,ø*). En imposant une représentation visuelle, on transforme l'objet (*Xm,ø*) en (*Xm,X'i*), et ainsi, il peut être visualisé. De la même manière, on a des objets (comme des dessins) qui ont un aspect visuel inhérent, mais pas de partie logique : (*ø,Xi*). Et de façon similaire, on transforme l'objet (*ø,Xi*) en (*X'm,Xi*) en imposant un sens logique. Une icône processus a la même composition (à savoir une partie logique et une partie graphique). Par contre, elle ne représente plus un objet, mais une action, ou un processus calculatoire.

2.3. Avantages et inconvénients

2.3.1. Avantages

Citons quelques avantages de la programmation visuelle :

- Elle permet un raisonnement visuel: une partie importante des activités cognitives est accompagnée d'une 'visualisation' mentale; les langages visuels peuvent aider les utilisateurs à penser visuellement, par exemple en voyant des chemins plutôt qu'en les exprimant. Ils peuvent servir de moyen de communication pour le résultat d'un tel raisonnement.
- Elle offre une ouverture vers des utilisateurs moins spécialisés: les spécialistes d'un domaine applicatif ont rarement des compétences informatiques (ou du moins ce n'est pas leur travail de les acquérir); les techniques visuelles facilitent l'interaction homme-machine et aussi la création de programmes - tâche complexe réservée par tradition aux spécialistes informaticiens.
- Elle améliore la qualité de programmes: la programmation visuelle permet aux programmeurs expérimentés de se concentrer mieux à la résolution du problème qu'à l'écriture de programme.
- La programmation visuelle enlève aussi la barrière linguistique. Le langage visuel sera utile pour des personnes qui ont des problèmes avec l'anglais en général, puisque la plupart des langages textuels qui existent aujourd'hui tendent à être basés sur l'anglais.

Rendre la programmation plus accessible et améliorer l'exactitude et la vitesse avec lesquelles les gens accomplissent des tâches de programmation sont les buts spécifiques les plus communs des recherches dans le domaine de la programmation visuelle [Burnett99].

2.3.2. Inconvénients

Il ne faut pas, non plus, ignorer que la programmation visuelle présente quelques inconvénients qui pourraient, néanmoins, être résolus d'une façon ou une autre.

- Par rapport aux langages visuels, les langages textuels restent plus généraux et plus capables d'exprimer des abstractions. On peut 'discuter' de tout, mais on ne peut pas tout visualiser, par contre ce qu'on peut voir est mieux compris.
- Les langages visuels sont bien adaptés pour la description des objets manipulés mais ils ont plus de difficultés à exprimer le comportement. Ceci devrait être représenté par d'autres objets visuels abstraits.

- Il y a aussi des représentations graphiques qui dépendent de la culture. Il faut alors se mettre d'accord sur la façon de représenter les choses. La normalisation pourrait remédier à ce problème.
- La programmation visuelle a besoin d'un large espace sur écran pour créer un programme. Ceci peut être résolu partiellement par l'utilisation de défilement dans une fenêtre et/ou par diffèrent mécanisme d'abstraction comme la structure hiérarchique dans un programme. Mais la compréhension d'un programme peut être affectée par une visualisation partielle, car avec chaque élément invisible on perd aussi ses relations avec le reste des objets. La hiérarchisation couplée avec des représentations iconiques oblige à définir des représentations graphiques pour des objets de plus en plus généraux et abstraits. L'utilisation non seulement des icônes graphiques, mais également du texte, pour les noms des variables, les classes, ... permet aussi de résoudre ce problème. Les recherches visant à trouver de nouveaux moyens d'adresser le problème de l'espace d'écran sont ainsi de plus en plus utiles.

2.4. Langages visuels à domaines-spécifiques LVDS

Les chercheurs de la programmation visuelle ont œuvré pour étendre le champ d'application de la programmation visuelle par le développement des systèmes de programmation visuels orientés vers des domaines-spécifiques. Sous cette stratégie, l'addition de chaque nouveau domaine supporté a augmenté le nombre de projets qui pourraient être programmés visuellement. Un avantage supplémentaire était l'accessibilité améliorée, des utilisateurs pouvaient parfois employer ces nouveaux systèmes. Les développeurs de LPVs dans des domaines-spécifiques ont trouvé que cela fournit des manières d'écriture des programmes pour un domaine de problème particulier, ils ont éliminé plusieurs des inconvénients trouvés dans les approches traditionnelles, parce qu'ils ont utilisé des objets visuels (par exemple, des icônes et des menus) reflétant les besoins particuliers [Burnett99]. Cette approche a rapidement produit un certain nombre de succès dans la recherche et dans le marché. Aujourd'hui il y a des LPVs commerciaux et des EPVs disponibles pour de nombreux domaines.

Le défi original - pour concevoir des LPVs avec assez de puissance et adresser une variété augmentée de problèmes de programmation - est un domaine de recherche continu, qui a le but de continuer à améliorer les manières de l'utilisation de la programmation visuelle. Mais, les LPVs commerciaux avec les caractéristiques nécessaires pour la programmation d'usage universel ont émergé et sont employés pour produire des progiciels commerciaux; par exemple (Pictorius International's Prograph CPX).

2.5. Conclusion

La programmation visuelle; malgré le grand nombre de recherches correspondant; reste un domaine ouvert, car chaque domaine exige un langage visuel spécifique, et que les nouveautés dans chaque domaine doivent être traitées et prises en compte.

Nous avons cité que l'un des objectif de la programmation visuelle est d'assister les experts non programmeurs à construire des programme et de résoudre facilement les problèmes dans leurs domaines. Le domaine de la simulation constitue un domaine où la programmation visuelle avait un intérêt certain.

Dans le chapitre suivant nous allons parler sur la simulation en générale et donner les fondements de la simulation à évènement discret.

Résumé

ملخص

Depuis des années le domaine de modélisation et de simulation est devenu très intéressant dans l'analyse des systèmes complexes et la prévision des résultats pour les expériences difficile à effectuer.

Pour que la tâche de la programmation des modèles de simulation devienne plus facile et très rapide nous avons proposé une approche qui consiste à l'utilisation de la programmation visuelle pour la construction des modèles de simulation à évènements discrets en prenant en compte la dimension de groupe.

Pour cela nous avons choisi le formalisme DEVS (Discrete Event System Specification), qui est puissant est capable à représenter n'importe quel système réel. Notre travail se résume au développement d'un environnement qui permet à des utilisateurs non programmeurs de construire leurs modèles de simulation selon le formalisme DEVS d'une manière visuelle au lieu de la programmation textuelle, de plus cet environnement peut fonctionner en mode collaboratif, c'est-à-dire il permet à plusieurs utilisateur de travailler en collaboration via le réseau Internet afin de construire leur modèle de simulation.

La réalisation de ce travail a exigé l'emploi de deux techniques, la programmation orientée composants (JavaBeans) et le partage de données (boite à outils Java Shared Data Toolkit JSDT), pour assurer que le prototype développé permette la programmation visuelle et le travail collaboratif dans toutes les étapes d'un projet de simulation.

منذ سنوات أصبح مجال النمذجة و المحاكاة ذو أهمية كبيرة في تحليل الأنظمة المعقدة وكذلك توقع نتائج التجارب صعبة الإجراء.

ولجعل عملية برمجة النماذج الخاصة بالمحاكاة أسهل و أكثر سرعة قمنا باقتراح تصور لاستخدام البرمجة البصرية في إنشاء نماذج المحاكاة ذات الأحداث المنفصلة مع الأخذ في الحسبان بعد آخر و هو البعد الجماعي.

من أجل هذا قمنا باختيار شكلية دافز "DEVS" والتي تعني وصف نظام ذو أحداث منفصلة، وهي فعالة و ذات قدرة على تمثيل أي نظام واقعي حقيقي. إن العمل الذي قمنا به يتمثل في تطوير نظام يمكن مستخدمين غير متخصصين في البرمجة من إنشاء ما يشاءون من نماذج تبعا شكلية دافز بطريقة مرئية بدل البرمجة النصية، بالإضافة إلى ذلك هذا النظام يمكن من العمل بشكل جماعي تعاوني، أي أنه يسمح مجموعة من المستخدمين بالعمل متعاونين عن طريق شبكة الانترنيت لإنشاء نموذجهم المشترك لمحاكاة.

قد استدعى منا هذا العمل توظيف تقنيتين، البرمجة القائمة على المكونات (جافا بينز) و تقاسم البيانات(علبة أدوات جافا لبيانات المتقاسمة)، و هذا نضمن سماح النظام بالبرمجة المرئية وكذا العمل التعاوني خلال جميع مراحل أي مشروع لمحاكاة.

Table des matières

Table des figures

Bibliographie

[Acims] Arizona Center for Interactive Modeling and Simulation, web site : http://www.acims.arizona.edu.

[Balci97a] O.Balci,A.I.Bertelrud,C.M.Esterbrook,R.E.Nance. *Introduction to the Visual Simulation Environment.* Proceeding of the 1997 Winter simulation conference.1997.

[Balci97b] O.Balci,A.I.Bertelrud,C.M.Esterbrook,R.E.Nance. *The Visual Simulation Environment Technology Transfer* . Proceeding of the 1997 Winter simulation conference.1997.

[Balci97c] O.Balci,A.I.Bertelrud,C.M.Esterbrook,R.E.Nance. *The Visual Simulation Environment.* Proceeding of the 11th European Simulation Multiconference.1997.

[Balci98] O.Balci,R.E.Nance. *A Taxonomy of Layout Composition Techniques for Visual Simulation.* Proceeding of the 1998 Summer simulation conference.1998.

[Bachelet98] B.Bachelet, *Creation of libraries of reusable model components for the visual simulation environment*, Verginia university, August 1998.

[Banks00] J.Banks, *Introduction to simulation*, Proceedings of the 2000 winter simulation conference, Atlanta, 2000.

[Bany00] Omid Banyasad.*A Visual Programming Environment for Autonomous Robots.*Thèse de Master université Dalhousie.2000.

[Batti] Site web de J.B.Philippi. http://www.batti.org.

[Belattar93] B.Belattar, *Modélisation et Simulation sur Ordinateur*, université de Batna, 2003.

[Ben-Atallah97] S.Ben Atallah, *Creation of libraries of reusable model components for the visual simulation environment*, Verginia university, August 1998.

[Borland] Site Web de Borland, http://www.borland.com/

[Burnett94a] Margaret M. Burnett, Adele Goldberg, Ted Lewis (1994). Visual Object - Oriented Programming. Printice-Hall/Manning.

[Burnett94b] Margaret M. Burnett and Marla J. Baker (1994). A Classification System For Visual Programming languages. *Technical Report* 93-60-14.

[Burnett99] Margaret M. Burnett. *Visual Programming.* In: Encyclopedia of Electrical and Electronics Engineering (John G. Webster, ed.), John Wiley & Sons Inc., New York, (to appear approx. 1999).

[Campos00] A.M.C.Campos, *Une architecture logicielle pour le développement de simulations visuelles et interactives individu-centrées : Application à la simulation d'écosystèmes età la simulation sur le Web,* Thèse de doctorat, Université de Blaise Pascal-Clermont II, Septembre 2000.

[Chang90] Shi-Kuo Chang. *Principles of Visual Programming Systems*. Prentice-Hall International Editions, 1990.

[Chang95] Chang, S. K., G. Costagliola, G Pacini, M. Tucci, G. Tortora, B. Yu, and J. S. Yu, "*Visual Language System for User Interfaces*," IEEE Software, pp. 33-44, March 1995.

[Cristea98] V.Cristea, A.M.Florea, *Collaboration in high-performance computing via Internet*, university of Bucharest, 1998.

[Dilley03] E.Dilley. *VISSIM for Refinery and Plant Process Control Engeneer.*Peterborough, UK.2003.

[Dong02] Lei Dong. Transforming Visual Programs into Java and Vice Versa. Thèse de Master université Dalhousie.2002.

[Fost95] Richard Fost. *Hight-Performance Visual Programming Environments: Goals and Considerations*. Special issue for ACM SIGGRAPH CG on MVE's.1995.

[Fukunary98a] M.Fukunary,Y.Chi,P.M.Wolfe. *JavaBeans-Based Simulation with a Dicision Making Bean.* Proceedings of the 1998 winter simulation conference.1998.

[Fukunary98b] M.Fukunary,Y.Chi,P.M.Wolfe. *Discrete Event Simulation with JavaBeans.* Arizona state university.1998.

[Ge98] Y.Ge. *Developpement of a Web-Based Simulation Environment using JavaBeans.* Master thesis. University of georgia.1998.

[Husman04] J.Husman, *Monitoring and debugging of collaborative simulation environment*, Master thesis, Stockholm university, 2004.

[JSound] Java Sound ,http://java.sun.com/products/java-media/sound/index.html/

[Knave03] E.Knave, E.Stackenland, *Distributed collaborative environment for simulation modelling and analysis*, KTD Sweden, March 2003.

[Lorcy00] S.Lorcy, *Infrastructure logicielle pour la gestion de la cohérence et de la qualité de service d'un environnement à objets reparti :Application au*

télétravail coopératif , thèse de doctorat, université de Rennes I, Janvier 2000.

[Mather03] J.Mather. THE DEVSJAVA SIMULATION VIEWER: A MODULAR GUI THAT VISUALIZES THE STRUCTURE AND BEHAVIOR OF HIERARCHICAL DEVS MODELS, Master thesis, University of Arizona, 2003.

[Miller98a] J.A.Miller.Y.Ge.J.Tao. *Component Based Simulation Environments : JSIM as a case study using JavaBeans.* Proceedings of the 1998 winter simulation conference.1998.

[Miller98b] J.A.Miller,A.F.Seila,X.Xiang. *The JSIM Web-Based Simulation Environment.* University of georgia.1998.

[Miroslav97] B.Miroslav. *JavaBeans Technology for Learning Discrete Simulation Methodes*. Ostrava university.1997.

[Myers90] Brad A. Myers. Taxonomies of visual programming and program visualization. *Journal of Visual languages and computing*, pages 97–123, 1990.

[Orca] Orca Computer. *Visual Simulation Environment*. http://www.orcacomputer.com/vse/VSESet.html.

[Owezarski96] P. Owezarski, *Conception et formalisation d'une application visioconférence coopérative application et extension pour la téléformation*, thèse de doctorat université de Toulouse III, 1996.

[Philippi03a] J.B.Philippi, M.Delhom, F.Bernardi. *The JDEVS Modelling and Simulation Environment.* Cosica university. 2003.

[Philippi03b] J.B.Philippi, P.Bisgambiglia. *JDEVS :An Implemntation of a DEVS Based Formal Framework for Environmental Modelling.* Cosica university. 08 August 2003.

[Philippi03c] J.B.Philippi. *Une Architecture Logicielle pour la Multi-Modélisation et la Simulation à Evènements Discrets de Systèmes Naturels Complexes.* Thèse de Doctorat. Université de Corse. 17 Décembre 2003.

[Praehofer98] H. Praehofer, J.Sametinger,A.Stritzinger. *Using JavaBeans to Teach Simulation and Using Simulation to Teach JavaBeans.* Linz university, Austria.April 1998.

[Praehofer99a] H. Praehofer, J.Sametinger,A.Stritzinger. *Discrete Event Simulation Using the JavaBeans Component Model.* Linz university, Austria.February 1999.

[Praehofer99b] H. Praehofer, J.Sametinger,A.Stritzinger. *Concepts and Architecture of a Simulation Framework Based on the JavaBeans Component Model.* Linz university, Austria.June 1999.

[Ptv-vis] Ptv-vision. *VISSIM State of the Art Multi-Modal Simulation*. http://www.ptv-vision.com.

[Rava02] H.R. Ravaosolo. *Intégration de flot de contrôle et de flot de données dans un langage de programmation visuelle*.Thèse de doctorat.université de Genève.2002.

[Renevier04] P.Renevier, *Systèmes mixtes collaboratifs sur supports mobiles : Conception et réalisation*, thèse de doctorat, université de Grenoble I, Juin 2004.

[Shu88] Nan C. Shu. *Visual programming*. Van Nostrand Reinhold, 1988.

[Shu89] Nan C. Shu. *Visual programming :Pespectives and Approaches.* IBM System Journal, Vol 28, No 4, 1989.

[Sun] Site Web de Sun MicroSystems, http://www.sun.com/

[Tarpin97] F.Tarpin-Bernard, *Travail coopératif synchrone assisté par ordinateur : Approche AMF-C*, thèse de doctorat, Ecole Centrale de Lyon, Juin 1997.

[Vapillon00] J.Vapillon, *Contribution à l'étude de la conversation dans le cadre du travail coopératif assisté par ordinateur (TCAO)*, thèse de doctorat université de Paris XI Orsay, Octobre 2000.

[Zhang97] Z.Zhang. *Java-Based Simulation and Animation Environmnet: JSIM's Foundation Library*. Master thesis. University of georgia. 1997.

[Zeigler95] B.P.Zeigler. *Object-Oriented Simulation with Hierarchical, Modular Models*. University of Arizona.1995.

[Zeigler98] B.P.Zeigler. *DEVS Theory of Quantized Systems*. University of Arizona.June 1998.

[Zeigler02] B.P.Zeigler,H.Sarjoughian. *DEVS Component Based M&S Framework : An introduction*. University of Arizona.2002.

[Zeigler04] B.P.Zeigler. *DEVS Today: Recent Advences in Descrete Event-Based Information Technology*. University of Arizona.2004.

[Zeigler05] B.P.Zeigler. *Introduction To DEVS Modeling and Simulation with Java: Developing component-Based Simulation Models*. University of Arizona. January 2005.

[Zhao97] H.Zhao. *A Graphical Designer for JSIM*. Master thesis. University of georgia. 1997.