Simulations et contrôle pédagogique : architectures logicielles réutilisables

Gloria Cortés Buitrago

UNIVERSITE JOSEPH FOURIER - GRENOBLE I
SCIENCES & GEOGRAPHIE

THESE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE JOSEPH FOURIER – GRENOBLE I

Discipline: INFORMATIQUE

Présentée et soutenue publiquement

par

Gloria CORTES BUITRAGO

le 15 octobre, 1999

Simulations et Contrôle Pédagogique : Architectures Logicielles Réutilisables

Composition du jury :

Président:	Marie-France Bruandet
Rapporteurs :	Isabelle Borne
	Brigitte de la Passardière
Examinateurs :	Rubby Casallas
Directeurs de thèse :	Viviane Guéraud
	Jean-Pierre Peyrin

THESE PREPAREE AU SEIN DU LABORATOIRE DE COMMUNICATION LANGAGIERE ET INTERACTION PERSONNE-SYSTEME - FEDERATION IMAG

A Jorge Mario, María Camila et le bébé

Remerciements

Je tiens à remercier tout d'abord Madame Marie-France Bruandet, Professeur à l'Université Joseph Fourier de Grenoble, qui a accepté de présider ce jury.

Je remercie très sincèrement Madame Isabelle Borne, Professeur à l'École des Mines de Nantes et Madame Brigitte de La Passardière, Maître de Conférences à l'Université Pierre et Marie Curie – Paris VI qui ont bien voulu accepter d'être rapporteurs de cette thèse.

Je remercie Madame Rubby Casallas, Maître de Conférences à l'Université des Andes, à Bogotá-Colombie qui a bien voulu être examinatrice de cette thèse et qui a toujours été très intéressée dans mon travail.

J'exprime toute ma reconnaissance à Monsieur Jean-Pierre Peyrin, Professeur à l'Université Joseph Fourier et à Madame Viviane Guéraud, Maître de Conférences à l'Université Stendhal qui ont dirigé cette thèse. Je remercie Jean-Pierre de m'avoir accueilli dans son équipe et pour toute la confiance qu'il m'a toujours accordé. Je remercie vivement Viviane pour sa présence, sa disponibilité, son écoute, sa patience, son soutien et son amitié.

Je remercie aussi les autres membres de l'équipe Arcade : Jean-Michel Cagnat, Jean-Philippe Pernin et Jean-Michel Adam. Je remercie tout particulièrement Jean-Michel Cagnat pour sa critique toujours constructive de mon travail, pour le temps qu'il m'a consacré, pour sa lecture et re-lecture ainsi que pour ses commentaires éclairés qui m'ont permis d'améliorer ce manuscrit et pour sa patience.

Je remercie le Centre Régional des Oeuvres Universitaires et Scolaires (CROUS) qui a assuré le financement de mes études doctorales en France et le Département d'Informatique de l'Université des Andes à Bogotá-Colombie pour son soutien.

Je voudrais aussi exprimer ma gratitude à tous les amis qu'ont été toujours présents : Claudia, Rubby, Koka, Bruno, Rodrigo, Mario, Pierre, Fernando, Frédérique, Sylvain, Béatrice, Agnès et Mohamed.

Je tiens à remercier mes parents et ma famille pour leur présence et l'affection qu'ils m'ont toujours témoigné.

Finalement, je remercie avec mon cœur à Jorge Mario et María Camila qui ont rendu possible la réalisation de ce rêve.

TABLE DE MATIERES

INTRODUCTION GENERALE

Les travaux de recherche de l'équipe ARCADE portent sur la production de logiciels pédagogiques fortement interactifs, basés sur la manipulation, la découverte et la simulation.

Dans un premier temps, ces travaux ont consisté à produire des applications de ce type, particulièrement dans le domaine de l'enseignement de l'informatique. Un ensemble de logiciels pédagogiques interactifs, qui permettent l'apprentissage par expérimentation et par découverte [GUE 93] est le principal résultat de cette étape. Cette expérience a conforté les idées de l'équipe sur le fait que ce type d'applications est un complément du processus de formation et qu'elles doivent être intégrées efficacement dans ce processus.

Une deuxième étape a été la définition de méthodes et d'outils pour la production de telles applications et l'équipe a concentré ses recherches sur un type d'environnement particulier : les simulations. Cette étape a été marquée par l'idée que l'intégration efficace des simulations dans le processus de formation, dans un milieu industriel ou dans un milieu académique, peut être réalisée seulement si les instructeurs et les enseignants disposent d'outils qui permettent une production rapide des applications, et ce à moindre coût. Les principaux résultats de cette étape ont été le modèle MARS qui définit un cadre méthodologique pour la conception de simulations pédagogiques, et des outils basés sur MARS : un outil général (1ère version : MELISA, 2ème version : OASIS) et un outil spécialisé GeneSimu.

Cette thèse se place dans la problématique de l'utilisation des outils de production de simulations pédagogiques par les enseignants et les formateurs.

J'ai réalisé un état de l'art des outils de production de simulations pédagogiques et j'ai expérimenté l'outil OASIS dans le cadre du projet européen ARIADNE. Ces études m'ont amené à formuler et tester des propositions sur deux aspects relativement indépendants: le contrôle pédagogique de simulations et le développement d'outils de production de simulations pédagogiques.

Pour l'aspect développement d'outils de production de simulations pédagogiques, nous avons constaté que les outils existants facilitent les tâches de conception et de production, mais que la distance cognitive entre les abstractions de l'auteur et les abstractions proposées par les outils est encore trop grande. L'idéal est un outil adaptable au domaine particulier de l'auteur. Essayer de construire cet outil est une utopie, néanmoins je propose une solution d'un point de vue génie logiciel, utilisant des techniques de réutilisation basée sur les architectures, les frameworks et les patrons. Pour cela, j'ai conçu et développé un framework basé sur MARS, qui permet au développeur de construire rapidement des outils adaptés. Ce framework a été utilisé pour construire une version d'OASIS, ainsi que pour construire FENIX, un outil pour la réalisation de simulations basées sur des modèles mathématiques simples.

Pour l'aspect contrôle pédagogique d'une simulation, j'ai constaté que, bien qu'il existe différentes approches dans les outils étudiés, il était possible de séparer la simulation et son contrôle pédagogique. Ceci permet d'utiliser différentes approches de contrôle pédagogique sur une même simulation. Pour cela, je propose ARGOS, une architecture pour faire communiquer une simulation avec un outil de contrôle pédagogique. Un sous-ensemble de ARGOS a servi à l'élaboration d'un prototype pour le contrôle pédagogique de simulations à distance.

La suite de cette thèse est organisée de la façon suivante:

La première partie présente le contexte du problème, l'état de l'art des outils de production de simulations pédagogiques, l'expérimentation des outils et les besoins identifiés.

Les deux parties suivantes sont indépendantes, et débouchent sur des propositions complémentaires.

La deuxième partie est consacrée à la proposition d'un framework pour le développement des outils de production de simulations pédagogiques.

La troisième partie présente ARGOS comme un modèle pour le couplage des outils de simulation et des applications de contrôle pédagogique.

Partie 1
Simulations et Contrôle Pédagogique

Cette partie a pour objectif de présenter le contexte du problème que nous avons abordé dans cette thèse, ainsi que les travaux et les expérimentations que nous avons réalisés.

Le chapitre 1 est consacré à la description de notre domaine : l'utilisation et le développement de simulations pédagogiques, et du cadre de ce travail. Il présente la démarche suivie dans cette thèse et situe nos contributions.

Dans les chapitres 2 et 3, nous présentons un état de l'art. Le chapitre 2 est consacré aux environnements pour la production de simulations pédagogiques, et le chapitre 3 se concentre sur l'aspect exercices dans ces environnements.

Le chapitre 4 présente les expérimentations réalisées avec les outils développés par notre équipe.

Tout au long de ces chapitres, nous repérerons la présence d'un certain nombre de besoins assez bien identifiés. Des réponses à ces besoins sont présentées dans les parties 2 et 3.

1 CONTEXTE ET CADRE DU TRAVAIL

Depuis l'apparition des simulations, la communauté enseignante considère que celles-ci peuvent modifier significativement l'enseignement et la formation. Le fait de pouvoir se poser des questions du type *Qu'est-ce qui se passe si…,* et d'agir ensuite sur une simulation pour trouver une réponse a une riche valeur pédagogique. Nous pouvons observer l'utilisation de simulations comme support au processus d'enseignement-apprentissage dans différents domaines : physique ([COR 88], [ESC 90], [HER 94], [SWA 96]), sciences sociales ([DIV 93], [MYC 96]), sciences économiques et gestion ([SAM 97], [WAR 99]), informatique ([GUE 91], [THO 90]), etc.

Notre équipe vise à fournir des méthodologies et des outils particulièrement adaptés à la production de simulations dont le but est l'apprentissage. Nous nous intéressons plus particulièrement aux aspects suivants:

- L'intégration des simulations dans les pratiques pédagogiques quotidiennes des enseignants et des formateurs. De notre point de vue, elle dépend en grande partie des outils que ces enseignants peuvent utiliser pour produire des simulations. Il existe des produits commerciaux ou librement diffusés qui sont parfois très spectaculaires, mais qui restent pédagogiquement limités; d'autres sont pédagogiquement très pertinents, mais ne sont pas des produits "finis".
- Les besoins liés à la production, rencontrés dans différents contextes d'utilisation (milieu académique ou industriel, communauté européenne,...) :

réduction des coûts, amélioration de la qualité des produits et facilitation de la réalisation.

- Les perspectives ouvertes par l'arrivée à maturité des nouvelles technologies éducatives, spécialement les multimédias et les réseaux. L'utilisation de simulations dans le contexte de la formation à distance nécessite aussi des outils pour spécifier et pour assurer une communication pédagogiquement efficace entre élèves et formateurs.

Il est important de bien distinguer les différentes catégories d'intervenants (cf.).

- Les concepteurs et développeurs d'outils de production analysent les besoins des auteurs de simulations pour leur fournir des outils adaptés. Ils sont concernés par la définition de méthodologies et de formalismes, par l'amélioration et la facilitation du processus de développement et par la conception d'interfaces homme-machine adéquates.
- Les auteurs de simulations spécifient et développent des simulations et les supports pédagogiques (informatisés ou non) nécessaires pour leur utilisation effective dans un environnement d'apprentissage. Ils sont concernés par la sélection des problèmes pédagogiques nécessitant le support d'une simulation, par la conception des simulations, par la définition d'exercices utilisant la simulation, et par l'utilisation des outils de création de simulations.
- Les enseignants et formateurs s'occupent de l'intégration des simulations dans un environnement d'apprentissage. Ils sont concernés par la sélection et l'adaptation des simulations et par la définition des conditions d'utilisation des simulations (travail autonome ou dirigé, présenciel ou à distance, individuel ou en groupe, etc.).
- Finalement, les apprenants utilisent les simulations dans leur processus d'apprentissage.

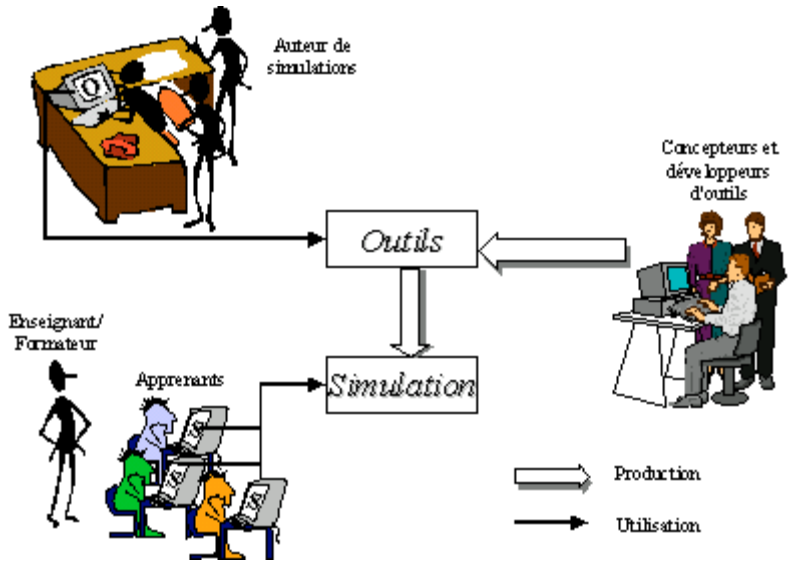


Figure 1-1: Acteurs dans la production et l'utilisation de simulations en éducation

Je vais présenter dans la suite comment les simulations sont intégrées dans les environnements d'apprentissage et les principaux problèmes liés à cette intégration. Je décrirai les propositions faites par notre équipe. Je situerai finalement ma démarche et les contributions de mon travail.

1.1 Les simulations dans les environnements d'apprentissage

Le mot simuler signifie imiter ou feindre. Dans le domaine de l'informatique, une *simulation est un programme qui imite le comportement d'un système réel ou abstrait* [APR 92].

De nos jours, nous ne pouvons pas douter du potentiel éducatif des simulations. De nombreuses raisons pratiques justifient l'utilisation de simulations pour l'enseignement et l'apprentissage [DEJ 91], [DEJ 94], [GAT 93], [HER 94], [PER 96]. Les simulations permettent de :

- Faire des expériences qu'il serait difficile d'effectuer en laboratoire, pour des raisons de sécurité ou de coût.
- Accélérer ou ralentir le temps pour mieux observer les phénomènes.
- "Donner" à chaque apprenant sa propre copie de l'appareil ou du système simulé.

- Travailler sur une réduction du monde réel offrant une vue "idéale" (par exemple un monde sans force de frottement).

D'autre part, un argument pédagogique est fréquemment avancé par les chercheurs [CEC 88], [DEJ 91], [DEJ 94], [DOU 95], [GAL 92], [HER 94], [PIL 96] : les simulations permettent l'application des théories d'apprentissage basées sur la découverte et l'exploration. Elles donnent la possibilité d'agir sur le concept à apprendre, à travers la manipulation, pour observer et analyser ses comportements et ses caractéristiques.

Les simulations peuvent jouer différents rôles dans le processus d'enseignement-apprentissage. Elles peuvent servir comme illustration pour motiver l'élève. Elles peuvent être l'élément fondamental pour l'apprentissage, et dans ce cas c'est à travers l'interaction avec la simulation que l'élève apprend. Elles peuvent être utilisées pour consolider les connaissances en confrontant l'élève à des situations diverses.

J.P. Pernin identifie trois contextes d'utilisation des simulations [PER 96, p. 45-46] :

- Le contexte traditionnel : le pédagogue se sert de la simulation pour faire des démonstrations ou propose des activités autour de la simulation. Il est présent pour guider et aider les apprenants.
- L'utilisation autonome pour une auto-formation ou une auto-évaluation.
- L'utilisation coopérative : un groupe d'apprenants réalise un travail coopératif s'appuyant sur la simulation.
- Ces trois contextes peuvent se présenter aussi bien dans le cadre d'une formation traditionnelle que dans le cadre d'une formation à distance.

Bien que les simulations soient pédagogiquement valorisées, utilisables dans différents contextes et dans différents cadres de formation, leur efficacité dans l'enseignement n'est pas encore démontrée [DEJ 94]. Il existe un consensus pour affirmer que cette efficacité dépend des objectifs pédagogiques fixés, des activités demandées à l'apprenant et des orientations données [DEJ 96], [PER 96], [HER 94], [GAL 92], [TOW 95], [VIV 91].

L'utilisation libre d'une simulation par l'apprenant ne garantit pas l'apprentissage. Comme pour les travaux pratiques en laboratoire, il est nécessaire de donner à l'élève des buts concrets, comme par exemple prouver une hypothèse, comparer des résultats théoriques

avec des résultats expérimentaux, étudier les relations existant entre les variables du système, mettre le système dans des conditions spécifiques, résoudre un problème, etc. De plus, une simulation doit permettre de guider, si nécessaire, l'apprenant vers le but qui lui a été fixé, comme le fait un formateur présent durant une expérience de laboratoire.

Dans un environnement d'enseignement traditionnel, le formateur peut fixer les objectifs et orienter le travail de l'apprenant sur la simulation [GUE 91], [PER 96]. Par contre, dans l'utilisation autonome d'une simulation (travaux pratiques en libre service, éducation à distance ou formation continue), le système devrait être capable de proposer les objectifs, de guider l'apprenant vers les objectifs et de vérifier si les objectifs sont atteints.

Le besoin de pouvoir contrôler pédagogiquement l'activité des apprenants sur une simulation a conduit plusieurs équipes de recherche à étudier les types de supports pédagogiques possibles et les outils nécessaires pour les auteurs et pour les apprenants (cf. chapitre 3).

Nous allons maintenant définir le terme simulation pédagogique que nous utiliserons tout le long de ce travail. Tout d'abord, une simulation pédagogique est une simulation et non une simple animation. La simulation, au contraire de l'animation, permet de changer le comportement du système simulé selon les valeurs fixées par l'utilisateur. Le but principal d'une simulation pédagogique est l'apprentissage; de ce fait la modélisation du système peut être simplifiée ou exagérée pour faire ressortir les aspects pédagogiquement les plus intéressants. Nous écartons ainsi les simulations "pleine échelle" qui sortent du cadre de notre travail. Une simulation pédagogique comporte en elle-même un contrôle pédagogique des activités de l'apprenant. Nous arrivons ainsi à la définition suivante :

> *Une **simulation pédagogique** est un programme qui imite **partiellement** le comportement d'un système réel ou abstrait, et qui inclut un certain **contrôle pédagogique** de l'apprenant.*

1.2 La production de simulations pédagogiques

Le développement de simulations pédagogiques de qualité n'est pas un travail simple. Nos expérimentations nous ont permis d'identifier deux contextes de production différents. Le premier correspond à la production commerciale de simulations pédagogiques, normalement effectuée par des entreprises spécialisées dans le développement de logiciels.

Le deuxième correspond à la production de simulations par des enseignants ou formateurs experts dans leur discipline, mais non nécessairement spécialistes de la programmation.

Notre équipe a réalisé une étude au sein de la société CORYS[1] afin d'identifier les principales caractéristiques du premier contexte [PER 96]. Nous avons constaté que la production était effectuée par une équipe pluridisciplinaire incluant un expert du domaine, capable de modéliser le système à simuler, un expert pédagogue chargé de la définition de la progression pédagogique la mieux adaptée au domaine, un scénariste interactif pour assurer la meilleure utilisation des ressources technologiques et d'interaction, des graphistes pour l'élaboration des interfaces et des informaticiens pour le développement de l'application. Les outils utilisés regroupent des langages de programmation comme C++, des logiciels spécialisés pour le développement de simulations (non pédagogiques[2]), et des outils de production d'applications multimédias. Les principaux problèmes pour pouvoir développer les simulations dans les temps fixés et aux coûts annoncés, identifiés par J.P. Pernin dans cette étude, sont la nécessité d'une coordination entre les différents experts et l'absence d'environnements adaptés au processus de développement spécifique aux simulations pédagogiques.

Le deuxième contexte, la production par les enseignants ou par les formateurs, est celui normalement rencontré dans les milieux académiques (écoles, lycées et universités) et dans le milieu industriel (centres de formation des entreprises). La différence principale avec le premier est que le développement des simulations pédagogiques est fait par des instructeurs ou par des enseignants, et non plus par une équipe de développement. Les outils de réalisation dépendent des connaissances du développeur. Les plus utilisés sont des systèmes de production d'applications hypermédias tels que ToolBook, Visual Basic et Authorware. Le problème principal de ce contexte est la réduction du temps de développement. Les formateurs et les enseignants voudraient pouvoir répondre rapidement aux difficultés pédagogiques qu'ils rencontrent au jour le jour, en réalisant ou en adaptant une simulation pour bien cibler leur problème pédagogique et pour expérimenter diverses

---

[1] La société CORYS (Grenoble) est spécialisée dans la conception de systèmes de simulation pour les métiers de l'énergie et du transport, incluant des simulations pour la formation.

[2] Car ne disposant pas de contrôle pédagogique.

approches. Le développement de simulations pédagogiques avec des logiciels hypermédias nécessite une maîtrise de l'outil par l'auteur, dans certains cas des connaissances de programmation, et surtout du temps pour le développement.

L'analyse faite par J.P. Pernin de ces deux contextes a révélé le besoin de disposer de méthodologies et d'outils adaptés à la réalisation de simulations pédagogiques. Les travaux de notre équipe ont porté ces dernières années sur la définition d'un modèle de conception et d'une méthodologie pour le développement de simulations pédagogiques. Nous avons également développé des outils de support à la méthodologie. Nous allons présenter brièvement le modèle proposé et les outils construits.

1.3 MARS et ses outils

Proposé par J.P. Pernin en 1995, le modèle MARS catégorise les différents aspects de la réalisation d'une simulation pédagogique :

- **M**odèle : définition du modèle de la simulation
- **A**ssociations : définition des liens entre les différents composants
- **R**eprésentation : définition de l'interface de la simulation
- **S**cénario : définition de la mise en scène pédagogique

MARS propose aussi des formalismes pour la représentation de chaque composant de la simulation : le modèle, l'interface ou la présentation, les associations et les scénarios (cf. chapitre 2 et chapitre 3).

Sur ce modèle, notre équipe a construit deux versions d'un environnement de production, MELISA puis OASIS, qui fournissent des outils spécialisés pour chaque tâche et chaque formalisme proposé par MARS. Ces environnements favorisent une méthodologie de production des simulations pédagogiques.

1.4 Démarche et contributions de ce travail

La démarche et les contributions de mon travail sont schématisées dans la Figure 1-2 et résumées ci-dessous.

Parallèlement au travail de notre équipe, d'autres équipes se sont intéressées à la même problématique et ont construit des systèmes qui poursuivent les mêmes objectifs [DEJ 94], [MUN 95], [ROS 94], [MAR 96]. Il apparaissait alors intéressant de comparer ces propositions. Un des objectifs de cette thèse a été de faire cette étude pour positionner nos propositions et nos outils face aux autres projets.

D'autre part, l'expérimentation auprès d'auteurs était nécessaire pour vérifier l'effectivité de nos méthodologies et de nos outils. Nous avons ainsi proposé et réalisé une expérimentation avec des enseignants du milieu académique.

L'étude des outils et l'expérimentation ont conduit à deux propositions qui sont les résultats principaux de la thèse :

- Bien que nos outils essaient de fournir des environnements appropriés au développement de simulations pédagogiques, les expérimentations réalisées nous ont permis de constater qu'il fallait rapprocher encore plus les outils et les auteurs. Ce besoin d'adaptation nous a amenée à la conception et au développement d'un framework qui permet de construire des outils-auteurs adaptés aux types des simulations à produire. Notre objectif essentiel est de fournir aux développeurs d'outils de réalisation de simulations pédagogiques une base logicielle réutilisable permettant de produire rapidement de nouveaux outils basés sur MARS.
- L'étude comparative des outils a révélé que chaque outil propose une approche différente pour la réalisation du contrôle pédagogique de l'apprenant, mais que la réalisation du contrôle pédagogique est conceptuellement indépendante de la simulation même. Les approches proposées, quoique différentes, ne sont pas incompatibles. Ceci nous a amenée à étudier la séparation entre la simulation et le contrôle pédagogique, avec l'objectif de permettre de greffer différents contrôles pédagogiques sur une même simulation. Le résultat de cette étude a été la proposition d'ARGOS, une architecture pour rendre indépendants le composant simulation et le composant pédagogique.

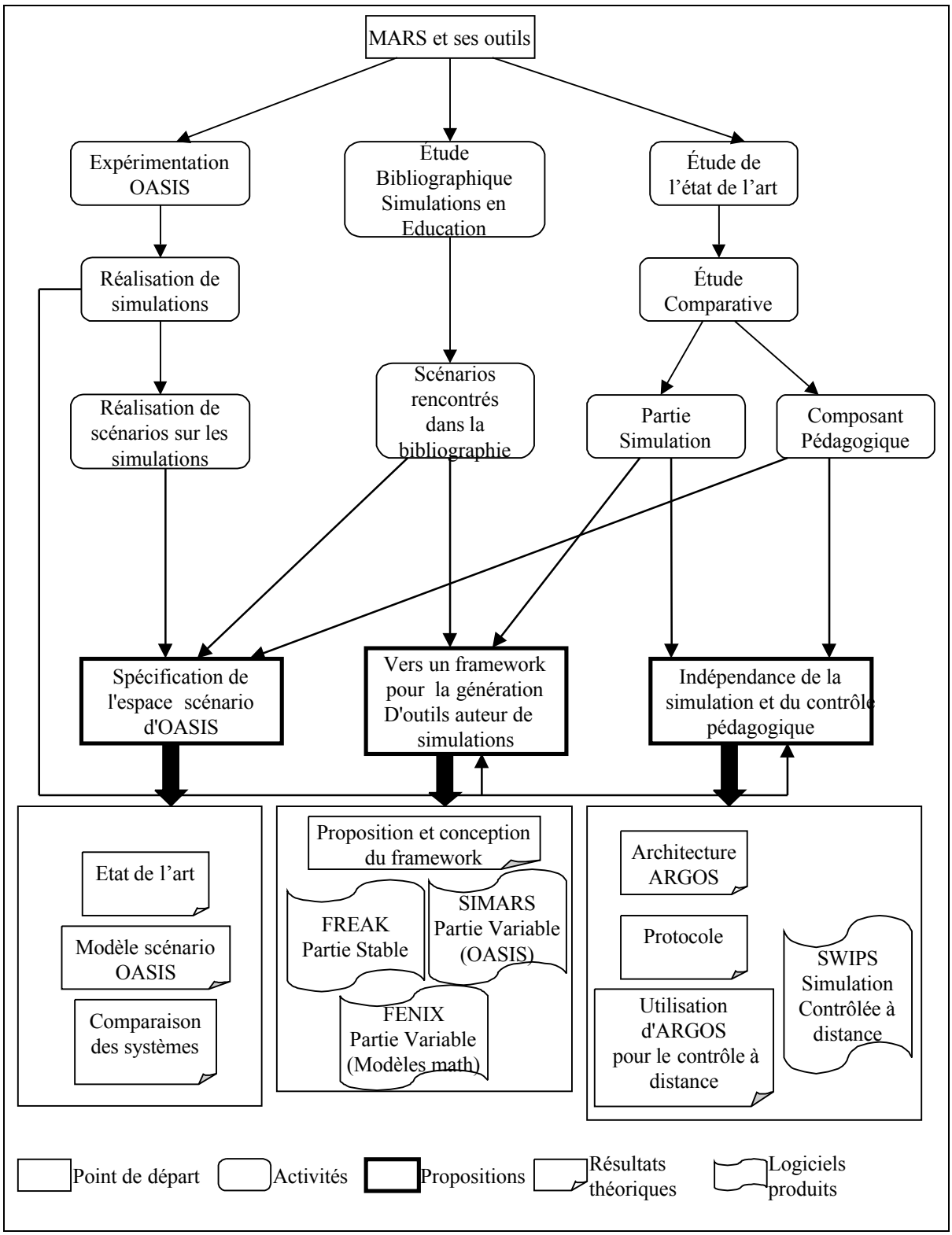


Figure 1-2 : Démarche et contributions de ce travail

2 LES ENVIRONNEMENTS POUR LE DEVELOPPEMENT DE SIMULATIONS PEDAGOGIQUES

Ce chapitre et le suivant sont consacrés à un état de l'art des environnements pour la réalisation de simulations pédagogiques. Cette étude a pris comme cadre d'analyse les concepts proposés par le modèle MARS. Le chapitre 2 est dédié à l'analyse du processus de développement associé à chaque outil et des espaces de travail fournis. Le chapitre 3 analyse l'aspect contrôle pédagogique.

Différentes équipes dans le monde ont entrepris des projets concernant la production de simulations pédagogiques. Elles ont ainsi souhaité répondre à un ensemble de besoins détectés dans ce domaine. Ces projets visent tous à diminuer l'effort de production (coûts, temps) et à favoriser la réutilisation des simulations dans différentes situations pédagogiques. Nous nous intéresserons au processus de développement qu'ils proposent à cet effet.

Les outils fournis présentent des caractéristiques similaires : profil des auteurs et domaines des simulations. Les domaines ciblés sont scientifiques et techniques: physique, chimie, électronique, assemblage et réparation de matériel, etc. Les auteurs visés sont essentiellement des enseignants et formateurs qui n'ont pas nécessairement de compétences en programmation; on parle alors d'environnements auteurs.

L'aboutissement récent de ces projets justifie la nécessité de cet état de l'art. Signalons qu'en 1991, un numéro spécial de la revue "Education & Computers" était dédié à la production de simulations pédagogiques. Il offrait un premier cadre de réflexion sur le domaine en proposant d'analyser séparément la conception du modèle, de l'interface et de la mise en scène pédagogique [ELS 91]. Souhaitant aujourd'hui analyser les environnements existants, nous utiliserons le modèle MARS et ses différents espaces de travail comme cadre de référence pour cette analyse. Précisons maintenant ce cadre.

2.1 MARS un cadre de référence

Le modèle MARS [PER 96] est avant tout un cadre conceptuel résultant d'une étude des processus de développement d'une entreprise de production de simulations pédagogiques. Il est basé sur la notion d'espace de travail, que J.P. Pernin définit comme:

> *Un environnement autonome dans lequel un ou plusieurs intervenants disposent de **formalismes**, de **données** et d'**outils**, leur permettant d'exprimer des **spécifications** et de produire un **résultat** évaluable.*

MARS identifie quatre espaces pour le développement d'une simulation pédagogique : l'espace **M**odèle, l'espace **A**ssociations, l'espace **R**eprésentation et l'espace **S**cénario. Chaque espace assiste l'auteur (ou les auteurs) dans un ensemble de tâches spécifiques à la réalisation d'un composant de la simulation :

- Dans l'espace **M**odèle, l'auteur produit un modèle du système à simuler, c'est-à-dire une description abstraite et partielle du système. L'auteur doit pouvoir structurer, organiser et manipuler l'information qui décrit le modèle.
- Dans l'espace **A**ssociations, l'auteur lie entre eux les éléments produits dans les autres espaces afin d'obtenir la simulation.
- Dans l'espace **R**eprésentation, l'auteur produit l'interface utilisateur de la simulation. Il doit pouvoir créer et manipuler les objets d'interface nécessaires pour visualiser le système simulé.
- Dans l'espace **S**cénario, l'auteur produit le composant pédagogique de la simulation. Il doit pouvoir définir les activités d'apprentissage proposées à l'apprenant sur la simulation, ainsi que le contrôle pédagogique effectué.

2.2 Présentation générale de systèmes existants

Nous voulons faire un inventaire et une analyse des formalismes, des données et des outils que différents environnements proposent pour chaque espace. Nous faisons d'abord une présentation générale en analysant les processus de développement proposés et en identifiant, si possible, les espaces de MARS (§2.2). Nous analyserons ensuite pour chaque espace les propositions faites, avec leurs avantages et leurs inconvénients (§2.3 à 2.5).

2.2.1 SAM (1992-1994)

Le projet SAM (*Simulation and Multimedia*) a été un projet du programme DELTA de la Commission Européenne. Son objectif principal était l'intégration d'outils pour construire des environnements d'apprentissage basés sur des simulations. La philosophie du projet a été d'intégrer dans un environnement auteur des outils existants pour développer des simulations, des outils pour construire des systèmes multimédias, des applications de bureautique (traitement de texte et tableurs) et des outils créés pour aider l'auteur à développer le composant pédagogique de la simulation.

Dans SAM, une simulation pédagogique a trois composants :

- Une simulation développée dans un environnement spécialisé comme LabView, ou avec un langage de programmation, ou encore avec des outils comme des tableurs.
- Des matériaux pédagogiques développés avec des outils spécialisés en production de systèmes multimédias comme Hypercard et Authorware, ou avec des traitements de texte ou des tableurs. Ces matériaux peuvent être des explications, des illustrations, des évaluations, des animations, etc.
- Un cours basé sur la simulation. Le cours est un ensemble d'objectifs d'apprentissage à atteindre par l'apprenant. Chaque objectif d'apprentissage peut être atteint par différents plans d'instruction. Un plan d'instruction est un ensemble d'objets d'instruction. Un objet d'instruction propose une activité pédagogique à l'apprenant, avec la simulation ou avec un des matériaux pédagogiques.

SAM propose des outils pour le développement des cours et le support nécessaire pour contrôler la simulation et utiliser les matériaux pédagogiques: un éditeur de cours, un éditeur de plans d'instruction et un éditeur d'objets d'instruction. Ces outils seront analysés en détail dans le chapitre 3. L'intégration d'autres outils avec SAM dépend seulement de leurs possibilités de communication à travers des mécanismes comme DDE* sur Windows (Data Dynamique Exchange) ou AppleEvents* sur les ordinateurs Macintosh. L'architecture proposée par SAM est décrite dans la Figure 2-1.

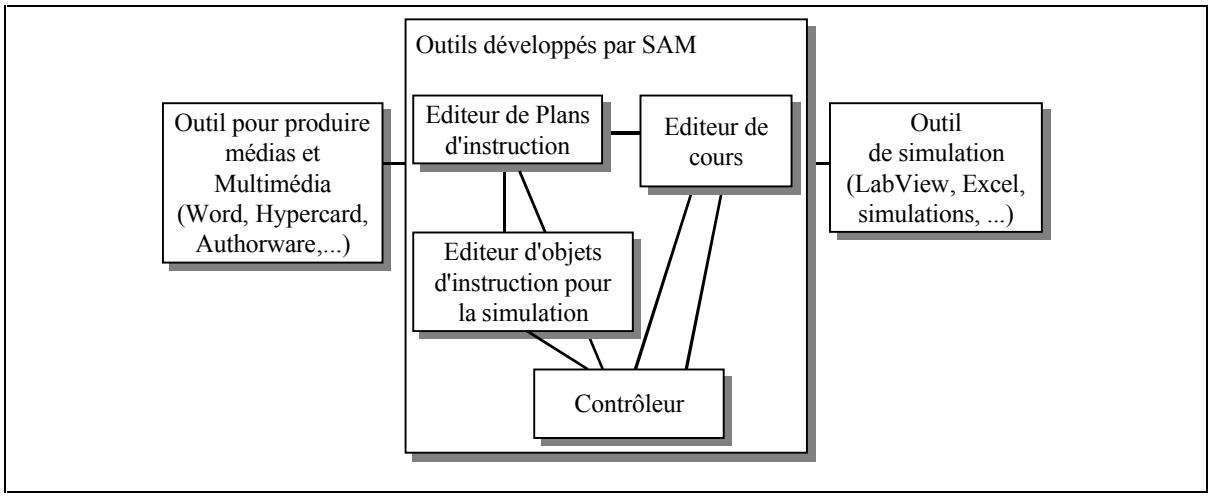


Figure 2-1 : Architecture proposée par SAM

Le processus de développement consiste à développer et à tester indépendamment la simulation et les matériaux pédagogiques pour les intégrer dans SAM, puis à construire les objets d'instruction avec la simulation, et finalement à tout intégrer dans un cours. Ce processus est illustré dans la Figure 2-2.

*Ces mécanismes étaient ceux existants à ce moment pour la communication entre applications sur les machines Windows et sur les machines Macintosh.

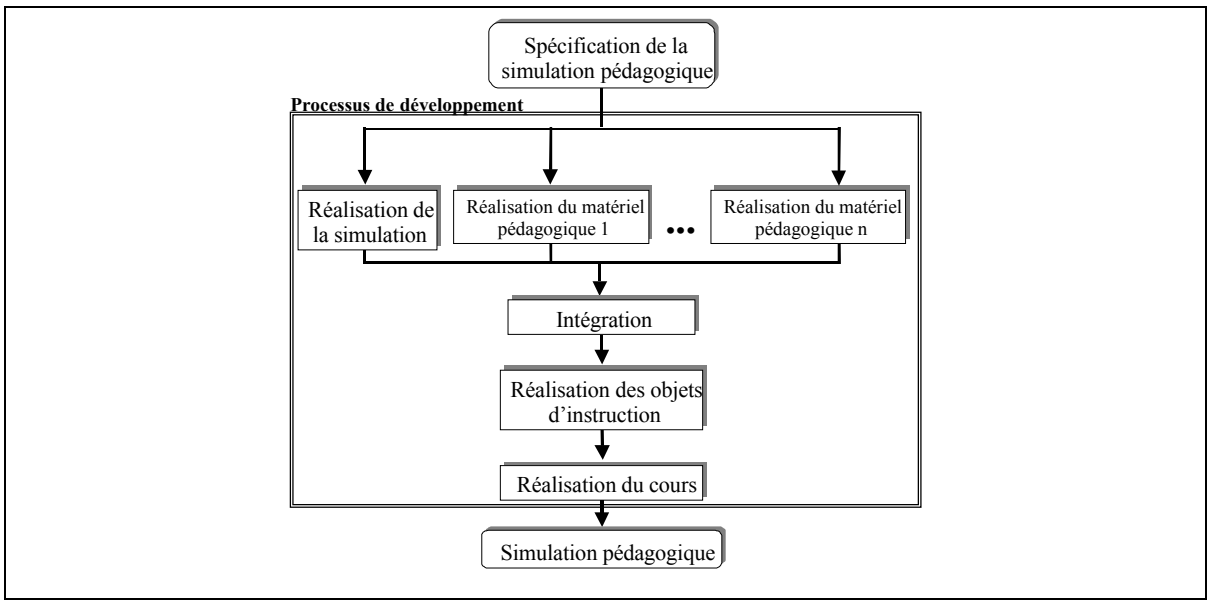


Figure 2-2: Processus de développement avec SAM

Nous repérons bien dans SAM l'espace scénario de MARS. Il est composé des outils pour élaborer les cours et des outils multimédias utilisés dans la réalisation des matériaux pédagogiques. L'espace modèle et l'espace représentation ne sont pas identifiables car SAM ne s'occupe pas de la réalisation de la simulation. En conséquence, les associations modèle–interface ne sont pas identifiables non plus. Les associations modèle-scénario sont définies explicitement au moment de l'intégration de la simulation.

Les références [ROS 92], [ROS 93] et [ROS 94] donnent davantage d'information sur le projet SAM.

2.2.2 RIDES – VIVIDS (1990-1999)

Le Laboratoire Amstrong de la USFA (*United States Air Force*) et le Laboratoire BTL (*Behavioral Technology Laboratories*) de l'université de la Californie du Sud travaillent depuis 1990 à la conception et au développement d'environnements d'apprentissage basés sur des simulations. L'objectif principal du projet est le développement d'outils permettant la production, rapide et à des coûts raisonnables, de tuteurs basés sur des simulations pour l'entraînement à la manipulation et à la maintenance de dispositifs complexes. Les environnements résultants sont RIDES (*Rapid Intelligent Tutoring System Development Shell*) et son successeur VIVIDS (*Virtual Interactive ITS Development Shell*). RIDES permet de construire des simulations pédagogiques en 2D. VIVIDS étend l'outil pour

obtenir des simulations pédagogiques en 3D. La base d'outils de construction étant la même dans les deux cas, nous décrivons ici seulement RIDES.

Une simulation pédagogique dans RIDES est composée de deux parties, la simulation et l'instruction associée :

- La simulation est un modèle graphique du système à simuler. Le modèle est composé d'un ensemble d'objets graphiques, chaque objet ayant un comportement défini.
- L'instruction est un ensemble de cours sur le système simulé. Un cours comporte une liste de leçons; chaque leçon donne une liste d'objectifs à atteindre (ou exercices). Certains types d'exercices peuvent être créés automatiquement.

L'auteur dispose d'un éditeur de simulation et d'un éditeur de leçon. Le processus de développement avec RIDES est montré dans la Figure 2-3. L'auteur crée d'abord les objets graphiques qui composent le système puis il définit leurs comportements. Une fois la simulation créée, l'auteur ajoute des connaissances sur le domaine (par exemple, le rôle d'un objet, ses fonctions,…). A ce moment, le système auteur peut générer automatiquement certains exercices (par exemple, des exercices qui demandent à l'apprenant d'indiquer les parties du système) ou permettre à l'auteur de créer d'autres exercices. Pour finir, l'auteur peut regrouper les exercices dans différentes leçons.

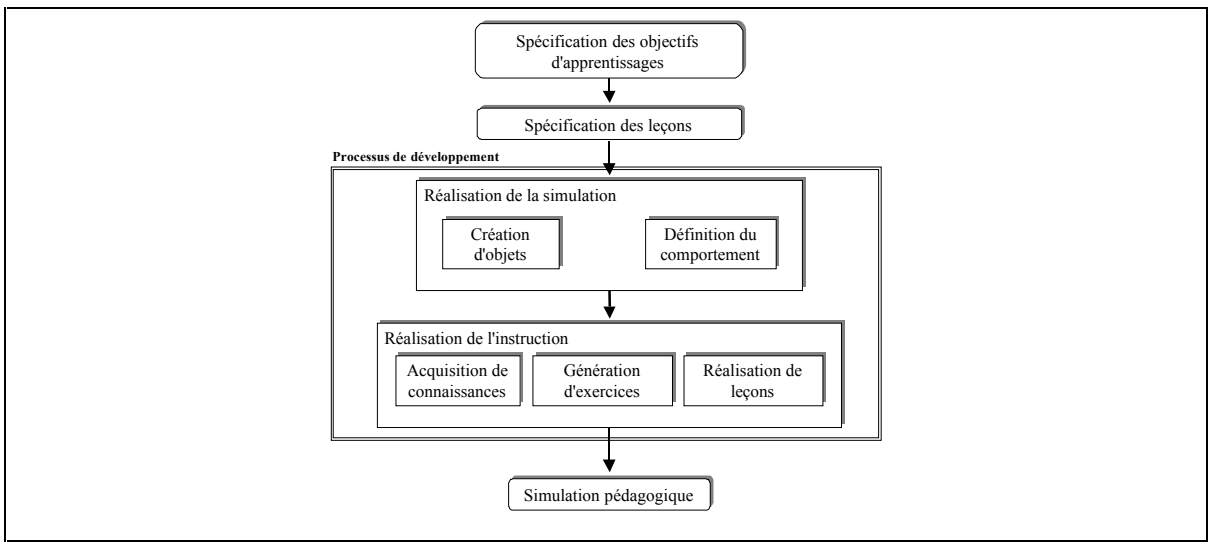


Figure 2-3: Processus de développement avec RIDES

Dans RIDES, les espaces modèle, représentation et associations (de MARS) ne peuvent pas être différenciés: ils sont tous trois contenus dans un seul espace de travail qui est l'espace simulation. De son côté, l'éditeur d'instruction correspond clairement à l'espace scénario de MARS.

Les références [MUN 93], [MUN 95], [MUN 95a], [FLE 96], [HOR 98], [TOW 95] contiennent des informations sur RIDES et sur des simulations produites.

2.2.3 SIM-BEST(1995-1997)

SIM-BEST (***Sim****ulation Computer* ***B****ased* ***E****ducational* ***S****upport* ***T****ools*) est un outil pour le développement de modèles de simulation et d'activités d'apprentissage avec ces modèles. Il a été développé par l'Université de Coimbra au Portugal.

L'objectif du projet est de fournir aux enseignants des outils pour construire des simulations pédagogiques, sans leur demander d'écrire des programmes et sans freiner leur créativité.

Dans SIM-BEST, une simulation pédagogique est constituée d'un modèle du système à simuler, d'un ensemble de scénarios éducatifs et d'une couche d'instruction (*instructional overlay*). Un scénario éducatif propose une interface permettant de manipuler les paramètres du modèle et de visualiser son comportement. La couche d'instruction est un logiciel éducatif créé avec le système auteur AIDA (développé aussi par l'Université de Coimbra), son intégration est planifiée mais pas encore publiée.

SIM-BEST est composé de trois éditeurs: un éditeur de modèles, un éditeur de scénarios éducatifs (interfaces) et un éditeur d'instruction (AIDA). Son architecture est présentée dans la Figure 2-4.

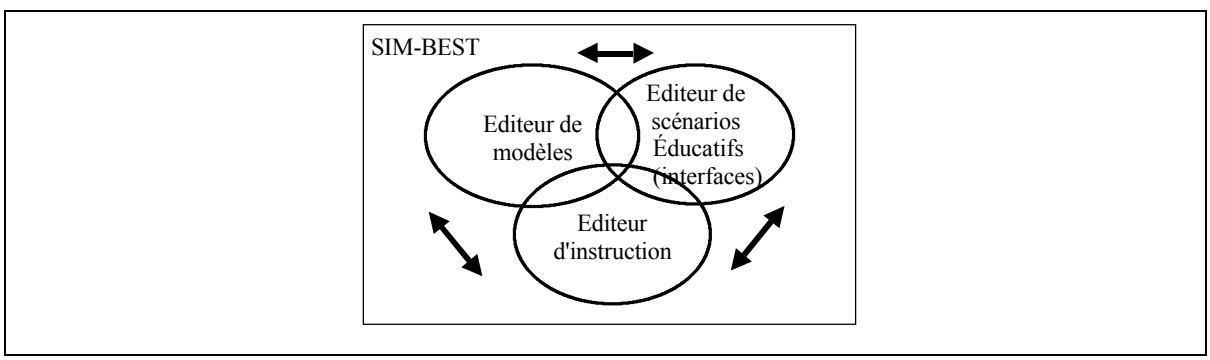


Figure 2-4: Architecture de SIM-BEST (D'après [MAR 97])

Le processus de développement proposé est montré dans la Figure 2-5. L'auteur construit d'abord le modèle, puis il construit les différents scénarios (interfaces) et finalement il intègre les scénarios dans une application pédagogique. SIM-BEST propose l'espace modèle, l'espace représentation et l'espace scénario de MARS à travers ses éditeurs de modèle, de scénario éducatif et d'instruction respectivement.

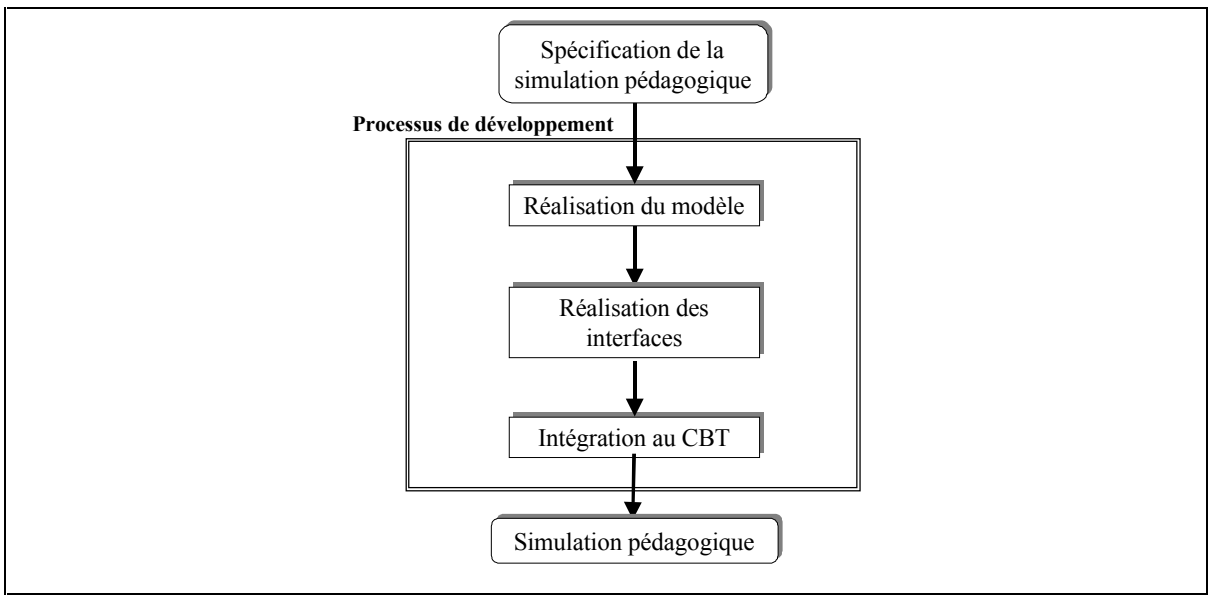


Figure 2-5: Processus de développement avec SIM-BEST

Les références [MAR 95], [MAR 96] et [MAR 97] décrivent plus précisément le projet SIM-BEST.

2.2.4 SERVIVE-SimQuest (1991-1998)

Le projet européen SERVIVE[3] (**Serv**ice for **I**ntegrated **V**irtual **E**nvironments) fait partie du programme "Telematic Applications". Il fait suite aux projets SIMULATE et SMISLE [DEJ 94] qui ont débuté en 1991.

Le projet vise essentiellement à:

- Améliorer la formation à travers l'apprentissage par découverte.
- Munir les auteurs d'un système auteur flexible.
- Favoriser l'échange de matériaux entre les participants du développement.

[3] SERVIVE est le projet ET1020 du Programme Télématique de la Commission Européenne.

- Aider l'auteur à concevoir la partie pédagogique de la simulation.

Dans l'environnement auteur proposé, SimQuest, une simulation pédagogique est constituée d'un ensemble d'éléments de trois types: modèles conceptuels, interfaces et objets d'instruction.

- Un modèle conceptuel est une description abstraite du système à simuler. Les modèles peuvent se différencier entre eux, par exemple par leur niveau de complexité ou par les algorithmes utilisés. A chaque modèle est associée une description en termes des noms des variables manipulées. Ces descriptions (*Model Wrapper*) sont utilisées par les éléments interface et objet d'instruction, pour consulter ou modifier les valeurs des variables du modèle.
- L'élément interface permet à l'utilisateur de la simulation (l'apprenant) d'interagir avec le modèle.
- Un objet d'instruction permet de définir une activité pédagogique que l'apprenant peut réaliser avec un modèle.

L'auteur dispose d'une bibliothèque d'éléments pour chaque type de composant et d'éditeurs pour instancier ces éléments ou pour créer de nouveaux éléments. La Figure 2-6 montre l'architecture globale de l'outil auteur SimQuest.

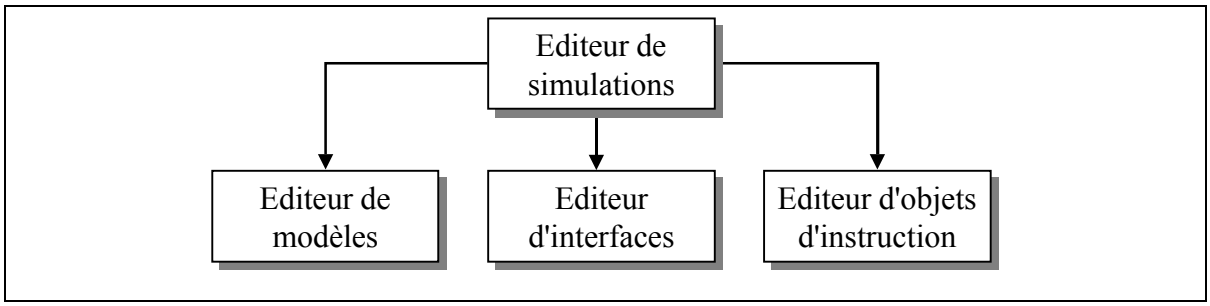


Figure 2-6: Architecture Globale de l'outil auteur SimQuest

Le processus de développement est donné dans la Figure 2-7. L'auteur peut commencer par réaliser le(s) modèle(s) avant de s'intéresser à l'interface et aux objets d'instruction. Il peut également se contenter de donner le nom des variables manipulées par le modèle (*Model Wrapper*) afin de réaliser l'interface et les objets d'instruction avant le modèle lui-même.

Les éditeurs de SimQuest correspondent aux espaces proposés par MARS. L'éditeur de modèles est l'espace Modèle, l'éditeur d'interfaces est l'espace Représentation et l'éditeur

d'objets d'instruction est l'espace Scénario. L'espace Associations est contenu implicitement dans les *Model Wrappers* qui définissent les noms que les interfaces et les objets d'instruction doivent utiliser pour consulter ou modifier les valeurs des variables du modèle.

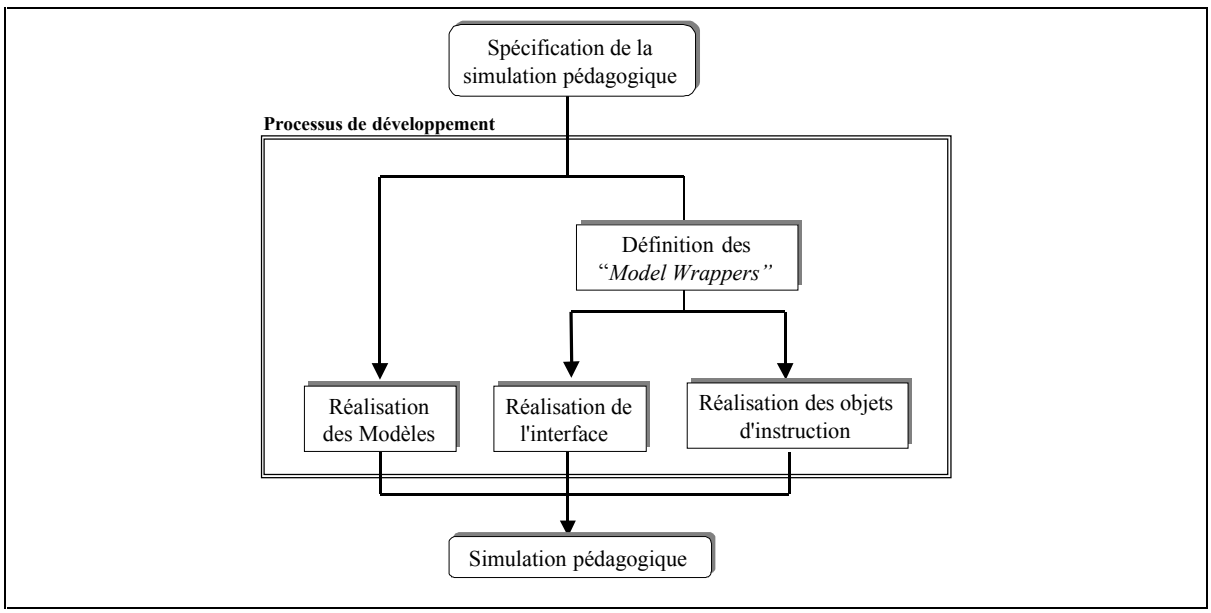


Figure 2-7: Processus de développement avec SimQuest

Les références [DEJ 94], [DEJ 96a], [DEJ 96b], [JOO 96] contiennent des descriptions détaillées de SimQuest et de simulations produites.

2.2.5 MELISA- OASIS (1994-1999)

L'outil OASIS (**O**util **A**uteur de **S**imulations **I**nteractives avec **S**cénarios) a été développé par notre équipe dans le cadre du projet ARIADNE du programme Telematic Applications de la Commission Européenne. L'objectif principal d'OASIS est de fournir, aux auteurs de simulations pédagogiques, des outils visuels permettant de suivre la démarche de production proposée par le modèle MARS.

OASIS constitue de fait une 2ème version du logiciel MELISA développé par l'équipe ARCADE dans le cadre d'un contrat avec le T.P.E.C. (*Technical Planning and Education Center*) de Hewlett-Packard avec le même genre d'objectif. Les interfaces, et tout particulièrement celles du contrôle pédagogique, ont été remaniées après les premières expérimentations; son code interne a également été repris et optimisé.

Le processus de production proposé par OASIS est décrit dans la Figure 2-8.

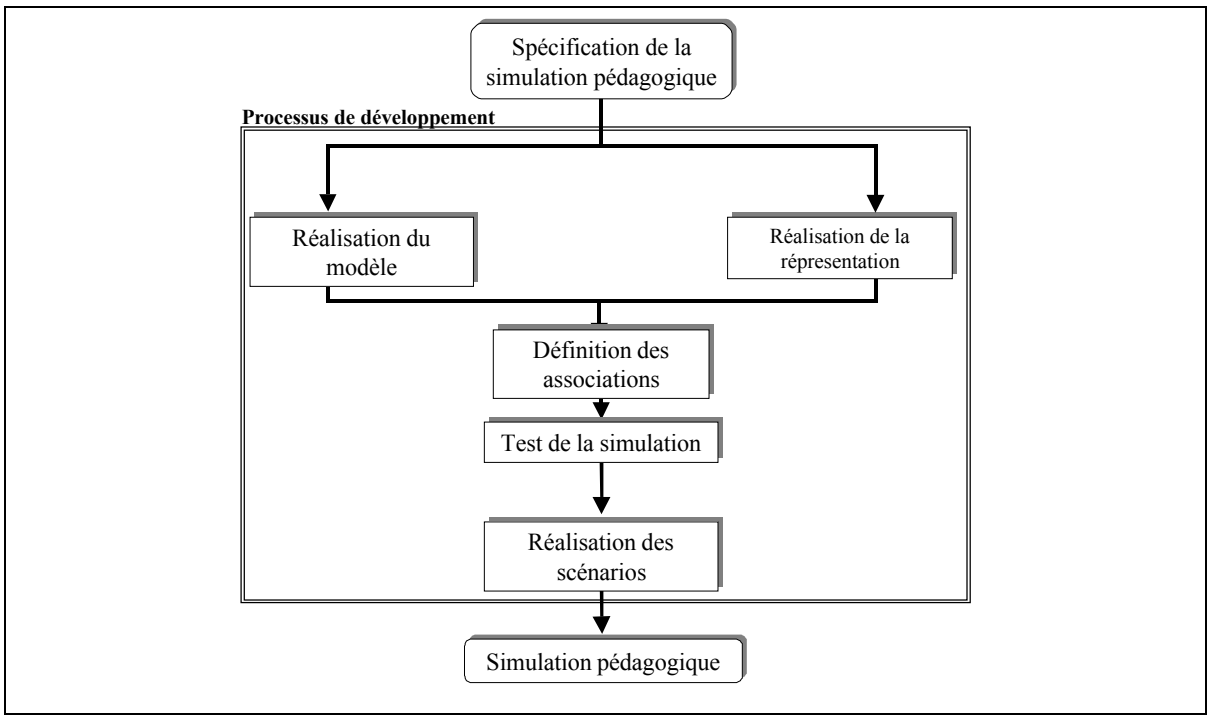


Figure 2-8: Processus de développement avec OASIS

Dans OASIS, une simulation pédagogique est constituée des quatre composants proposés par MARS: un modèle, une représentation, des associations et des scénarios. Les formalismes utilisés seront décrits dans les paragraphes 2.3, 2.4 et 2.5.

OASIS fournit un éditeur pour chaque espace de travail de MARS : un éditeur du modèle de la simulation, un éditeur d'objets de représentation, un éditeur d'associations et un éditeur de scénarios. L'architecture d'OASIS est décrite dans la Figure 2-9.

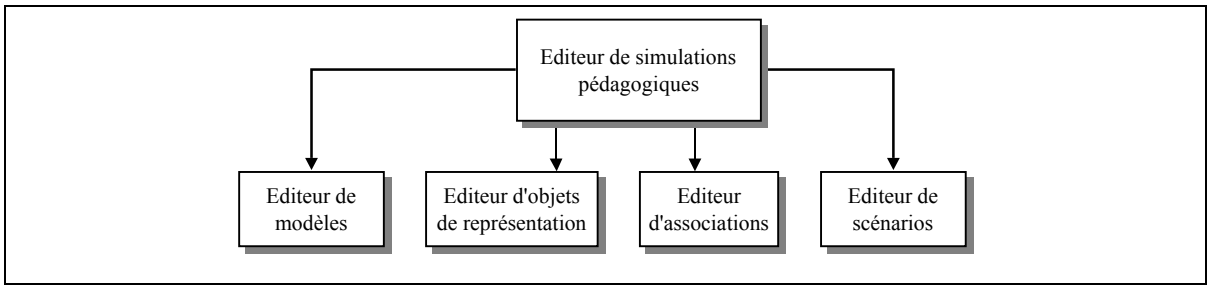


Figure 2-9: Architecture globale d'OASIS

Les références [PER 96], [CAG 97], [COR 97], [COR 97a], [COR 97b], [COR 98] et [COR 98a] décrivent en détail le modèle MARS et l'outil OASIS.

2.2.6 Conclusions sur les architectures et les processus de développement

Nous venons d'examiner les architectures et les processus de développement de simulations pédagogiques proposés par différents environnements. Il ressort de cette analyse que dans tous les processus de développement, la tâche de réalisation de la partie pédagogique est bien séparée de la construction de la simulation proprement dite. Il s'agit le plus souvent de réaliser la simulation avant de développer la partie pédagogique. Toutefois l'ordre de réalisation inverse est possible à condition que soient définis au préalable les points d'union entre les deux parties (cf. *Model wrapper* de SimQuest). La séparation de ces tâches dans le processus de développement se retrouve au niveau architecture, le composant d'édition de la partie pédagogique étant toujours un composant à part entière dans tous les outils.

Nous pouvons en conclure que ces deux tâches sont indépendantes au moment de la réalisation, mais dépendantes en ce qui concerne les informations.

2.3 L'espace modèle

L'objectif de l'espace modèle est d'assister l'auteur dans sa tâche de modélisation du système à simuler. Un modèle est une représentation abstraite d'un système et peut être exprimé de différentes façons. L'utilisation d'un formalisme permet de rendre le modèle explicite, par opposition aux modèles implicites présents dans un programme.

Les formalismes fournis conditionnent ce qui peut être réalisé avec l'outil: simulation de systèmes continus ou de systèmes discrets. Les deux grands types de formalismes proposés s'inspirent soit des principes de la programmation par objets, soit de la modélisation mathématique avec des systèmes d'équations différentielles.

SimQuest, par exemple, utilise comme formalisme des systèmes d'équations différentielles et algébriques et se limite à la simulation de systèmes continus. Son éditeur de modèles manipule des blocs fonctionnels: additionneurs, multiplicateurs, intégrateurs, …; chaque bloc a des entrées et des sorties, et les sorties d'un bloc peuvent être connectées aux entrées d'autres blocs. Un bloc peut être aussi un modèle construit avec SimQuest, et les modèles peuvent être stockés dans une bibliothèque. Cette caractéristique favorise la réutilisation. La Figure 2-10 montre un modèle fait avec SimQuest.

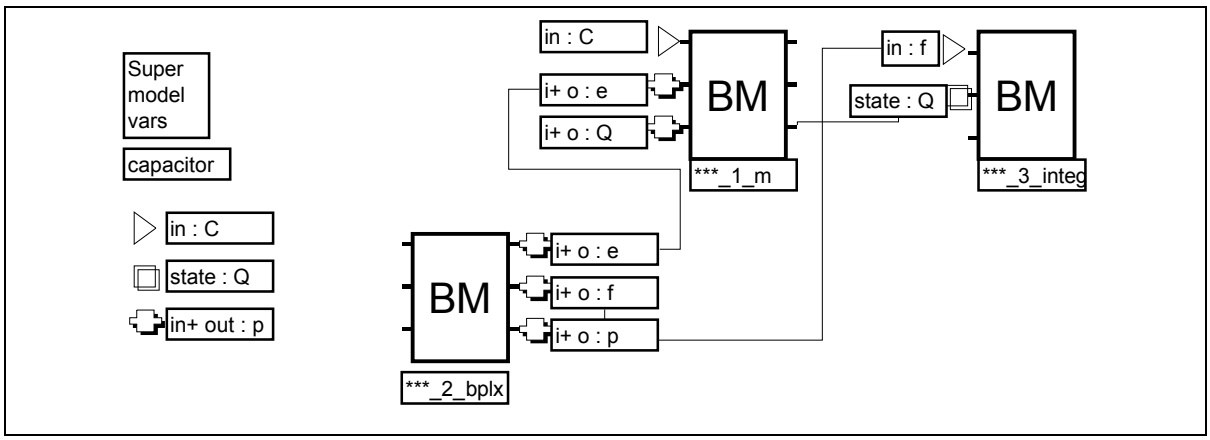


Figure 2-10: Exemple d'un modèle en SimQuest

SimBest utilise également pour les systèmes continus le formalisme des systèmes d'équations différentielles et algébriques. De plus, pour les systèmes discrets, SimBest propose un autre formalisme basé sur les files d'attente, les serveurs, les clients, les sémaphores, les activités et les convoyeurs. Les deux formalismes peuvent être combinés pour construire un modèle. L'éditeur de modèles, propose une interface basée sur la manipulation directe d'icônes pour définir les entités du modèle (quantités, dérivateurs, intégrateurs, serveurs, sémaphores, etc.) et les relations entre les entités. Une entité d'un modèle peut aussi être un modèle fait avec SimBest. La Figure 2-11 montre un exemple de modèle.

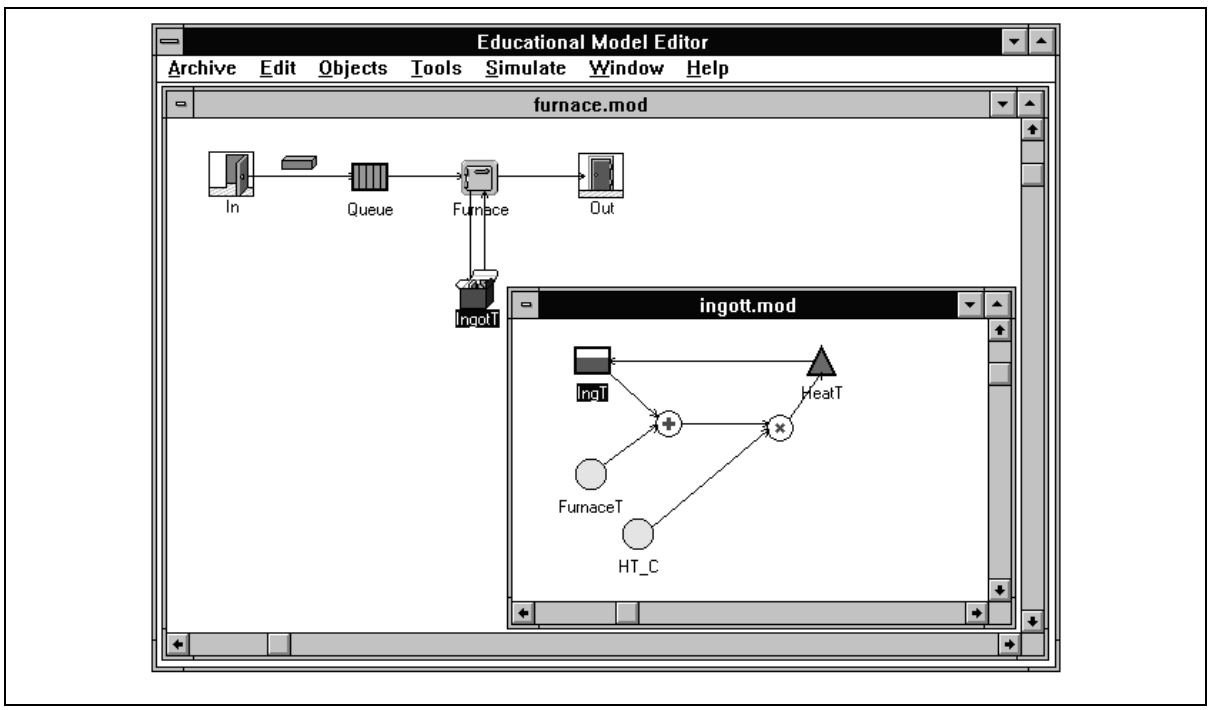


Figure 2-11: Exemple de modèle en SimBest (d'après [MAR 95])

Des outils, tels que OASIS et RIDES, inspirés par des langages de programmation, permettent de modéliser des systèmes continus et des systèmes discrets, en utilisant un seul formalisme. Une caractéristique commune de ces outils est qu'ils ont besoin de s'appuyer sur un langage de scripts pour définir certaines parties des modèles. Aucun outil de gestion de bibliothèques de modèles n'est actuellement disponible dans ces environnements-là.

Le formalisme d'OASIS est une combinaison d'objets et de Statecharts de Harel. Le modèle d'une simulation est défini avec des attributs, publics et privés, et des méthodes, publiques et privées, plus une dynamique exprimée avec les formalismes offerts par les Statecharts de Harel (événements, états, transitions, activités, actions, conditions etc.). Les activités, les actions et les méthodes sont écrites dans un langage de scripts. La Figure 2-12 montre les différentes parties de l'éditeur de modèles d'OASIS.

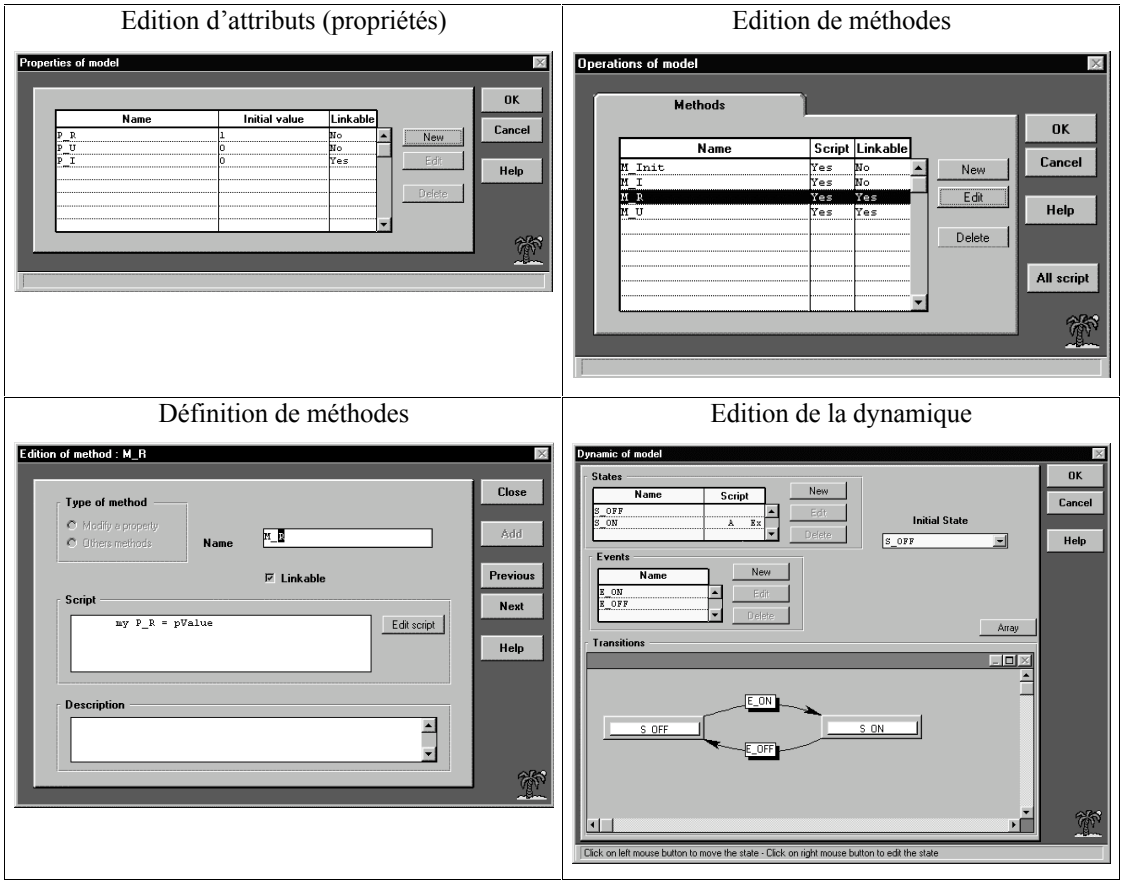


Figure 2-12: Edition de modèles en OASIS

RIDES utilise comme formalisme une combinaison d'objets et de programmation par contraintes. Le modèle du système est défini par les objets qui composent le système, leurs attributs, leur comportement et les relations entre eux. Le comportement et les relations des

objets sont définis par leur façon de répondre aux événements et par les contraintes dans les valeurs de leurs attributs. Les réactions aux événements et les contraintes sont exprimées dans un langage fourni par l'outil. La Figure 2-13 montre les interfaces de définition des attributs et des événements d'un objet.

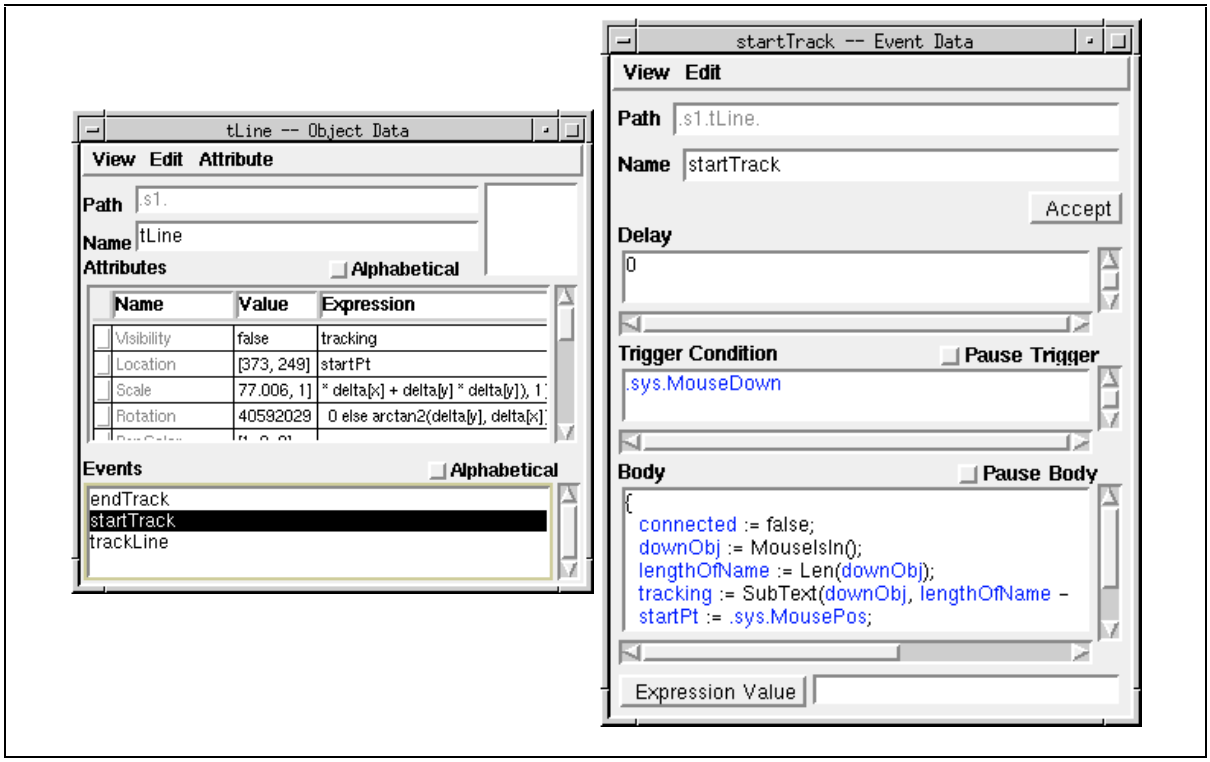


Figure 2-13: Interfaces de RIDES pour l'édition du modèle (d'après [Mun 95a])

Les formalismes pour la construction des modèles recherchent une généralité d'expression pour favoriser l'utilisation dans différents domaines. Comme nous le verrons dans le chapitre 4, cet avantage peut se convertir en inconvénient du point du vue de l'auteur, car un double processus d'abstraction est requis pour arriver au modèle. Et il faut souvent une participation de spécialistes pour arriver à de bons modèles. La solution parfois proposée est l'utilisation de bibliothèques de modèles spécialisées par domaine et alimentées par des experts en modélisation.

2.4 L'espace représentation

L'espace représentation des outils s'inspire de la troisième génération de systèmes de gestion d'interface utilisateur. L'objectif de ces systèmes est de fournir aux programmeurs des outils qui permettent la construction d'une interface de manière interactive, par la manipulation directe et par l'utilisation de bibliothèques d'objets d'interaction

(*widgets*)[HIX 90]. Les objets d'interaction sont des objets génériques, réutilisables et personnalisables comme les boutons, les champs de texte, les barres de défilement, etc.

Dans les outils SimQuest, OASIS, RIDES et SimBest, l'interface est un tableau de bord où des objets d'interaction sont présentés. Nous trouvons quatre types d'objets d'interaction dans les outils :

- Objets d'entrée : ils permettent à l'utilisateur d'agir pour changer des valeurs, entrer des informations, etc. Rentrent dans ce type les boutons, les champs de saisie, les barres de défilement, etc.
- Objets de sortie : ils montrent à l'utilisateur des informations provenant du modèle. Rentrent dans ce type les champs d'affichage numériques et textuels, les graphiques, des objets spécialisés tels que vumètres, thermomètres, etc.
- Objets d'entrée/sortie : ils permettent à la fois l'entrée et la sortie d'informations.
- Objets de décoration (non-interactifs): ils servent à la décoration de l'interface. Rentrent dans ce type les lignes, les figures géométriques, les libellés, les images fixes, etc.

Les fonctionnalités proposées à l'auteur sont :

- La manipulation directe pour placer les objets d'interaction.
- Une bibliothèque d'objets d'interaction avec des objets génériques aux interfaces graphiques, mais aussi avec certains objets spécialisés pour des simulations comme des interrupteurs, des lampes, des connecteurs, des claviers numériques, etc.
- La création de nouveaux objets d'interaction (sauf SimBest).
- La gestion des bibliothèques d'objets (sauf SimBest).
- La personnalisation des objets en apparence (couleur, taille, visibilité, etc.) et au niveau sémantique (par exemple la valeur maximum d'un champ numérique). Les outils proposent pour cela un éditeur des propriétés des objets.

L'aspect Représentation est l'espace le plus mûr des outils, les différences entre les outils résident dans les bibliothèques d'objets fournies aux auteurs. Néanmoins il est regrettable

de ne pas pouvoir utiliser des composants comme les objets ActiveX ou JavaBeans déjà existants, comme point de départ pour la construction de nouveaux objets d'interaction.

2.5 L'espace associations

L'espace associations est consacré à l'établissement de liaisons entre les espaces modèle, représentation et scénario. Nous analysons ici les associations entre l'espace modèle et l'espace représentation; le chapitre 3 s'intéressera aux associations avec l'espace scénario.

Nous reprenons ici la classification des types d'associations faite par J.P. Pernin [PER 96] pp 205-207. Une association est une coopération établie entre deux éléments X et Y, X étant le modèle et Y un objet de représentation ou l'inverse. J.P. Pernin classifie les associations en quatre catégories :

- Activation : le changement ou la consultation d'une propriété de X active une opération de Y. Le déclenchement de cette activation peut dépendre d'une condition sur les propriétés de X.
- Flots de données : le changement d'une propriété de X (P1) affecte la valeur calculée F(X, P1) à une propriété (P2) de Y. Le déclenchement de cette affectation peut dépendre d'une condition sur les propriétés de X.
- Synchronisation : le changement de l'état de X envoie un événement à Y. Le déclenchement de cet envoi peut dépendre d'une condition sur les propriétés de X.
- Consultation : le changement ou la consultation d'une propriété de X active une opération de lecture par X d'une propriété de Y. Le déclenchement de cette lecture peut dépendre d'une condition sur les propriétés de X.

Nous analysons maintenant les types d'association fournis par chaque outil et les fonctionnalités fournies à l'auteur pour les établir.

SimQuest et SimBest ont des associations du type flots de données sans conditions. Une association avec un objet d'entrée signale quelle variable du modèle doit changer si la valeur de l'objet de représentation change. Une association avec un objet de sortie indique quelle variable (ou variables) du modèle influence la visualisation de l'objet. Dans l'éditeur des propriétés des objets de représentation, l'auteur doit spécifier le nom de la variable (ou

attribut) du modèle avec laquelle les objets sont liés. L'avantage principal de l'approche est la simplicité d'utilisation pour l'auteur. Le gros inconvénient est le besoin de définir, pour chaque nouveau type d'objet de représentation, la manière de lier les objets avec le modèle; cela limite la création de nouveaux types d'objets.

OASIS propose un formalisme qui permet l'utilisation des quatre types d'associations. Une association est définie par l'événement (modification ou consultation d'une propriété) qui déclenche l'association, par une condition portant sur les propriétés de l'objet émetteur X et par une action qui correspond à l'activation d'une opération de l'objet destination Y. Etant donné le caractère complexe de la définition des associations avec ce formalisme, OASIS fournit un éditeur spécialisé pour les associations. La Figure 2-14 montre l'interface de cet éditeur. Les avantages principaux de cette approche sont la flexibilité de définition des associations et l'indépendance totale entre l'espace représentation et l'espace modèle. L'inconvénient est le besoin pour l'auteur de faire un travail de modélisation pour les associations, ce qui peut rendre la tâche plus complexe.

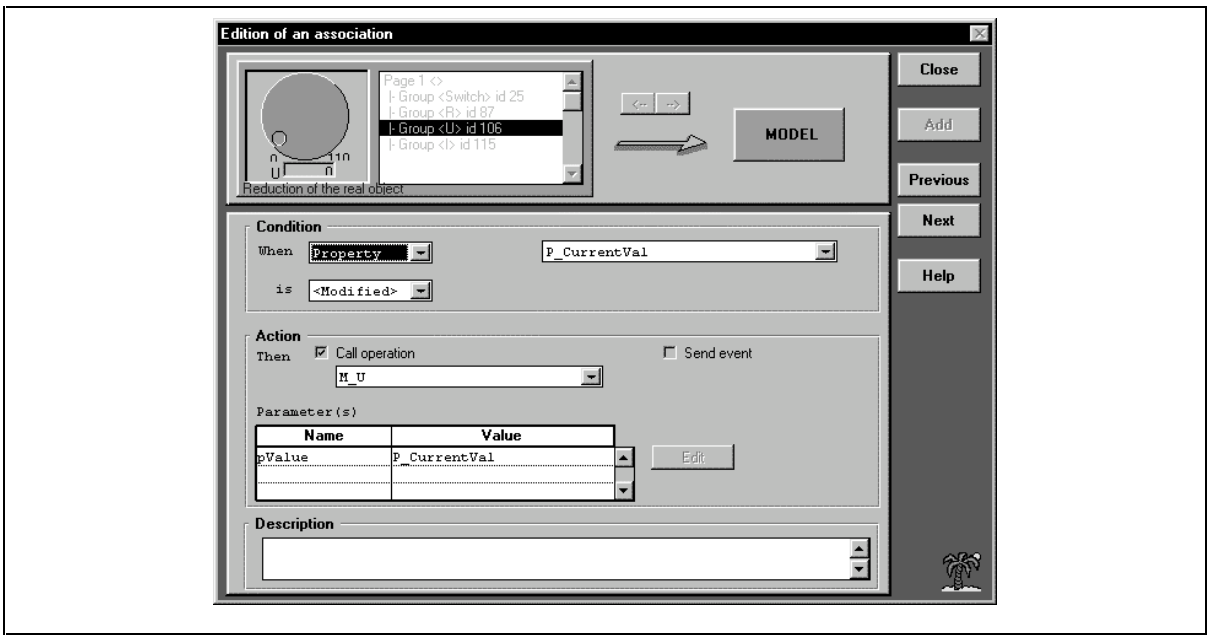


Figure 2-14: Editeur d'associations d'OASIS

RIDES, quant à lui, n'utilise aucun type d'associations. Les associations sont imbriquées dans le modèle: le modèle fait directement des changements sur les objets et vice-versa.

2.6 Résumé

Les systèmes auteurs de simulations étudiés proposent de façon générale une distinction claire entre le modèle fonctionnel et la représentation avec des éditeurs spécialisés pour chaque aspect. Des bibliothèques d'objets prédéfinis sont aussi presque toujours disponibles.

Le mécanisme d'association est moins clairement identifié sauf dans les produits MELISA et OASIS de notre équipe. Il est le plus souvent implicite en reposant sur les noms des variables. Sa mise en évidence dans un espace séparé semble toutefois beaucoup contribuer à la lisibilité et à l'évolutivité des applications.

L'usage de chaque système auteur reste en général très lié au type de formalisme qu'il propose.

Le Tableau 2-1 résume les caractéristiques de chaque système étudié dans ce chapitre.

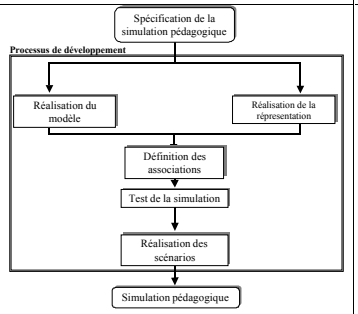
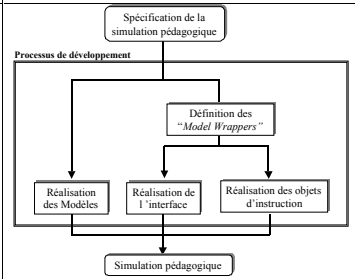
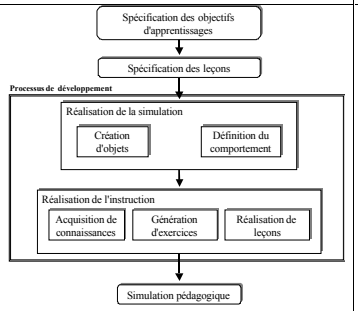
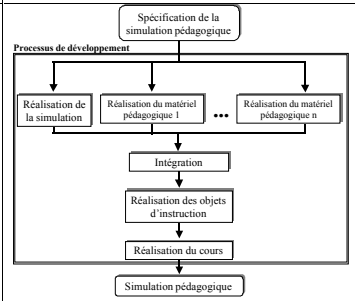
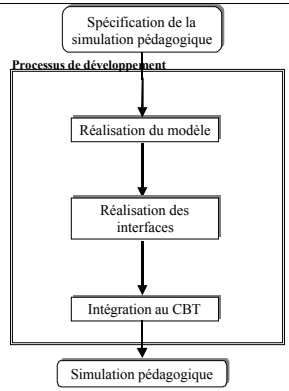
	OASIS	SimQuest	RIDES	SAM	SIMBEST
Processus					
M	*Propriétés + Méthodes + Dynamique (Diagramme d'états et transitions)*	*Equations différentielles et algébriques, modèles construits avec langages de programmation comme C++*	*Objets avec attributs + contraints et événements pour changer la valeur des attributs*	*Dépendant de l'outil utilisé pour construire la simulation*	*Equations différentielles et événements discrets*
A	*Explicites, dans les deux sens, avec conditions*	*Implicites par le nom de la variable*	*Relation un à un, chaque objet est propriétaire de sa représentation*	*Dépendant de l'outil utilisé pour construire la simulation*	*Implicites par le nom de la variable*
R	*Bibliothèques d'objets propriétaires.* *Chaque objet suit le modèle MVC.* *Possibilité d'extension de la bibliothèque*	*Bibliothèque d'objets propriétaires.* *Possibilité de création de nouveaux objets.* *Possibilité d'extension de la bibliothèque*	*Bibliothèque d'objets propriétaires.* *Chaque objet est construit à partir d'un ensemble d'objets graphiques simples.* *Possibilité d'extension de la bibliothèque*	*Dépendant de l'outil utilisé pour construire la simulation*	*Petite bibliothèque d'objets d'entrée et de sortie* *Il n'y a pas possibilité d'extension*

Tableau 2-1: Caractéristiques des outils pour le développement de simulations pédagogiques

3 SCENARIOS ET SIMULATIONS PEDAGOGIQUES

Ce chapitre a pour but d'analyser l'espace scénario dans les systèmes de réalisation de simulations pédagogiques. Comme dans le chapitre 2, nous analysons les formalismes et les outils présents dans les systèmes existants.

3.1 Simulation et objectifs d'apprentissage

L'utilisation d'une simulation s'inscrit dans un processus d'enseignement-apprentissage. L'enseignant (ou formateur) a des objectifs d'apprentissage concrets pour les apprenants. Ces objectifs sont représentables par un graphe qui détermine les pré-requis de chaque objectif (cf. Figure 3-1). L'enseignant propose, pour chaque objectif, des activités d'apprentissage appropriées.

L'enseignant planifie et conçoit les activités d'apprentissage (cours, devoirs, travaux pratiques, discussions en groupes, évaluations, etc.), ainsi que les médias à utiliser. Chaque activité a un but spécifique dans le processus d'enseignement-apprentissage : motivation, présentation de concepts, illustration, application de la théorie, généralisation, transfert de savoir-faire, etc.

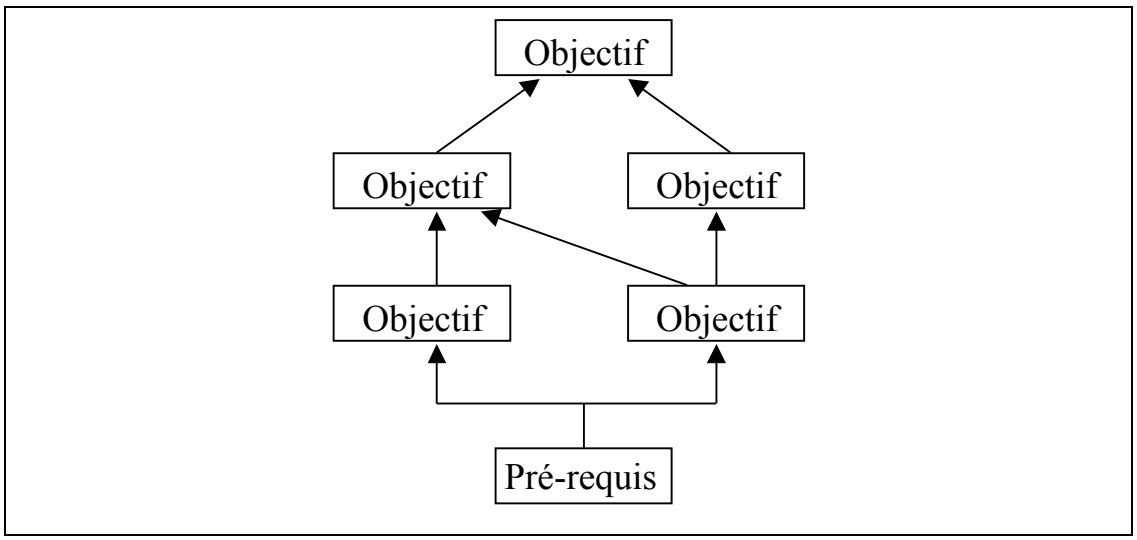


Figure 3-1: Graphe d'objectifs d'apprentissage

Une simulation peut être utilisée pour différents propos (démonstration, introduction de concepts, acquisition ou renforcement) et des activités différentes peuvent être offertes à l'apprenant (par exemple, découvrir la relation entre deux variables, prédire le comportement du système sous des conditions différentes ou vérifier un théorème).

Nous séparons dans ce processus deux niveaux de contrôle pédagogique par l'enseignant. A un premier niveau, l'enseignant définit l'enchaînement des objectifs, les activités à réaliser pour chaque objectif et les critères pour le passage d'un objectif à un autre. Ce niveau est appelé scénario d'enchaînement pédagogique par J.P. Pernin [PER 96].

A un deuxième niveau, l'enseignant contrôle en détail chaque activité: but (appropriation, pratique, renforcement, évaluation, etc.), énoncé, aides à l'apprenant, surveillance à faire pendant la réalisation de l'activité, critères pour considérer l'activité comme accomplie, etc. Ce niveau correspond au scénario de contrôle pédagogique de MARS [PER 96].

L'espace scénario des systèmes de réalisation de simulations pédagogiques est concerné par ces deux niveaux de contrôle. Nous allons maintenant analyser des approches et des outils proposés pour chaque niveau.

3.2 Enchaînement pédagogique

Nous devons examiner les approches que les systèmes de réalisation de simulations pédagogiques adoptent pour définir le scénario d'enchaînement pédagogique. On peut distinguer trois types d'approches :

Une première approche, comme dans OASIS, est de ne fournir aucun outil pour définir cet enchaînement.

Une deuxième approche est de proposer un outil permettant de définir un cours comme une hiérarchie d'objectifs d'apprentissage et d'y associer des exercices. Du point de vue de l'apprenant, l'outil gère automatiquement la progression entre les exercices selon les objectifs atteints. Cette approche est celle de RIDES et de SAM.

La dernière approche est celle de SimQuest, qui base le contrôle de l'enchaînement pédagogique sur l'activation des exercices. Pour chaque exercice, l'auteur définit quels sont les exercices qui vont s'activer quand l'apprenant réussit, échoue ou abandonne. Le contrôle de l'enchaînement est ainsi réparti dans tous les exercices. Du point de vue de l'apprenant, il y a une liste d'exercices actifs.

Une caractéristique importante des outils de ce niveau est qu'ils sont indépendants de la simulation. Ce type d'outil peut être utilisé dans d'autres contextes que les simulations pédagogiques et, d'autre part, il est bien rare d'avoir des cours complets basés uniquement sur des simulations. A notre avis, il est préférable d'avoir un outil spécialisé dans la gestion de la formation et qui permette l'intégration de différents médias et de diverses activités d'apprentissage pour un objectif.

3.3 Contrôle pédagogique

A la différence du niveau précédent, le scénario de contrôle pédagogique a, lui, un rapport direct avec la simulation. L'auteur définit pour l'apprenant une activité précise à faire avec la simulation, activité que nous appellerons exercice. Quels sont les types d'exercices que l'auteur peut définir ? Avec quels outils ? Nous essayons dans la suite de répondre à ces questions. Nous analysons d'abord les types d'exercices proposés par les différents systèmes et leur intérêt. Ensuite, nous présentons les outils nécessaires pour la définition et la mise en œuvre des exercices,

en regardant leurs aspects génériques. Finalement, nous présentons brièvement les environnements spécifiques à chaque système étudié.

3.3.1 Types de contrôles pédagogiques sur des simulations

L'encadrement pédagogique, que les systèmes de réalisation de simulations pédagogiques permettent d'ajouter aux simulations, répond au besoin de fournir à l'apprenant des objectifs précis, de le guider et de l'aider quand il utilise une simulation dans une activité d'apprentissage. Le type d'encadrement pédagogique fourni est influencé par la démarche pédagogique précise que le système veut favoriser.

3.3.2 SimQuest et l'approche cognitive

L'approche cognitive de l'apprentissage, proposée par les théories psychologiques de l'apprentissage comme la Gestalt et les théories de Piaget [GAL 92], a influencé la démarche pédagogique basée sur l'exploration et la découverte, adoptée par SimQuest. L'apprentissage par découverte s'appuie sur l'interaction de l'apprenant avec l'objet de connaissance dans un processus d'exploration, de formulation de vérification d'hypothèses.

Les études faites par l'équipe de SimQuest sur l'apprentissage par découverte avec des simulations [DEJ 96a], [DEJ 96b], ont permis d'identifier quatre types d'aide à fournir à l'apprenant :

- Accès direct à l'information : l'apprenant doit avoir accès aux informations du domaine au fur et à mesure de ses besoins dans son processus de découverte. L'information peut se manifester de différentes manières, par exemple par des explications ou par des démonstrations du fonctionnement du système.
- Aide à la génération d'hypothèses : la génération d'hypothèses est le processus central de l'apprentissage par découverte. Des aides pour la gestion des hypothèses et pour leur vérification sont souhaitables. Les cahiers d'hypothèses et les outils pour la construction guidée d'hypothèses sont des exemples de ce type d'aide.

- Aide pour la réalisation de prédictions : la prédiction est un pas nécessaire pour la vérification des hypothèses. Il s'agit de prédire les valeurs des variables dépendant du système pour un état initial connu, d'observer le comportement du système et finalement de vérifier si les valeurs prédites correspondent aux résultats réels. Quelques exemples de ce type d'aide sont les graphes pour décrire la relation entre deux ou plusieurs variables ou les tableaux de valeurs attendues.
- Aide pour la construction d'un processus d'apprentissage régulé : le processus de découverte doit être un processus systématique et planifié. Diverses études ont montré l'efficacité de l'apprentissage par découverte quand l'apprenant suit un plan dans la recherche et la vérification des hypothèses. L'idée est de structurer le processus autour de la progression du modèle cognitif du système simulé. L'élève progresse d'un modèle simple vers un modèle complet du système, au fur et à mesure qu'il avance dans son processus de découverte.

En considérant ces quatre types d'aides pédagogiques, les projets SMISLE et SERVIVE proposent des exercices (*Instructional Measures*) pour favoriser le processus de découverte de l'apprenant :

1. Exercice de recherche : Exploration libre par l'apprenant qui doit répondre à une question à choix multiple. L'auteur définit la question et les réponses.

2. Exercice d'explicitation : L'apprenant est confronté à une question à choix multiples, et dispose d'un ensemble d'états initiaux pour démarrer l'exploration de la simulation. L'auteur définit la question, les réponses et les états initiaux.

3. Exercice de prédiction : L'apprenant doit prédire les valeurs qu'auront certaines variables à un moment donné. L'état initial est défini par l'auteur.

4. Exercice de manipulation : A partir d'un état initial, l'apprenant doit atteindre un état final, en respectant des contraintes fixées par l'auteur (par exemple, garder une variable à l'intérieur d'une certaine plage de valeurs).

Pour favoriser la progression cognitive, chaque exercice s'appuie sur un modèle du système associé, et c'est par l'enchaînement des exercices (cf. 3.2) que l'auteur peut guider l'apprenant dans le processus de découverte.

3.3.3 RIDES et le béhaviorisme cognitif

La démarche pédagogique de RIDES a été influencée par les théories d'apprentissage de Gagné et par les propositions de Reigeluth [REI 89] basées aussi sur Gagné.

Gagné propose un modèle cognitif de l'apprentissage, ainsi qu'une taxonomie des résultats de l'apprentissage (réponses) et des événements (ou principes) d'apprentissage (stimulus). Ce modèle et cette taxonomie permettent d'associer pour chaque type de résultat les événements appropriés à chaque phase d'apprentissage [GAL 92].

L'approche suivie par RIDES a consisté à définir des types de connaissances désirables pour un apprentissage basé sur des simulations et à proposer les types d'instruction à fournir à l'apprenant, en se basant sur les événements et les phases d'apprentissage proposées par Gagné [TOW 95]. RIDES propose aussi deux moments d'apprentissage ou modes d'instruction, qui correspondent à des regroupements des phases de Gagné :

- Moment d'appropriation, qui regroupe les phases de motivation, de compréhension, d'acquisition et de rétention. Le début du processus est la génération d'une expectative qui incite l'apprenant à apprendre (motivation), et qui ensuite attire l'attention de l'apprenant sur les aspects importants à apprendre (compréhension); l'apprenant ajoute la nouvelle connaissance à sa structure mentale dans les phases d'acquisition et de rétention.
- Moment de pratique, qui regroupe les phases restantes du processus; la phase de récupération qui demande de mobiliser la connaissance, la phase de transfert qui exige le transfert et la généralisation de la connaissance acquise, la phase de réponse qui met l'apprenant dans des situations d'évaluation (ou d'auto-évaluation) et la phase de renforcement qui permet de réorienter le processus selon les résultats obtenus.

Le Tableau 3-1 montre les types d'activités pour chaque type de connaissance et pour chaque mode d'apprentissage.

	Description	Classes de connaissances	Types d'activités pour la classe
Connaissances sur les objets	Relatives à la compréhension des caractéristiques et du fonctionnement des éléments du système	Nommer Localiser Aspect Fonction générale Fonction spécifique	**Appropriation:** Démonstrations des objets avec noms, localisations et fonctions Demander à l'apprenant de répéter les actions de la démonstration. **Pratique:** Étant donné le nom et/ou la fonction d'un objet, demander à l'apprenant de le localiser dans le système. En cas d'échec, il faut guider l'apprenant et lui donner un feed-back.
Connaissances d'opérations et de procédures	Relatives à la manipulation du système et à l'évaluation de l'état du système	Manipulation des contrôles Lecture et interprétation des indicateurs	**Appropriation:** Démonstrations des réglages et des interprétations des contrôles. Demander à l'apprenant de répéter les actions de la démonstration. **Pratique:** Demander à l'apprenant de régler ou d'interpréter un contrôle. En cas d'échec, il faut guider l'apprenant et lui donner un feed-back.
		Configurations (Combinaisons de réglages des contrôles)	**Appropriation:** Démonstrations de comment faire une configuration. Demander à l'apprenant de répéter les actions de la démonstration. **Pratique:** Demander à l'apprenant de faire une configuration. En cas d'échec, il faut guider l'apprenant et lui donner un feed-back..
		Procédures : deux types de procédures, fixes (séquence fixes de pas) ou variables (plusieurs séquences valides de pas)	**Appropriation:** Démonstrations des procédures en indiquant à l'apprenant les effets de chaque pas dans le système. Pour les procédures variables, diverses possibilités sont montrées. **Pratique:** Trois niveaux de pratique: Chaque action à faire est demandée explicitement à l'apprenant, pas à pas. Chaque action à faire est demandée partiellement à l'apprenant, pas à pas. Par exemple, régler un contrôle mais sans dire la valeur. L'apprenant doit reproduire la procédure sans aide, pas à pas.
Connaissances du système	Relatives à l'organisation du système dans des conditions normales et anormales.	Structure	**Appropriation:** Démonstrations des objets, par sous-systèmes ou classes. Demander à l'apprenant de répéter les actions de la démonstration. **Pratique:** Demander à l'apprenant de repérer les objets d'une classe ou d'un sous-système
		Relations de causalité dans des conditions normales	**Appropriation:** Pour différentes conditions du système, démonstrations des objets du système qui causent ou sont affectés par la condition. Il faut ajouter des explications complémentaires pour chaque objet. Demander à l'apprenant de répéter les actions de la démonstration. **Pratique:** Demander à l'apprenant de montrer les objets affectés par une condition. Demander à l'apprenant de prédire les valeurs des objets affectés par une condition. Demander à l'apprenant de mettre le système dans une condition normale, à partir d'une condition anormale.
		Relations de causalité dans des conditions anormales	
Connaissances de diagnostic	Relatives à la l'interprétation et réalisations de test de diagnostic	Tests du système Stratégies de diagnostic	**Pratique:** A partir d'une situation de panne du système, demander à l'apprenant de réparer le système. Les remplacements et le temps sont des facteurs à mesurer et vérifier. L'apprenant peut consulter une documentation d'aide.

Tableau 3-1: Types d'activités par classe de connaissance en RIDES

3.3.4 Types d'exercices de SAM

SAM ne mentionne pas d'approche pédagogique particulière et ses propositions sont extrêmement ouvertes car le type d'exercices dépend de la configuration particulière du système.

Trois types d'exercices sont proposés :

- Démonstration : Utilisation de la simulation pour démontrer le comportement du système simulé dans le temps.
- Solution de problèmes : Demande à l'apprenant de résoudre un problème en manipulant la simulation. Les actions de l'apprenant sont surveillées et des aides sont fournies si nécessaires.
- Exploration : L'apprenant utilise la simulation librement pour explorer le comportement du système.

3.3.5 Comparaison entre les différents types d'exercices

Le fait que chaque outil ait suivi une approche différente pour définir ses exercices a conduit à un éventail varié d'activités que nous présentons de façon synthétique ci-dessous.

- la démonstration : la simulation est utilisée pour informer sur la composition, la structure et le comportement du système simulé. Elle sert aussi de support pour démontrer des procédures opératoires (réglage de contrôles, configurations, dépannage,…). Une vérification des connaissances données par la démonstration peut être proposée sous forme de QCM (éventuellement graphique) ou en demandant à l'élève de reproduire la démonstration.
- l'exploration libre, sans contrôle pédagogique.
- l'exploration motivée par un QCM. L'apprenant doit manipuler la simulation pour trouver la réponse aux questions. L'exploration peut éventuellement être guidée par la suggestion d'un ensemble d'états à explorer.

- l'exercice de prédiction. Étant donné l'état initial d'une simulation ou certaines conditions de fonctionnement, l'apprenant doit prédire les valeurs qu'auront certaines variables à un moment donné.
- l'exercice à but. A partir d'un état initial, l'apprenant doit manipuler la simulation pour atteindre l'état final demandé. L'exercice peut être plus ou moins complexe selon l'objectif fixé : régler un contrôle, réaliser une configuration, exécuter une procédure opératoire, réparer le système, etc. L'élève doit éventuellement respecter certaines contraintes pendant la manipulation de la simulation (par exemple, garder une variable à l'intérieur d'une certaine plage de valeurs).

Nombre de ces exercices ne nécessitent pas de surveiller l'action de l'apprenant sur la simulation; il suffit de vérifier que les réponses données sont correctes. En revanche, les exercices de reproduction d'une procédure opératoire et les exercices à but exigent un suivi des actions de l'apprenant sur la simulation. Dans ce dernier cas, le contrôle généralement proposé est relativement restreint: objectif final atteint ou non, contraintes respectées ou non.

Une autre différence entre les propositions est la considération du moment d'apprentissage. RIDES et OASIS se distinguent en offrant différents modes d'exécution d'un exercice selon que l'apprenant veut l'utiliser pour apprendre, pour pratiquer ou pour s'évaluer.

3.4 Le contrôle pédagogique dans OASIS

Nous avons contribué à définir le contrôle pédagogique mis en œuvre dans OASIS. L'approche adoptée s'appuie sur la démarche pratiquée dans des expérimentations réelles. La base n'est pas une théorie d'apprentissage, mais une démarche concrète semblable aux pratiques observées en séances de travaux pratiques:

- Une expérimentation fixe un objectif à atteindre en un temps limité.
- "…l'enseignant ne s'intéresse de façon continue ni aux intentions ni aux comportements des élèves. Sa démarche est de vérifier, le plus souvent possible, l'état d'avancement de l'expérimentation en cours, et de réagir en cas de nécessité." [PER 96]

Cette démarche a servi de base à la définition de la notion de scénario de contrôle pédagogique dans MARS et à son implémentation en OASIS. Nous proposons un type d'exercice à but défini par un scénario qui comporte (cf. Figure 3-2) :

- Un état initial, qui correspond à l'état dans lequel est placée la simulation au début de l'exercice.
- Une séquence d'étapes à franchir par l'apprenant; la dernière étape correspond à l'objectif final à atteindre par l'apprenant.
- Un ensemble de contrôles globaux (situations à surveiller).

Une étape est composée de trois éléments : une situation (ou objectif) à atteindre, un ensemble de contrôles et une réactivité. La situation à atteindre est un état de la simulation par lequel doit passer l'élève. La réactivité permet de définir les instructions à donner à l'élève pour l'étape, les messages (en forme de texte, vidéo, et autres) à délivrer à l'élève en cas d'échec ou de réussite et pour le guider, et s'il y a lieu le temps maximum accordé à l'élève pour atteindre l'objectif.

Les contrôles sont des situations de la simulation à vérifier pendant l'étape, ou pendant tout l'exercice, soit parce qu'elles correspondent à des erreurs, soit parce que ce sont des situations spéciales où il est nécessaire de guider ou d'aider l'élève. Une réactivité est associée à chaque contrôle; elle permet de définir les messages d'aide ou d'explication à délivrer à l'élève s'il place la simulation dans l'état défini par le contrôle.

Nous permettons ainsi d'étoffer le contrôle habituellement proposé sur des exercices à but en nous intéressant à la façon dont l'élève progresse vers l'objectif.

Comme dans RIDES, nous considérons divers modes d'apprentissage. Dans le mode entraînement, l'apprenant réalise l'exercice étape par étape, en vérifiant à chaque fin d'étape s'il a réussi; il a la possibilité de refaire l'étape en cas d'échec. Le mode évaluation consiste à laisser agir l'apprenant durant un temps limité sur la simulation pour atteindre l'étape finale. Le contrôle pédagogique vérifie en arrière plan les étapes franchies par l'apprenant.

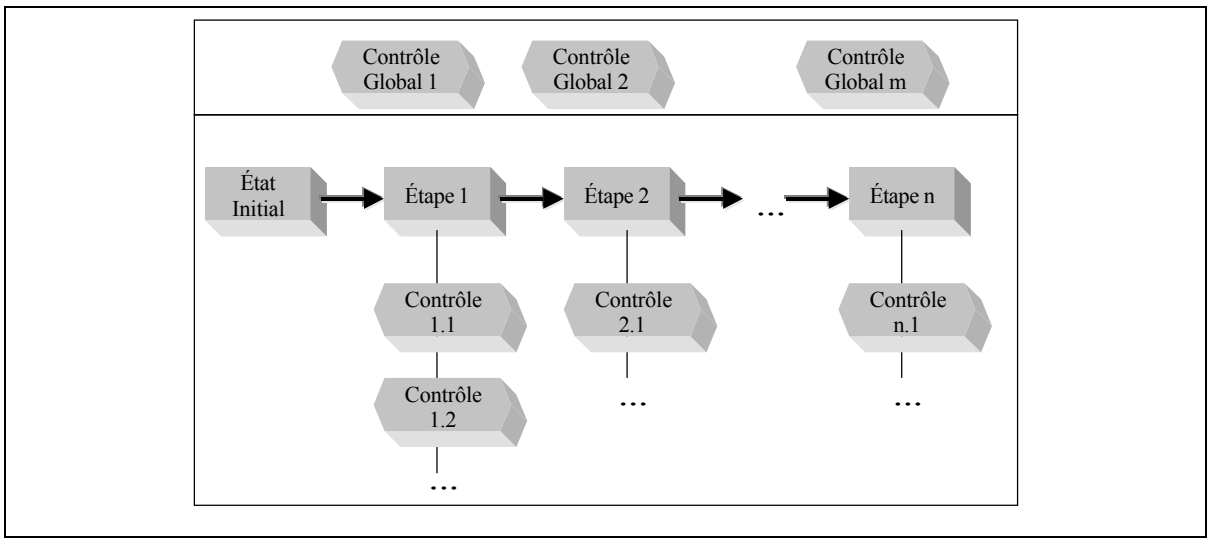


Figure 3-2: Scénario de contrôle pédagogique d'OASIS

3.5 Environnements de contrôle pédagogique

Les fonctionnalités des outils de contrôle pédagogique concernent d'une part la définition des exercices par l'auteur et d'autre part la réalisation de l'exercice par l'apprenant.

Une partie du système permet la création et l'édition du contrôle pédagogique. Cette partie définit ce que nous appelons l'**environnement auteur**. Notons que, dans certains cas, pour définir le contrôle pédagogique, l'auteur peut être amené à utiliser aussi la simulation (Cf. Figure 3-3).

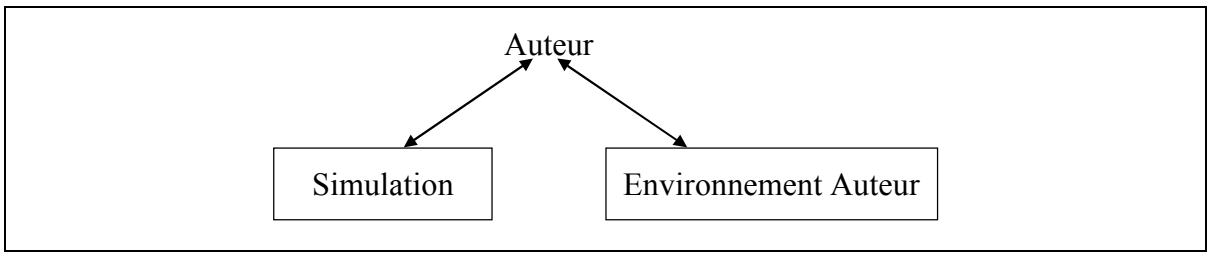


Figure 3-3: Phase de création du contrôle pédagogique

Dans la phase d'exécution, c'est l'apprenant qui réalise les exercices proposés. Dans ce cas, l'environnement utilisé est l'**environnement apprenant** qui permet l'exécution des contrôles pédagogiques définis par l'auteur (Cf. Figure 3-4).

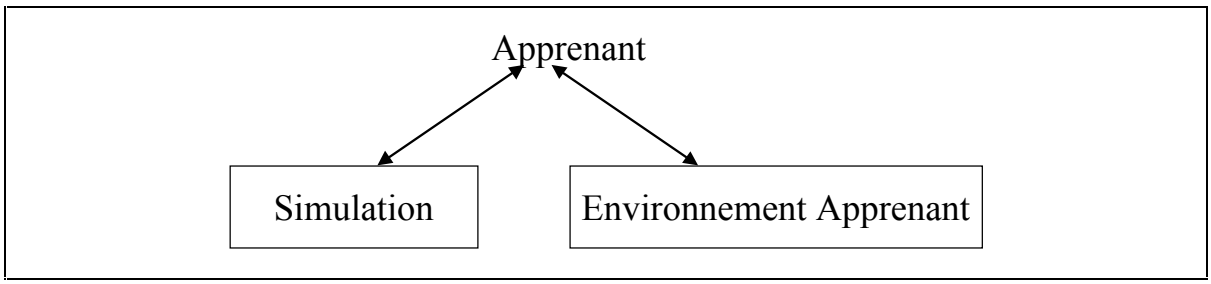


Figure 3-4: Phase d'exécution du contrôle pédagogique

Nous allons maintenant analyser les aspects généraux relatifs à chacun de ces deux environnements, ainsi que les environnements spécifiques fournis par les systèmes de réalisation de simulations pédagogiques.

3.5.1 L'environnement auteur

Un exercice a en général trois phases :

- La phase de consigne
- La phase d'action de l'apprenant sur la simulation
- La phase d'évaluation de la performance de l'apprenant

Dans la phase de consigne, le système présente à l'apprenant le but de l'exercice et met, si nécessaire, la simulation dans l'état initial défini pour cet exercice. Les messages d'instruction peuvent utiliser différents médias (son, texte, vidéo, animations, etc.). Les environnements auteurs doivent fournir des moyens pour définir ces instructions ainsi que l'état initial.

Dans la phase d'action, l'apprenant interagit avec la simulation. Selon le type d'exercice, il peut être guidé par le système. Pour un exercice de recherche de SimQuest, il n'y a pas contrôle de la part du système, mais pour un exercice de RIDES ou OASIS, l'apprenant est surveillé en permanence et averti de ses erreurs. Les environnements auteurs doivent fournir des moyens pour définir le contrôle à faire et les messages destinés à l'apprenant.

Dans la phase d'évaluation, le système vérifie si l'apprenant a réussi l'exercice. Les environnements auteurs doivent fournir des moyens pour définir la réponse en termes des actions à effectuer ou d'état à atteindre, et les messages destinés à l'apprenant en cas de réussite et en cas d'échec.

3.5.2 L'environnement apprenant

L'environnement apprenant est chargé de présenter les exercices à l'apprenant et d'aider l'apprenant. Les deux niveaux de contrôle pédagogique sont concernés. Au niveau de l'enchaînement pédagogique, l'environnement doit permettre de choisir un exercice à faire en fonction de l'enchaînement proposé par l'outil. Au niveau du scénario de contrôle pédagogique, l'environnement doit permettre de travailler sur l'exercice.

Au niveau du scénario de contrôle pédagogique, la fonction de l'environnement consiste à exécuter les trois phases d'un exercice : donner la consigne, mettre la simulation dans l'état initial fixé par l'exercice, surveiller la progression de l'apprenant sur la simulation, fournir le guidage défini pour l'exercice, et finalement déterminer si l'apprenant a réussi ou non l'exercice.

La vérification peut être indépendante de la simulation, ou dépendre de l'état de la simulation ou des actions effectuées par l'apprenant sur la simulation. Pour les exercices où la réponse n'est pas associée directement à la simulation, on se limite à vérifier si la réponse de l'apprenant est correcte par rapport à la réponse établie dans l'exercice. Pour les exercices qui dépendent de l'état de la simulation, la vérification consiste à récupérer l'état de la simulation, puis à le comparer avec la réponse définie pour l'exercice. Pour les exercices qui dépendent des actions de l'apprenant, on doit vérifier si l'action de l'apprenant correspond à l'action spécifiée dans l'exercice.

L'environnement doit assurer la surveillance et l'évaluation adéquates. Il peut fournir en complément des aides indépendantes des exercices, par exemple un cahier de notes ou un tableur pour enregistrer les données, etc.

3.5.3 Les environnements proposés par les systèmes étudiés

Nous allons maintenant présenter les principales fonctionnalités des environnements auteur et apprenant proposés par les systèmes de réalisation de simulations pédagogiques que nous avons étudiés.

3.5.3.1 SAM

L'environnement auteur

SAM offre un éditeur graphique pour définir un exercice comme un *flowchart,* dont les nœuds sont des fonctions à choisir dans une palette. La palette contient une icône pour chaque outil intégré dans SAM, et des fonctions génériques telles qu'instructions conditionnelles (if) et instructions itératives (while). L'icône d'un outil donne accès à une palette avec les commandes disponibles pour cet outil. En particulier, pour la simulation, SAM liste un ensemble de commandes qui doivent être proposées par l'outil de simulation : consulter/changer la valeur d'une variable, activer/désactiver la surveillance de changement d'une variable, consulter/changer l'état de la simulation.

L'environnement auteur est ainsi un éditeur graphique de programmes. La tâche de construction d'un exercice sur la simulation est convertie en une tâche de programmation, ce qui n'est pas forcément adapté aux auteurs ciblés par les systèmes de réalisation de simulations pédagogiques.

L'environnement apprenant

L'environnement apprenant de SAM n'est pas standard pour toutes les applications car il dépend des outils que l'auteur veut intégrer dans le système.

3.5.3.2 RIDES

L'environnement auteur

RIDES offre deux éditeurs pour la création d'exercices: un éditeur d'exercices libres et un éditeur d'exercices types (*Patterned exercise editor*). Le premier permet de créer des exercices de manière libre, tandis que l'éditeur d'exercices types permet de générer automatiquement des exercices correspondant aux types présentés dans le Tableau 3-1. L'auteur peut ensuite grouper les exercices dans différentes leçons (cf. Figure 3-5).

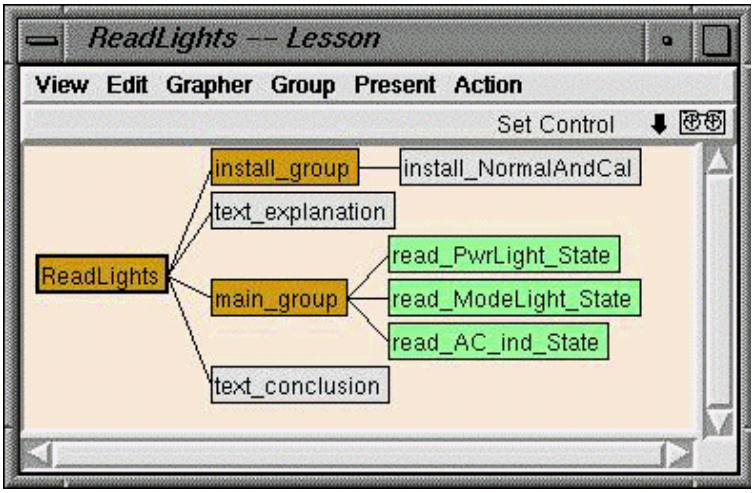


Figure 3-5: Exemple de RIDES (pris de [MUN 95a])

Pour construire un exercice libre, l'auteur dispose d'un ensemble d'items d'instruction primitifs qu'il peut utiliser, grouper, personnaliser. Chaque item permet à l'auteur de définir une interaction avec l'apprenant : régler un contrôle, montrer un texte, signaler un objet, lire un indicateur, remplacer un objet, etc. A chaque item est associé un éditeur qui permet de le personnaliser, la Figure 3-6 montre l'exemple de l'éditeur de l'item de réglage d'un contrôle (*Set Control*). En général, dans un éditeur d'item, l'auteur définit les modes dans lesquels l'item s'utilise, les objets concernés, les messages à montrer à l'apprenant dans chaque mode, le niveau de difficulté de l'item, la limite de temps pour réaliser l'action, la note en cas de réussite, …

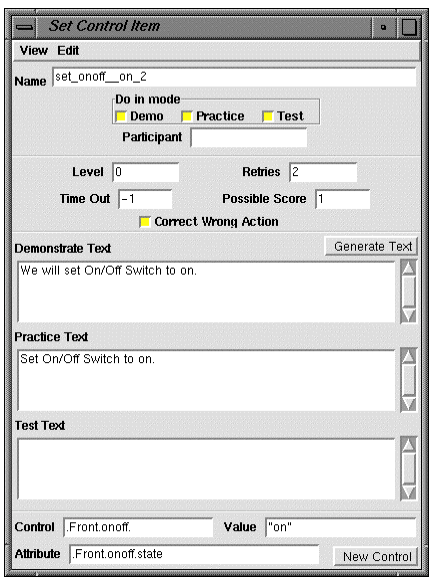


Figure 3-6 :Editeur de l'item d'instruction *Set Control* (pris de [MUN 95a])

Pour générer un exercice de l'un des types proposés, l'environnement auteur RIDES extrait des connaissances de la simulation proprement dite et demande à l'auteur un certain nombre d'informations complémentaires. Ce mécanisme est basé sur le processus de génération d'entraînement par ordinateur, appelé CGT pour *Computer Generated Training* et proposé par Towne [TOW 95]. Le processus CGT est décrit dans la Figure 3-7.

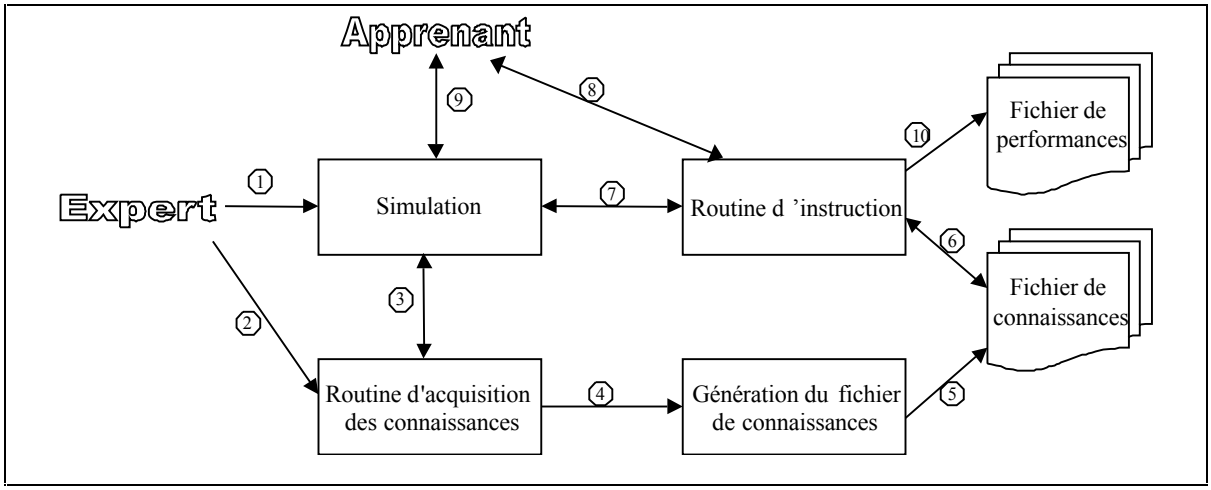


Figure 3-7: Le processus CGT

Le point de départ (1) est la construction d'une simulation par l'expert. Ensuite, une routine d'acquisition de connaissances (2 et 3) extrait les informations pertinentes du modèle (par exemple les noms des objets de la simulation, les influences de chaque objet sur les autres) et demande à l'expert des informations complémentaires qui dépendent du type de connaissances à enseigner. Par exemple, pour les connaissances sur les objets, la routine d'acquisition peut demander la fonction de chaque objet. Pour les connaissances sur les procédures, la routine peut demander à l'expert de montrer sur la simulation la séquence d'actions nécessaires au déroulement d'une procédure spécifique. Avec les informations données par l'expert et les informations du modèle, la routine d'acquisition génère un fichier de connaissances (5), qui contient les données nécessaires pour générer l'instruction. La routine d'instruction consulte les informations du fichier de connaissances et du modèle (6, 7 et 8). L'apprenant réalise l'exercice (9). Et la routine d'instruction mémorise sa performance dans le fichier de performance (10).

RIDES offre trois fonctionnalités pour donner des informations complémentaires pour la génération des exercices :

- Définition des attributs des objets graphiques, par exemple le nom de l'objet.

- L'éditeur d'unité de connaissance, chaque unité étant composée d'une référence à un objet de la simulation, d'une collection de discussions (textes) créés par l'auteur et classés par thèmes, d'une liste d'exercices associés, d'une liste d'unités de connaissances connexes et d'une liste de mots clés.
- L'éditeur d'interactions qui permet de définir une liste de références des objets qui dépendent d'un objet, et une liste de références des objets qui agissent sur cet objet.

Pour générer automatiquement un exercice, l'auteur utilise un éditeur de type d'exercice pour personnaliser l'exercice, puis RIDES génère automatiquement une leçon. La Figure 3-5 montre la leçon générée pour un exercice de lecture d'indicateurs. RIDES propose 15 types d'exercices et, pour chacun, un éditeur de paramètres. Ces paramètres incluent les états initiaux de la simulation, les objets, les contrôles et les indicateurs qui interviennent, et l'ordre aléatoire ou séquentiel d'exécution.

L'environnement apprenant

Pour l'exécution d'un exercice, l'apprenant dispose de l'interface de la simulation et, en plus, d'une fenêtre pour l'affichage des consignes et des messages. L'apprenant a trois boutons: pour arrêter ou reprendre l'exécution et pour dire qu'il ne connaît pas la réponse. L'environnement produit, dans un fichier texte, une trace sur la performance de l'apprenant.

3.5.3.3 SimQuest

L'environnement auteur

L'éditeur de l'environnement auteur de SimQuest (cf. Figure 3-8) permet de définir d'une part l'enchaînement des exercices (appelé contrôle) et d'autre part les exercices eux-mêmes (de type recherche, explicitation, prédiction ou opération). Un exercice est défini à l'aide de quatre onglets :

- L'onglet général pour le nom de l'exercice, sa description, son niveau de difficulté (de 1 à 4), la langue, et la visibilité des variables.
- L'onglet état initial pour l'ensemble des états initiaux de l'exercice. Pour chaque état initial, l'auteur définit les valeurs initiales des variables du modèle.

- L'onglet question pour la question à poser à l'apprenant. Il est possible de la définir avec texte, vidéo ou son.
- L'onglet réponse dépend du type de l'exercice : pour les exercices de recherche et d'explicitation, l'auteur définit les options de la question à choix multiples, une explication pour chaque option et la réponse correcte; pour les exercices de prédiction, l'auteur définit l'ensemble de variables dont il faut prédire la valeur, les valeurs à prédire, et le marge d'erreur accepté; et pour les exercices d'opération, l'auteur définit l'état final à atteindre avec des valeurs minimum et maximum pour chaque variable, et les contraintes à appliquer pendant l'exercice, chaque contrainte étant une plage de valeurs pour les variables plus une explication associée à donner si la variable sort de la plage fixée.

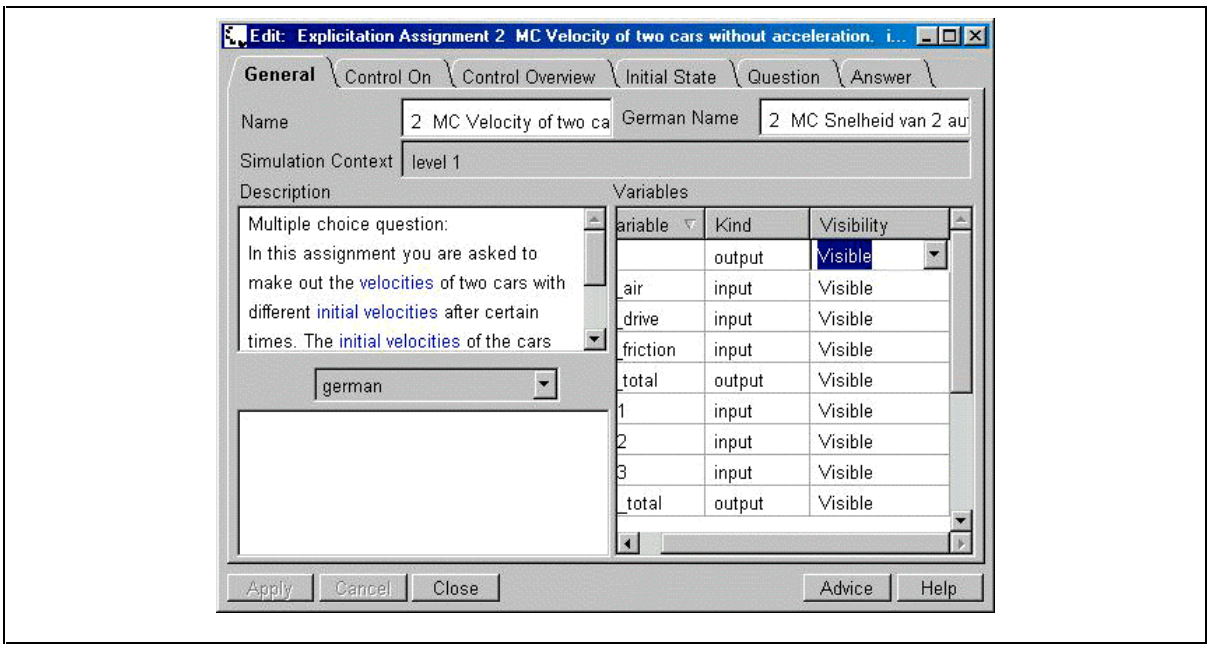


Figure 3-8: Environnement auteur de SimQuest (d'après [SER 99])

Insistons sur les caractéristiques les plus importantes de cet éditeur :

- Pour chaque variable du modèle, l'auteur peut définir sa visibilité et sa modifiabilité dans l'exercice. Ceci permet à l'auteur de contrôler l'interface de la simulation pour chaque exercice.
- L'auteur peut définir le niveau de difficulté des exercices. Ceci permet à l'environnement apprenant de proposer les exercices par niveaux.

- L'auteur dispose d'une aide pédagogique pour la construction des exercices (bouton *Advice* dans la Figure 3-8). Cette aide est un système expert qui conseille sur le type d'exercices à utiliser en fonction de la simulation et du domaine.
- Ce type d'environnement est très simple à utiliser par l'auteur (sauf, rappelons-le, la partie enchaînement qui est difficile à maîtriser parce que répartie entre les différents exercices).

L'environnement apprenant

L'environnement apprenant de SimQuest présente trois fenêtres (cf. Figure 3-9): une pour la simulation, une pour l'exercice et une pour lister les éléments disponibles (simulations, exercices par niveau, aides et explications). L'apprenant peut choisir parmi les exercices disponibles, la liste de ceux-ci étant automatiquement mise à jour en fonction de ses réussites et échecs.

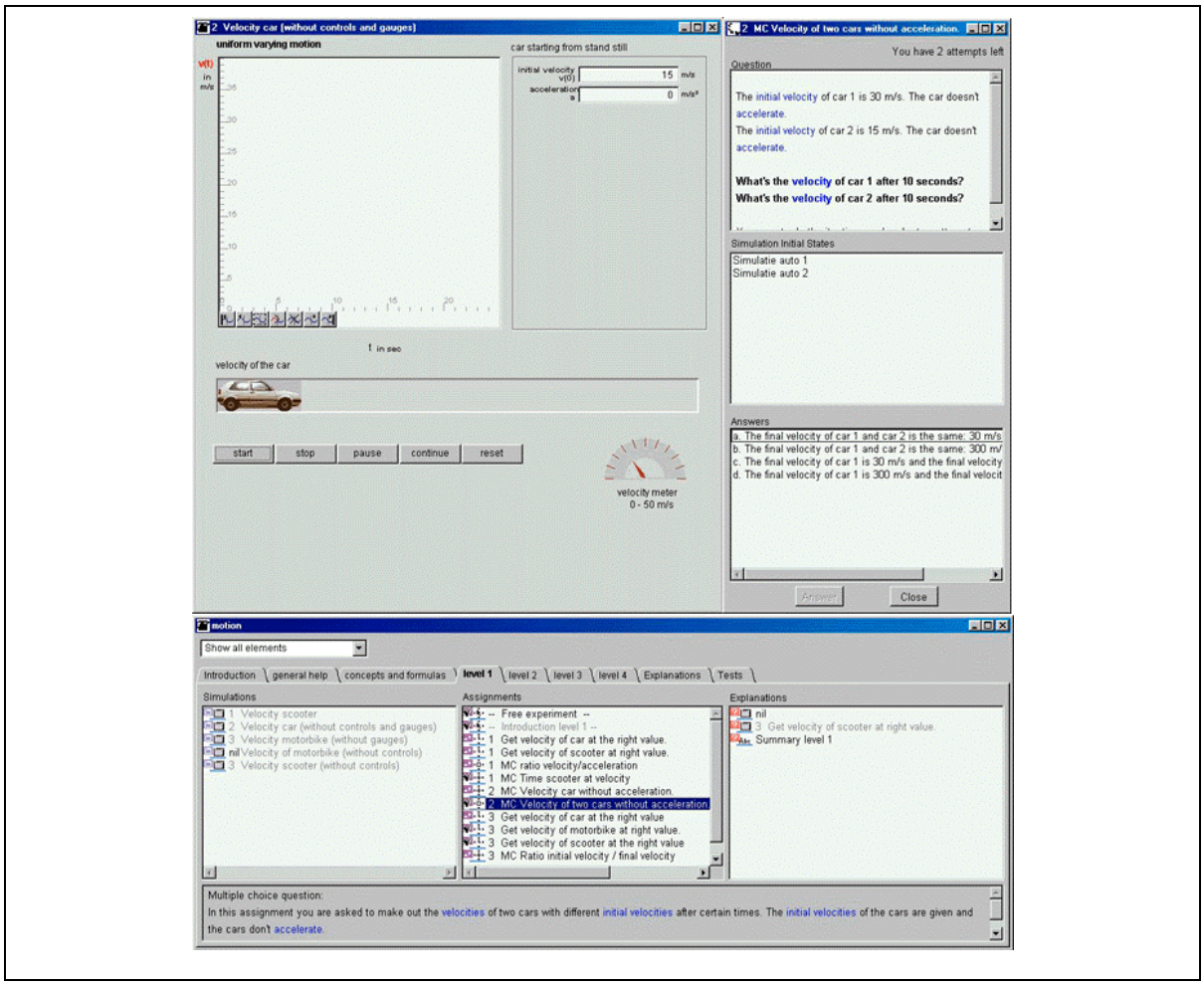


Figure 3-9: Environnement Apprenant de SimQuest (D'après [SER 99])

L'avantage de ce type d'environnement est que l'apprenant est graduellement conduit dans une démarche d'exploration car il n'a à sa disposition à un moment donnée que les exercices qu'il est capable de réaliser. Bien que ce soit avantageux pour l'apprenant, nous avons déjà remarqué que la planification des enchaînements n'est pas une tâche facile pour l'auteur.

Dans SMISLE, le prédécesseur de SimQuest, l'apprenant disposait de plus d'un éditeur lui permettant de construire des hypothèses à partir des résultats obtenus dans les exercices.

3.5.4 Conclusion sur les environnements auteurs étudiés

Les environnements étudiés peuvent se regrouper en catégories selon qu'ils permettent de produire des exercices de types prédéfinis ou absolument quelconques. L'aide fournie par l'environnement est inversement proportionnelle à la liberté qui est laissée à l'auteur.

- Les outils génériques SAM et RIDES (en partie) fournissent un ensemble d'instructions de base que l'auteur compose à l'aide d'un éditeur spécialisé, en effectuant une tâche proche de la programmation.

- L'environnement SimQuest propose différents exercices types et fournit à l'auteur des éditeurs spécifiques à ces types. Le travail de l'auteur en est facilité.

- L'environnement RIDES, dans sa partie génération automatique d'exercices types, réduit encore la tâche de l'auteur.

3.6 L'environnement OASIS

L'environnement auteur OASIS, en permettant à l'auteur de définir des exercices types à l'aide d'éditeurs spécialisés, se situe dans la même catégorie que SimQuest. Nous avons spécifié avec précision au paragraphe 3.4 le contrôle pédagogique d'OASIS sur des exercices à but et vu en quoi il était plus riche que les contrôles habituellement proposés sur ce type d'exercices.

Nous souhaitons maintenant décrire les environnements auteur et apprenant correspondants, en essayant de souligner l'originalité de nos outils.

L'environnement auteur

L'éditeur de scénarios permet à l'auteur de définir l'état initial, les étapes à franchir successivement par l'apprenant et les contrôles d'un scénario; il propose des fonctionnalités pour ajouter, supprimer et modifier tous ces éléments. La Figure 3-10 montre une partie de l'interface de cet éditeur.

Nous avons vu que dans l'environnement Simquest, la situation initiale, la situation à obtenir et les contraintes caractérisant un exercice sont décrites par l'auteur en donnant pour chacune la valeur (ou la plage de valeur) de chaque variable de la simulation. Dans l'environnement OASIS, le processus de développement préconisé permet d'obtenir d'abord une simulation dite libre (sans contrôle pédagogique), avant de définir les exercices portant sur cette simulation. Nous avons saisi cette opportunité pour faciliter la description des différentes situations caractérisant un exercice. En effet, l'auteur va pouvoir spécifier une situation en manipulant simplement la simulation libre (comme le ferait un élève) jusqu'à obtention de cette situation et en la "photographiant". Le système OASIS mémorise alors automatiquement les valeurs caractérisant cette situation.

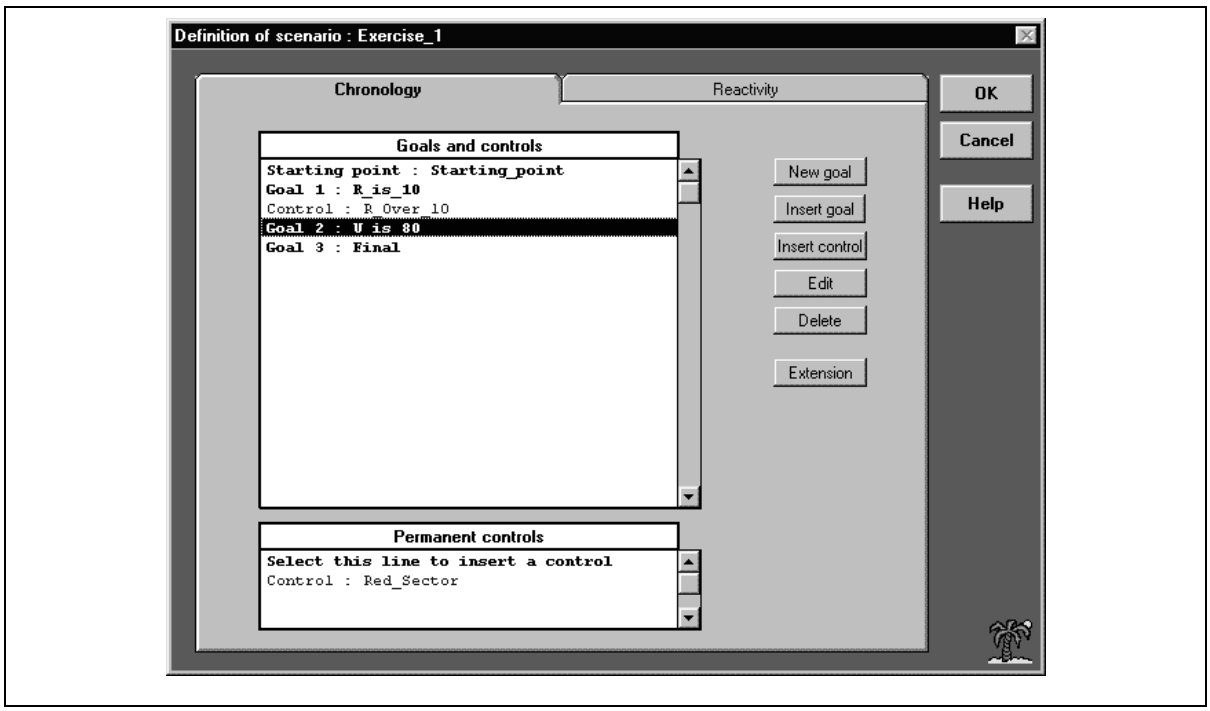


Figure 3-10: Editeur de scénarios d'OASIS

L'auteur peut de plus ensuite expliciter dans quelle mesure une situation doit être considérée comme correcte lorsqu'elle est proche de la situation à atteindre (telle que photographiée). Il fait afficher les valeurs automatiquement mémorisées et peut élargir les

réponses acceptables en précisant par exemple que la valeur d'une certaine propriété est indifférente, ou que la valeur d'une autre propriété doit être comprise dans un certain intervalle (alors que la photographie imposait une valeur précise contenue dans cet intervalle).

Un des intérêts de cette définition par "photographie" est la simplicité de mise en œuvre par l'auteur de l'exercice (qui n'est pas forcément l'auteur de la simulation). De plus, cette façon de procéder permet à l'auteur de réaliser l'exercice (situation initiale, objectif de l'étape 1,…) comme le ferait l'élève, et donc de garantir sa faisabilité. L'auteur dispose d'une vue chronologique des situations caractérisant l'exercice.

L'auteur spécifie également les différentes instructions et retours d'informations qu'il souhaite donner à l'élève :

- la consigne globale, le retour d'information final (en cas d'échec ou de réussite) et éventuellement le temps limite imposé pour faire l'exercice (en mode évaluation).
- pour chaque étape, la consigne, l'aide disponible, les retours d'informations (en cas d'échec ou de réussite).
- pour chaque situation à surveiller, le retour d'information à donner lorsqu'elle est atteinte.

L'auteur dispose d'une vue chronologique des informations qui pourront être diffusées à l'élève.

Ces informations étant données, l'auteur peut tester l'exercice qui est alors disponible dans les 2 modes d'exécution possibles, apprentissage et évaluation. Un gestionnaire d'exercices permet de créer, modifier, supprimer, tester… des exercices portant sur une simulation.

L'environnement apprenant

L'apprenant peut utiliser la simulation de manière libre sans faire appel à un scénario. Il peut aussi choisir un scénario parmi ceux définis par l'auteur, ainsi que le mode d'exécution du scénario : entraînement ou évaluation. Pour contrôler le déroulement d'un scénario, l'apprenant dispose d'une palette et d'un menu pour consulter la consigne et l'aide, pour

démarrer/arrêter l'exécution de l'exercice, et pour demander la vérification du bon achèvement d'une étape.

En mode entraînement, l'environnement "suit" la progression de l'élève à l'intérieur des étapes et détecte le déclenchement de contrôles. Les retours d'information sont donnés en direct.

En mode évaluation, l'environnement produit un rapport final indiquant quelles étapes ont été franchies par l'apprenant et quels contrôles ont été déclenchés.

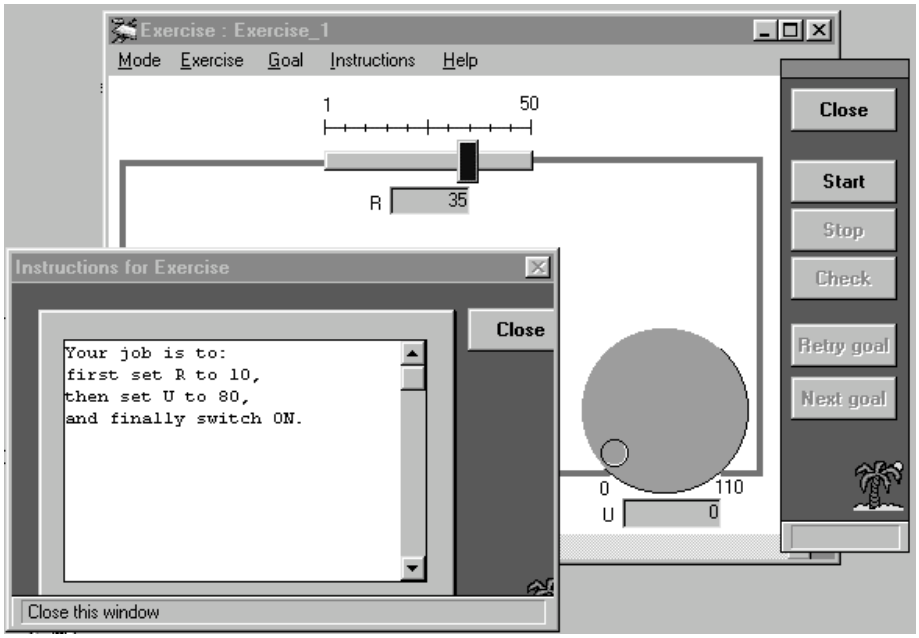


Figure 3-11: Environnement Apprenant d'OASIS

3.7 Une proposition: indépendance entre simulation et scénario

Nous venons d'étudier l'espace scénario de quelques systèmes de réalisation de simulations pédagogiques. Nous constatons que :

- La définition des exercices est un processus postérieur à la réalisation de la simulation.
- Des approches pédagogiques différentes ont été utilisées dans chaque système pour définir la notion d'exercice.
- L'espace scénario est lié à la simulation au moment de la définition des exercices et au moment de l'exécution.

- L'espace scénario d'un système ne peut utiliser que des simulations réalisées avec ce même système.

Bien que les approches pédagogiques soient très différentes, elles ne s'excluent pas les unes les autres. Nous pouvons imaginer un cours où il est souhaitable d'utiliser tous les types d'exercices sur une même simulation. Comme l'espace scénario de chaque système n'utilise que des simulations faites avec ce même système, il est nécessaire de reconstruire la simulation avec chaque système (Partie gauche Figure 3-12). Mais si le processus de développement sépare clairement d'un côté la construction de la simulation, et de l'autre côté la construction du contrôle pédagogique, pourquoi restreindre l'usage de la simulation à un seul type de contrôle pédagogique ? Comme indiqué dans la Figure 3-12, nous proposons de rendre l'espace scénario indépendant du système avec lequel la simulation a été construite.

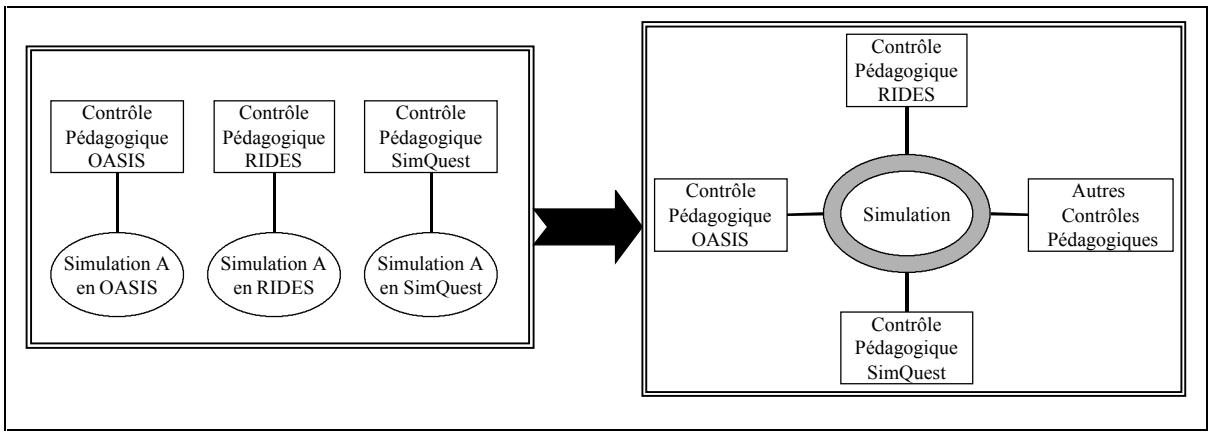


Figure 3-12: Indépendance de l'espace scénario et du système de simulation

Cette indépendance suppose que la simulation fournisse un ensemble de services suffisants pour tous les types de contrôles pédagogiques. Ces services seront utilisés par les espaces scénario des systèmes de réalisation de simulations pédagogiques. Cette approche a été utilisée avec succès dans d'autres domaines, tels que les bases de données par exemple : l'interface JDBC de Java permet aux programmes Java d'utiliser différents systèmes de bases de données relationnelles, en utilisant une seule interface et en permettant de remplacer un système par un autre.

Parmi les avantages de l'indépendance de l'espace scénario, nous pouvons mentionner :

- La possibilité d'appliquer des approches différentes de contrôle pédagogique à une même simulation.

- La réutilisation des simulations entre systèmes
- La possibilité d'effectuer le contrôle pédagogique à distance et de façon synchrone.

Quels sont les services nécessaires pour l'environnement auteur ? Pour l'environnement apprenant ? Comment peut être gérée la connexion entre la simulation et l'espace scénario? Dans les chapitres 7 et 8, nous présentons notre étude de ces questions, les résultats obtenus et une proposition pour les interfaces logicielles.

4 EXPERIMENTATION DE SYSTEMES DE REALISATION DE SIMULATIONS PEDAGOGIQUES BASES SUR MARS

Jusqu'ici nous avons présenté la problématique de réalisation de simulations pédagogiques et les réponses données par des systèmes existants. Nous sommes convaincus que l'intégration des simulations dans les pratiques pédagogiques quotidiennes des enseignants et formateurs dépend absolument de la maîtrise qu'ils peuvent avoir des systèmes de développement. Il convient donc, pour avoir un panorama complet, d'étudier aussi l'utilisation de ces systèmes par les enseignants et les formateurs. Pour ce faire, nous allons présenter deux expérimentations conduites avec MELISA et OASIS, les logiciels développés par notre équipe. Finalement, nous analyserons les résultats de ces expériences et nous présenterons quelques propositions déduites de cette analyse.

Le but fondamental de ces expériences était d'évaluer les difficultés que les auteurs rencontrent. Pour cela, nous avons demandé à des auteurs, représentatifs de ceux ciblés par ces systèmes, de produire des simulations et nous avons observé les démarches suivies et les résultats obtenus. La première expérimentation a eu lieu dans un contexte de formation interne en entreprise et l'autre dans un contexte académique universitaire. Dans le premier cas, nous avons travaillé avec le TPEC (*Technical Planning and Education Center*, L'Isle d'Abeau - France), le principal centre européen de formation de l'entreprise Hewlett-Packard. Pour le contexte universitaire, nous avons travaillé à l'Université Joseph Fourier, avec des enseignants de différentes spécialités, dans le cadre du projet européen ARIADNE (1996 – 1998).

Ces deux contextes diffèrent sur plusieurs aspects : les caractéristiques des auteurs, les domaines des simulations, les types de simulations à produire et les contraintes du processus de développement.

Nous présentons maintenant les deux expérimentations et leurs résultats.

4.1 Expérimentation dans un contexte de formation en entreprise

Cette expérimentation a été faite par l'équipe ARCADE dans le cadre du projet Melisa (*Methodology and Environment for developing Learning, Instruction and Simulation Applications*, 1994-1997) pour le TPEC de Hewlett-Packard [PER 96], [PER 98]. L'objectif principal a été de fournir des méthodes et des outils aux formateurs du TPEC afin qu'ils puissent créer, modifier et tester rapidement, et à des coûts raisonnables, leurs propres simulations pédagogiques.

Les formations du TPEC s'adressent essentiellement à des personnels techniques et portent sur les matériels produits par Hewlett-Packard : configuration, installation, fonctionnement, maintenance, diagnostics de dysfonctionnements et réparation. Le TPEC est confronté à diverses carences des formations traditionnelles :

- Une partie importante de la transmission de savoir-faire est basée sur la manipulation directe des équipements. Cette approche est difficile car, d'une part, il n'y a pas toujours une machine disponible par participant, et d'autre part, la mise en évidence de certains dysfonctionnements implique de dégrader les machines utilisées dans la formation.
- La formation est soumise à des contraintes diverses comme la disponibilité des formateurs et la présence d'un nombre minimum de candidats, et n'est quelquefois offerte que dans des sessions très distantes dans le temps. Le résultat est qu'au moment de l'intervention du technicien chez le client, la formation est insuffisante ou trop ancienne.
- L'expérience montre que l'efficacité des techniciens sur le terrain est souvent inférieure aux résultats des évaluations faites en fin de formation.

Ces difficultés ont amené le TPEC à envisager l'usage de simulations pédagogiques comme solution à ces problèmes, pour les raisons suivantes :

- Les simulations peuvent être distribuées facilement et rapidement par le réseau interne de l'entreprise; ce qui assure la disponibilité à tout moment de formations actualisées.
- Chaque participant peut disposer de son propre exemplaire de la simulation.
- Il est possible de mettre les matériels simulés dans des situations de panne difficiles à atteindre dans la réalité.
- Une simulation peut permettre d'évaluer effectivement les techniciens, si elle est dotée d'un contrôle sur le comportement de l'apprenant.

Après une évaluation des possibilités existantes de production de simulations, le TPEC a décidé de concevoir et de développer, avec notre équipe, un nouvel environnement auteur de simulations pédagogiques. Cet environnement devait satisfaire des besoins bien définis :

- Les auteurs responsables du développement sont les formateurs eux-mêmes. Ils n'ont pas de connaissances en développement de logiciel, mais ils ont des compétences techniques et pédagogiques.
- Les temps de développement sont liés aux calendriers de diffusion des nouveaux produits, et ne doivent pas dépasser 3 mois.
- Les simulations produites doivent être utilisables pour la formation initiale, pour l'auto-formation, ainsi que pour les évaluations (certifications).
- Le coût de développement d'une simulation doit être comparable aux coûts de préparation des supports pédagogiques traditionnels.

Le projet Melisa a comporté trois phases. La première a été dédiée à l'étude de la problématique et des besoins, puis à la proposition d'une maquette basée sur le modèle MARS. Un prototype opérationnel, MELISA, a été construit dans la deuxième phase. Finalement, une troisième phase a été consacrée à la fiabilisation du prototype, à son expérimentation dans des situations réelles et à l'élargissement de ses fonctionnalités.

Nous allons présenter les objectifs, le déroulement et les résultats de l'expérimentation qui s'est déroulée dans le groupe MediSchool (du TPEC), chargé de la formation sur les équipements médicaux. Des informations plus détaillées peuvent être trouvées dans les références [PER 98], [PER 98a], [GUE 99].

4.1.1 Objectifs de l'expérimentation

Les objectifs de l'expérimentation ont été de:

- Faire spécifier et développer, par les formateurs eux-mêmes, les simulations désirées.
- Évaluer l'adéquation de la méthodologie proposée par MELISA.
- Évaluer le processus de développement pour détecter les aspects à améliorer.
- Tester avec les apprenants les simulations produites.

4.1.2 Description de l'expérimentation

Les caractéristiques des auteurs

Les formateurs de MediSchool ont des compétences techniques approfondies sur les produits et des compétences pédagogiques sur la formation relative à ce type de produits. Ce sont aussi des utilisateurs d'applications de bureautique mais ils ne sont pas formés à la programmation ou au développement de logiciels interactifs.

Le déroulement de l'expérimentation

L'expérimentation a comporté 6 phases (cf. Figure 4-1). La première phase a été la formation des auteurs au système MELISA et à la méthodologie associée. Puis les formateurs ont spécifié leurs simulations, avec la collaboration de notre équipe. Nous avons ensuite étudié les simulations proposées pour évaluer si MELISA représentait une solution adéquate pour leur développement. Dans la quatrième phase, les formateurs ont construit les simulations avec le support de notre équipe. Une fois les simulations testées et affinées, elles ont été utilisées dans des situations réelles de formation. Nous avons pour finir analysé le processus et les résultats obtenus.

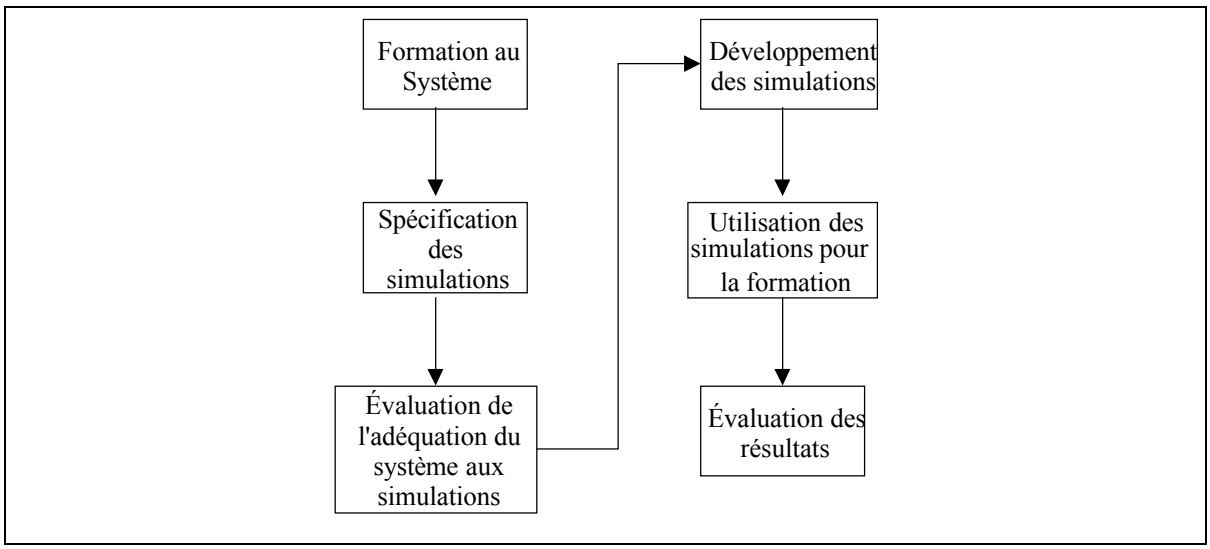


Figure 4-1: Déroulement de l'expérimentation de MELISA

Les simulations

Lors des séances de manipulation réelle des équipements, les participants réalisent deux types de tâches : des opérations de diagnostic, pour déterminer l'état de l'appareil, et des opérations de réparation, pour remettre l'appareil en état de marche. Ainsi pour chaque appareil, il est nécessaire de simuler deux aspects : le diagnostic et le dépannage.

Une simulation portant sur le diagnostic doit permettre à l'apprenant de manipuler l'appareil (boutons, contrôles, prises, connexions, etc.) et d'observer son comportement pour en déduire si l'appareil est en état normal ou en panne. Une simulation de dépannage doit permettre à l'apprenant de monter, démonter, tester, régler et remplacer les composants de l'appareil.

Le contexte de production

Le développement des simulations devait s'effectuer sur une période de trois mois. Les formateurs étaient chargés de tout le processus de développement, y compris la partie graphique, et n'avaient pas d'autre support informatique que notre équipe.

Le développement des simulations pédagogiques

Les phases de spécification et d'analyse de faisabilité ont permis d'établir des différences importantes entre le diagnostic et le dépannage [PER 98],[PER 98a].

Une simulation de diagnostic est spécifique à un équipement qui a ses règles propres de fonctionnement.

Les simulations de dépannage pour différents appareils ont des caractéristiques similaires:

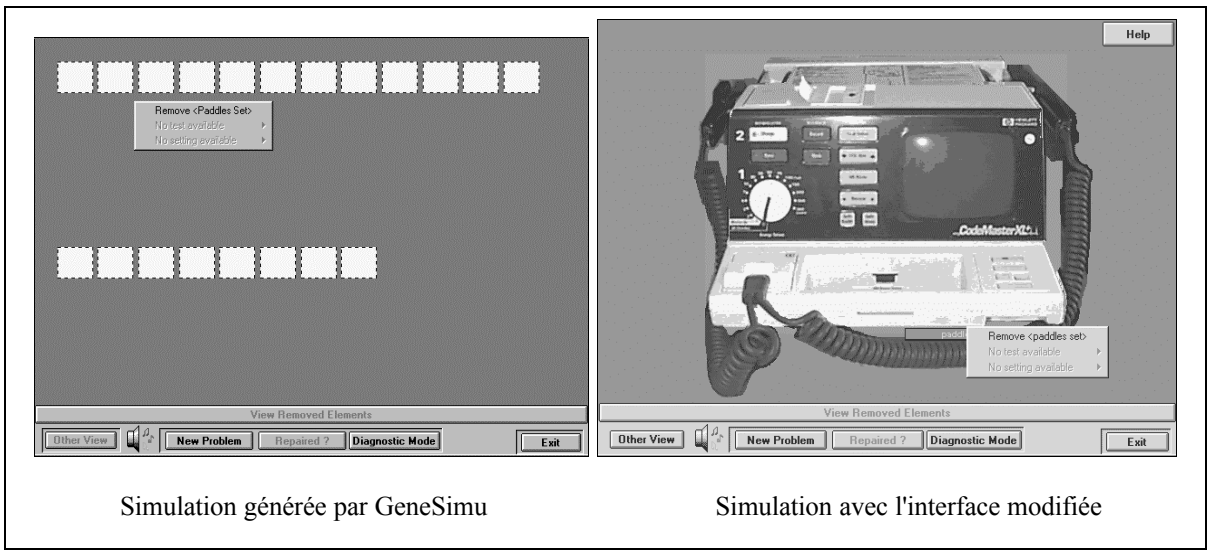
- Le processus de dépannage obéit à des stratégies semblables pour tous les équipements : quand le symptôme de la panne est connu, les méthodes de réparation utilisent le même type de manipulations sur l'appareil.
- La modélisation du dépannage en MELISA implique la saisie de nombreuses informations (plusieurs centaines), ce qui limite la maîtrise de l'application par l'auteur.
- Les interfaces pour l'apprenant doivent être homogènes.

Ces observations ont amené notre équipe à utiliser directement MELISA pour les simulations de diagnostic, et à construire un outil plus spécialisé pour les applications de dépannage.

Le développement des simulations de diagnostic a suivi la méthodologie proposée par MELISA (cf. Figure 2-8). Pour favoriser l'homogénéité des modèles et des interfaces, un ensemble de procédures et de règles a été établi pour préciser comment utiliser MELISA pour ce type de simulations.

Pour le développement des simulations de dépannage et afin d'alléger en grande partie le travail de l'auteur, l'équipe a construit GeneSimu, un système assurant une génération automatique de cette partie de la simulation. L'auteur définit, à l'aide de formulaires, les caractéristiques de l'équipement à simuler: les éléments, les relations de montage/démontage, les dépendances fonctionnelles des éléments, les principaux dysfonctionnements et les procédures de réparation associées. GeneSimu génère alors la simulation avec une interface standard. L'auteur peut ensuite améliorer et personnaliser la présentation obtenue, notamment en remplaçant les éléments d'interface générés automatiquement par des composants plus élaborés (images, photographies des équipements, etc.).

La Figure 4-2 montre un exemple d'interface standard et d'interface améliorée.



Simulation générée par GeneSimu

Simulation avec l'interface modifiée

Figure 4-2: Exemple d'interfaces standard et améliorée

Les simulations de diagnostic et de dépannage d'un appareil peuvent être liées; ceci permet à l'apprenant de passer de l'une à l'autre.

4.1.3 Résultats

Nous allons résumer les résultats de cette expérimentation pour chaque système utilisé et par rapport au point de vue des apprenants [PER 98], [PER 98a].

Les auteurs ont considéré l'approche de GeneSimu comme plus simple que celle de MELISA. Ainsi, la production complète d'une simulation de dépannage a été significativement plus facile que la production d'une simulation de diagnostic.

Dans MELISA, la tâche de modélisation de l'appareil simulé apparaît comme la moins évidente. Néanmoins, l'établissement de procédures et de règles a permis de guider le développement et d'arriver à la production effective des simulations. Ceci a toutefois réduit l'utilisation du système à une partie seulement des fonctionnalités disponibles.

GeneSimu a accéléré le développement des simulations de dépannage. Les auteurs ont consacré leur temps principalement à l'analyse de leurs problèmes et à la collecte des données. La tâche de modélisation a été réduite au minimum car les notions manipulées par les formulaires de GeneSimu correspondent étroitement aux concepts manipulés par les formateurs. Les auteurs ont toujours pu utiliser leurs propres abstractions et non celles imposées par le système.

Les réactions des apprenants ont été favorables. Ils ont trouvé que l'utilisation de ce type de support pédagogique améliorait significativement les formations. Ils ont jugé très favorablement la possibilité de travailler individuellement sur des exercices selon leurs niveaux de compétences. Les principales critiques ont porté sur le besoin de matériel pédagogique complémentaire lié aux simulations, par exemple des hyperdocuments.

4.2 Expérimentation dans le contexte académique universitaire

L'expérimentation d'OASIS a été faite dans le cadre du projet européen ARIADNE (*Alliance of Remote Instructional Authoring and Distribution Networks for Europe*) du programme *Telematic Applications* [FOR 97a], [FOR 97b]. Ce projet a pour but la mise en place de méthodologies et d'outils pour le développement et la gestion de ressources pédagogiques et de curriculums, supportés par des moyens télématiques. L'idée fondamentale du projet est montrée dans la Figure 4-3.

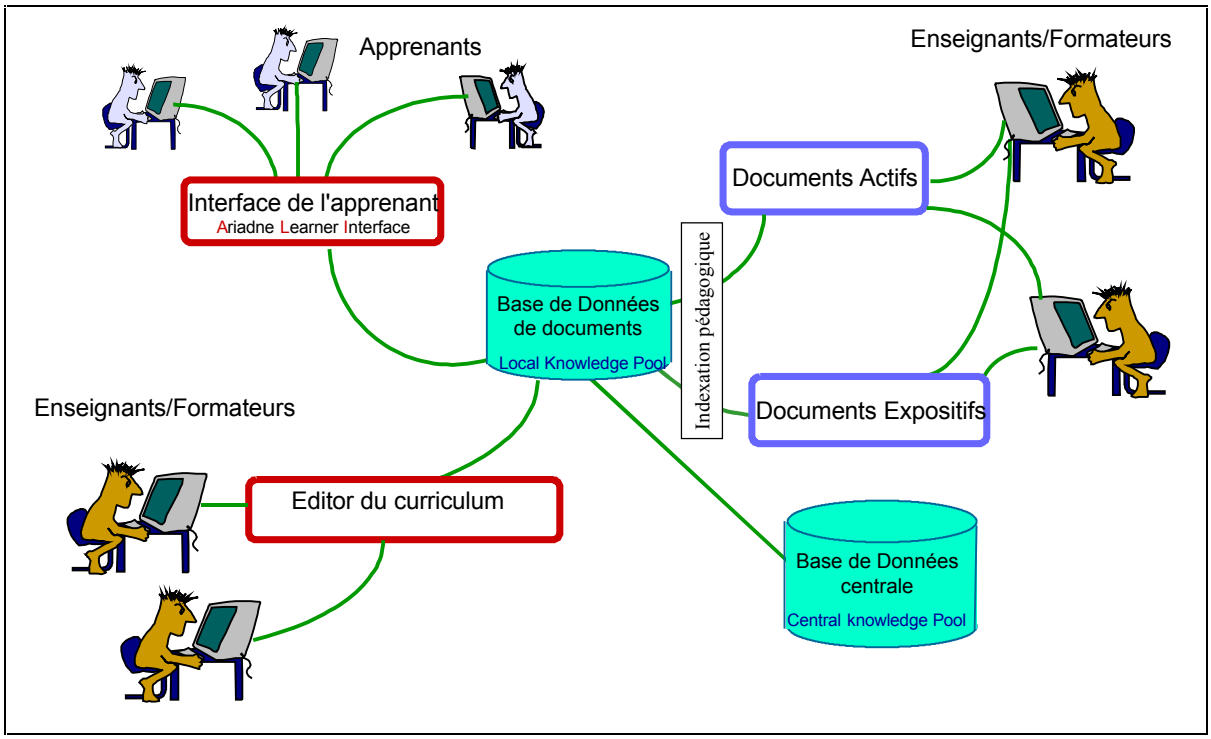


Figure 4-3: Schéma du projet ARIADNE

Les enseignants et formateurs ont à leur disposition des outils pour la création et la modification de deux types de ressources pédagogiques : des documents actifs (simulations, évaluations par QCM, documents hypermédias, documents d'auto évaluation) et des documents "expositifs" (textes, vidéos, transparents de présentations). Ces ressources sont stockées, après un processus d'indexation pédagogique, dans une base de

données distribuée. Les ressources pédagogiques peuvent être récupérées par les formateurs pour construire des cursus complets, avec l'éditeur de curriculums. L'apprenant peut suivre les cursus en utilisant l'interface apprenant d'ARIADNE. Il existe aussi des outils pour la gestion des bases de données locales et pour la gestion de cursus.

Le rôle de notre équipe dans ce projet a été de fournir un outil de construction de simulations pédagogiques et de l'expérimenter.

Après avoir participé à la spécification d'OASIS (espace Scénario), j'ai personnellement assuré l'expérimentation qui s'est déroulée au CAFIM (Centre d'Auto-Formation et d'Innovation Multimédia) de l'université Joseph Fourier. Nous voulions observer si la méthode de développement proposée était adaptée aux besoins des enseignants et s'ils arrivaient à utiliser le système. Nous voulions aussi vérifier si les auteurs devaient, pour utiliser OASIS, avoir des compétences différentes selon le travail à effectuer. Les activités, et les compétences nécessaires, que nous avons établies comme hypothèse sont décrites dans le Tableau 4-1.

Activité	Compétences requises	Niveau
Création du modèle de la simulation	• Connaissances minimales de Toolbook[4] • Connaissances du langage Openscript • Notions de types abstraits • Maîtrise de la méthode proposée par OASIS	avancé
Création de l'interface de la simulation (création et adaptation des objets)		
Adaptation de l'interface (paramétrage des objets) ou Création de l'interface à partir d'objets de la bibliothèque	• Connaissances minimales de Toolbook • Notions de propriétés et méthodes • Connaissances minimales de la méthode proposée par OASIS (la composition du modèle, la composition des objets et le mécanisme des associations)	intermédiaire
Association modèle - interface		
Création d'exercices (scénarios)	• Fonctionnement de la simulation • Éditeur de scénarios d'OASIS.	élémentaire

Tableau 4-1: Compétences des auteurs selon les activités à réaliser

La validation de ces niveaux impliquait de travailler avec des enseignants de profils différents. Le CAFIM nous a permis de contacter un groupe assez varié d'auteurs. Nous

[4] Les systèmes MELISA et OASIS ont été développés en tant que surcouches de ToolBook (d'Asymetrix), environnement initialement choisi comme plate-forme par nos partenaires dans les deux projets. De ce fait,

présentons maintenant le déroulement de l'expérimentation et les résultats obtenus. Les informations complètes sur cette expérimentation sont fournies dans les références [COR 98], [COR 97b].

4.2.1 Objectifs

L'objectif de l'expérimentation était de vérifier, par une étude en profondeur sur un petit échantillon, si OASIS, et la méthode proposée pour le développement de simulations pédagogiques, étaient compréhensibles et utilisables par des auteurs potentiels du milieu académique.

Les objectifs spécifiques de l'expérimentation étaient de:

- Faire spécifier et développer des simulations pédagogiques aux enseignants.
- Évaluer le processus de développement pour détecter les aspects à améliorer.
- Vérifier nos hypothèses de connaissances requises par activité (cf. Tableau 4-1).

4.2.2 Description de l'expérimentation

Les caractéristiques des auteurs

Un groupe de sept enseignants volontaires a participé à l'expérimentation : quatre enseignants de sciences (physique, chimie, géologie) de l'Université Joseph Fourier, deux enseignants du Secondaire (physique-chimie, automatique) et une enseignante en automatique de l'ENSIEG. Leurs connaissances informatiques étaient très variées. Les enseignants de l'UJF n'avaient pas de connaissances en programmation et deux d'entre eux seulement avaient des notions de ToolBook. Les enseignants de lycée avaient des connaissances de base en programmation et étaient des utilisateurs expérimentés de ToolBook. L'enseignant d'automatique était un programmeur expérimenté avec des notions sur l'approche orientée par objets et n'avait pas de connaissance de ToolBook.

---

nous nous sommes appuyés sur cet environnement pour proposer certains composants de nos systèmes (en particulier, le mécanisme de création d'interfaces ainsi que le langage de programmation OpenScript).

Quelques-uns des enseignants avaient déjà développé des simulations en utilisant des langages de programmation ou des outils comme ToolBook. Les autres n'avaient pas encore envisagé l'utilisation de simulations dans leurs pratiques pédagogiques.

Ainsi, nous avions un groupe d'enseignants avec une variété de spécialités et divers niveaux de connaissances en informatique.

Le déroulement de l'expérimentation

L'expérimentation a été réalisée en deux parties: formation des enseignants et développement des simulations. La Figure 4-4 montre les activités réalisées pendant l'expérimentation et la répartition des responsabilités.

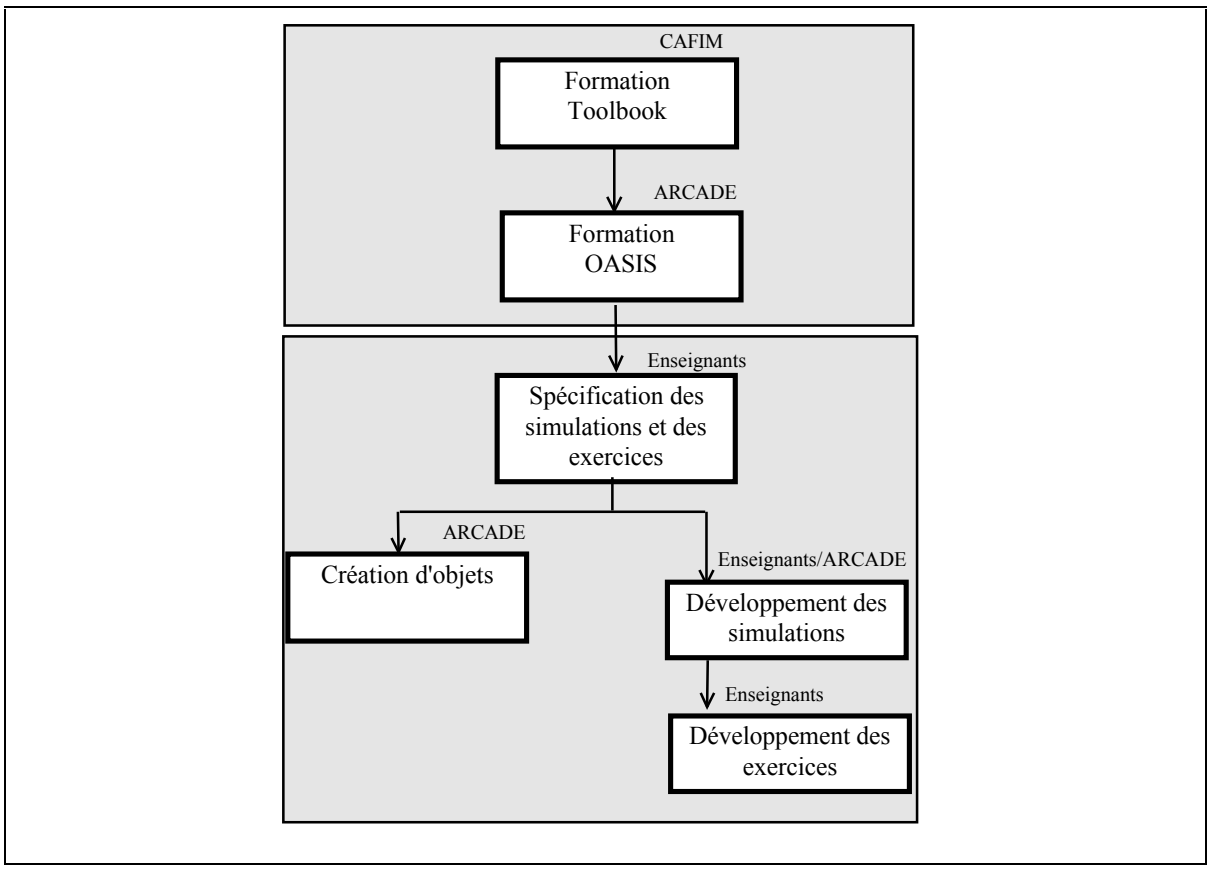


Figure 4-4: Déroulement de l'expérimentation d'OASIS

Les simulations

Dans cette expérience, aucune hypothèse ne pouvait être émise sur le type de simulation à développer. Chaque auteur avait ses propres besoins pédagogiques à satisfaire. L'unique facteur commun était le caractère scientifique des domaines.

Le contexte de production

Dans ce cas, les auteurs n'avaient pas de contraintes sur la durée de développement. Ils étaient plus intéressés à explorer les apports potentiels de ce type de logiciels dans leurs activités pédagogiques. Au contraire des formateurs de Hewlett-Packard, le développement des simulations pédagogiques n'était pas une priorité, ni une obligation. Un facteur important était la faible disponibilité de ces enseignants qui avaient leurs responsabilités habituelles et n'étaient que de simples volontaires pour cette expérimentation.

Ce contexte n'est pas loin des conditions réelles dans lesquelles un enseignant utilise des outils informatiques dans ses pratiques quotidiennes.

La formation

Étant donné un groupe d'enseignants si varié, il m'a été nécessaire d'assurer une formation spécifique pour chaque niveau d'auteur.

Les quatre enseignants de l'U.J.F. ont suivi une formation à ToolBook et aux notions de programmation en ToolBook.

Les enseignants de l'U.J.F. et les deux enseignants de lycée ont ensuite suivi une formation à OASIS.

Après ces formations, un seul enseignant de l'UJF et les deux enseignants de lycée ont abordé le développement des simulations. Les autres enseignants ont abandonné l'expérience par manque de temps ou parce qu'ils n'ont pas trouvé un problème d'apprentissage justifiant l'usage d'une simulation.

Avec l'enseignante de l'ENSIEG, j'ai suivi une démarche différente, car il s'agissait d'une programmeuse expérimentée et elle voulait développer avec OASIS deux simulations qu'elle avait déjà réalisées dans un autre environnement. Je l'ai directement formée à OASIS avec le développement d'un exemple concret et elle a ensuite développé une simulation.

Le développement des simulations pédagogiques

J'ai été disponible pour les auteurs pendant tout le processus de développement. Le déroulement de l'expérimentation et l'assistance fournie sont décrits ci-dessous.

- Le travail de *spécification de la simulation* à développer a commencé pendant la première séance de formation à OASIS. Après la présentation de la notion de simulation pédagogique, les enseignants ont réfléchi aux problèmes d'apprentissage de leur domaine pour identifier ceux auxquels des simulations pourraient apporter des solutions. J'ai fourni aux enseignants un guide d'analyse pour les aider à spécifier leur simulation, avec comme exemple la spécification de la simulation développée pendant la formation. Ayant choisi une simulation, ils l'ont spécifiée selon la méthode d'OASIS en remplissant notre formulaire de spécification. L'annexe 1 contient les supports utilisés.

- *La spécification des objets d'interface* : pour chaque simulation, j'ai déterminé, avec l'enseignant, les objets nécessaires pour l'interface. Dans un premier temps, j'ai identifié les objets de présentation qui pouvaient s'appuyer sur les objets de la bibliothèque d'OASIS. Nous avons ensuite repéré ensemble les objets complémentaires à construire et spécifié pour chacun ses fonctionnalités et son interface.

- *Le développement des objets d'interface* : pour diminuer le temps de développement, j'ai assuré moi-même la création des nouveaux objets de présentation.

- La *définition du modèle*, la *création de l'interface* et la *définition des associations* ont été faites avec plus ou moins d'assistance de ma part, selon le niveau de chaque auteur.

- Le *test de la simulation* a été effectué par les enseignants, qui ont vérifié en même temps le modèle et l'interface.

- Pour la *définition des scénarios*, les enseignants ont réfléchi aux types d'exercices qu'ils pouvaient proposer avec l'outil de scénarios. Ils ont ensuite défini, réalisé et testé un ou deux exercices.

4.2.3 Résultats

L'expérimentation a permis de produire des simulations et de dégager des conclusions sur l'outil OASIS et la méthodologie associée.

Les simulations produites

Un point important est que certains de nos enseignants ont effectivement utilisé les simulations produites dans leur enseignement. Nous donnons ici une brève description des simulations.

Pile de Daniell : l'objectif est d'enseigner la procédure correcte pour mesurer la force électro-motrice d'une pile de Daniell. Cette pile a trois solutions : une de cuivre, une de zinc et une de référence, chacune avec une électrode. L'apprenant peut changer la température ambiante, ainsi que la concentration des solutions de cuivre et de zinc. Il dispose aussi d'un voltmètre pour mesurer les potentiels entre deux électrodes quelconques. La Figure 4-5 montre l'interface de la simulation.

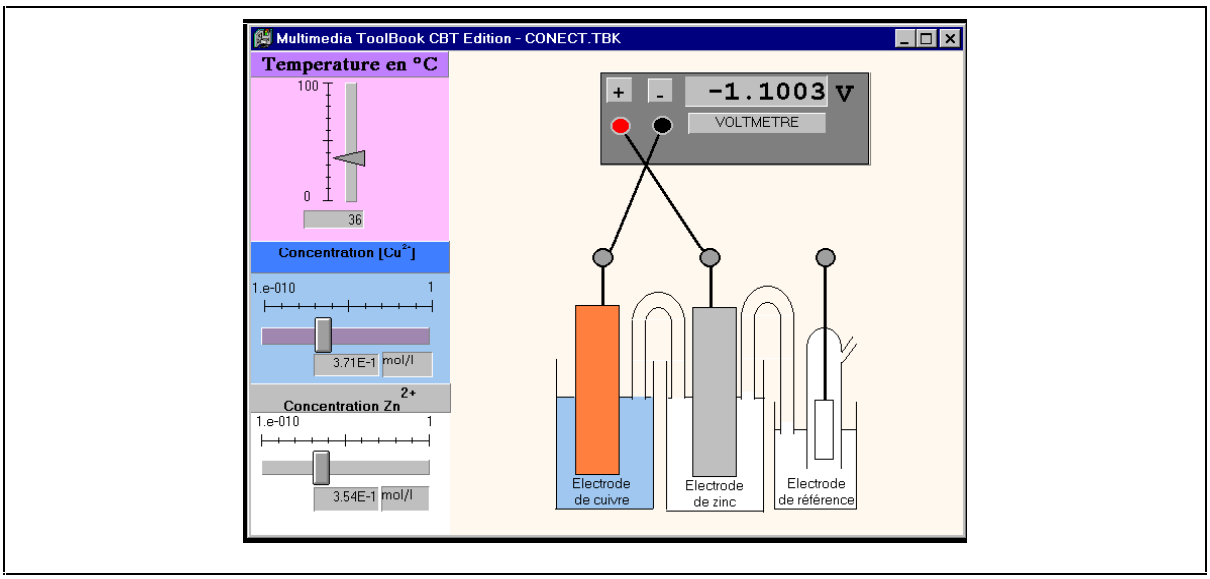


Figure 4-5: Interface de la simulation de la Pile de Daniell

Théorème de Shannon : l'objectif est d'illustrer le théorème de Shannon sur l'échantillonnage de signaux. Dans la simulation, on considère une voiture qui roule à vitesse constante. Chaque roue comporte un rayon visible. L'expérimentation consiste à échantillonner l'angle que forment ces rayons par rapport à l'horizontale, et à montrer la position de la voiture selon l'échantillonnage. L'apprenant peut changer la période d'échantillonnage, pour chercher celles qui permettent de reconstruire le signal original. La Figure 4-6 montre l'interface de la simulation.

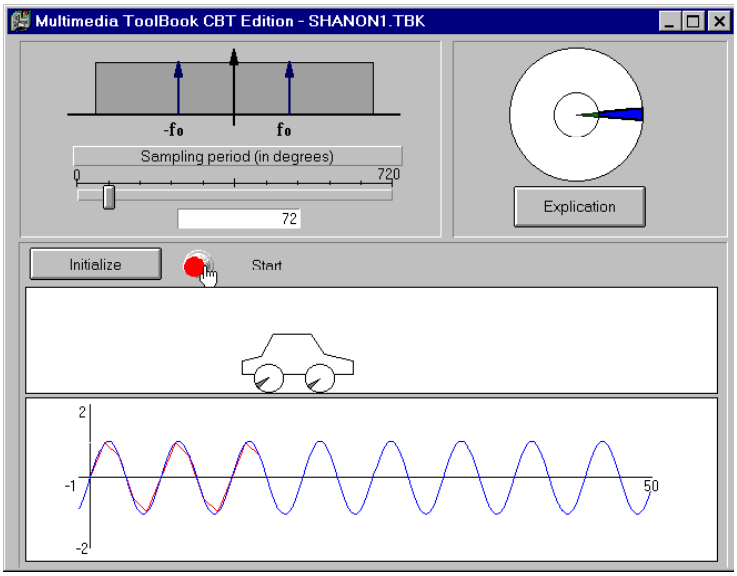


Figure 4-6: Interface de la simulation du Théorème de Shannon

Machine pour le dosage automatique : l'objectif est de permettre à l'apprenant de contrôler une machine de dosage automatique qui comporte les éléments suivants : un réservoir avec une vanne de sortie (VS), une vanne d'entrée (VE) d'un produit X, un tapis (M) qui transporte des briquettes solubles, un malaxeur (B), un système de chauffage, un thermomètre qui mesure la température du réservoir, un capteur (cv) qui s'active si le réservoir est vide et un capteur (cp) qui s'active si le réservoir est plein. L'apprenant peut, grâce aux commandes du clavier ou avec la souris, actionner les interrupteurs des vannes, du malaxeur et du moteur du tapis et agir sur la résistance du système de chauffage. L'apprenant peut ainsi valider les programmes qu'il écrit pour cette machine. La Figure 4-7 montre l'interface de la simulation.

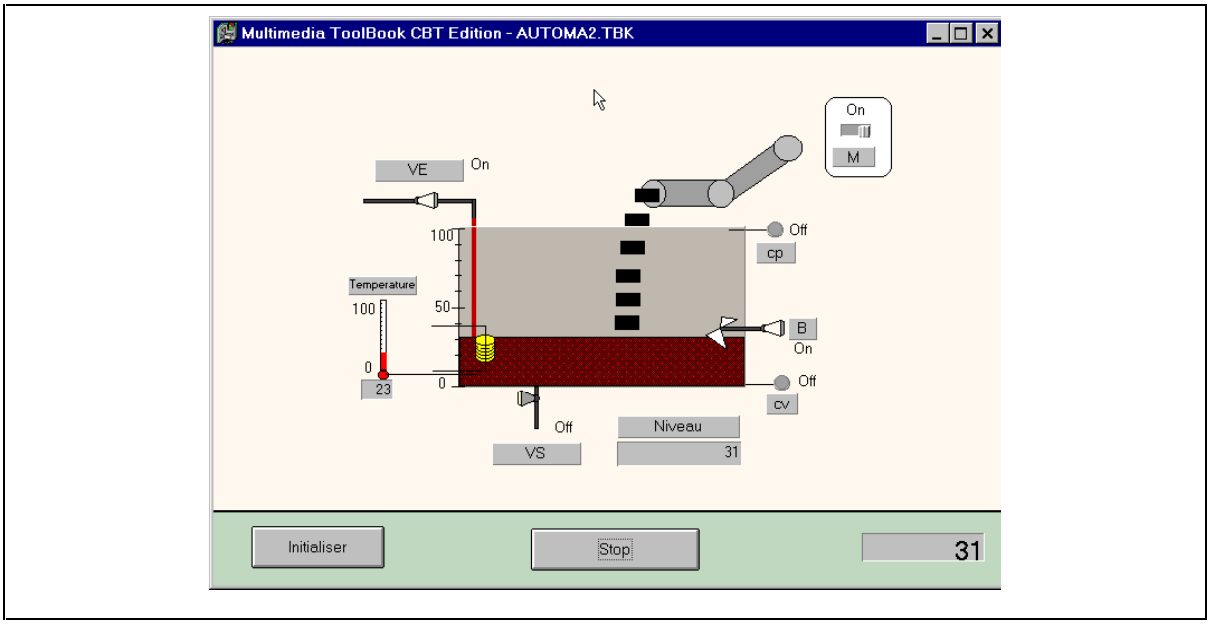


Figure 4-7: Interface de la simulation de la machine pour le dosage automatique

Asservissement de la position d'une bille sur un rail : l'objectif est d'étudier différents types de régulateurs (ou correcteurs numériques). Un régulateur permet de changer l'angle d'inclinaison du rail par rapport à l'horizontale. Le but consiste à déplacer la bille d'une position initiale à une position finale, en changeant successivement l'angle d'inclinaison. Les commandes envoyées par le régulateur au rail dépendent du type de régulateur choisi. L'apprenant peut définir la position initiale et la position finale de la bille, et choisir le type de régulateur ainsi que les paramètres d'échantillonnage.

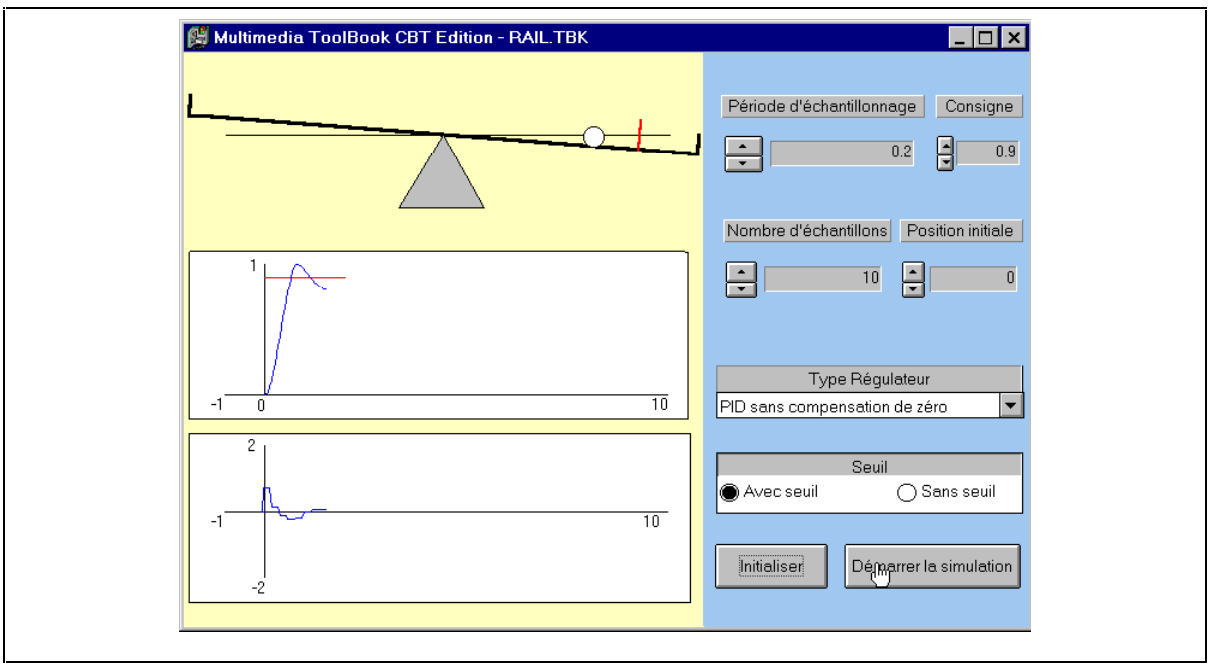


Figure 4-8: Interface de la simulation de l'asservissement de la position d'une bille sur un rail

Conclusions sur l'outil et la méthodologie

Nous allons synthétiser les observations que nous avons faites pendant l'interaction avec les auteurs et le développement des simulations. Nous avons classifié les observations par sujet.

Identification des difficultés de la méthodologie

Le suivi en détail du processus de développement de chaque simulation a permis de repérer les aspects de la méthodologie qui ont été difficiles pour les auteurs.

L'activité de conception des modèles de simulation a été la plus complexe. Les auteurs maîtrisent les formalismes propres à leurs domaines, mais l'expression des modèles dans les formalismes MARS n'a pas été évidente. Les abstractions requises pour cette activité nécessitent une formation spécifique. L'expérimentation confirme que cette tâche de modélisation ne doit être confiée qu'à des auteurs de niveau avancé (cf. *Tableau 4-1*).

Une deuxième source de difficulté a été la séparation entre l'interface et le modèle. Les enseignants ont tendance à penser que l'interface est le système simulé. A nouveau, un processus d'abstraction est nécessaire pour arriver à séparer les aspects purement de présentation et les aspects concernant le comportement du système simulé. Ceci représente un effort pour les auteurs (surtout pour ceux qui avaient déjà une expérience de construction de simulations), même s'ils en reconnaissent ensuite les bénéfices.

La troisième source de difficulté a été la définition des fonctionnalités objets de présentation à créer. Une simulation peut avoir besoin d'un objet de présentation qui n'est pas dans la bibliothèque d'objets d'OASIS. La définition des fonctionnalités de l'objet désiré n'a pas été facile pour l'enseignant car il devait généraliser le comportement de l'objet et trouver les paramètres appropriés.

Une dernière remarque porte sur la visible, mais surprenante, difficulté qu'avaient les enseignants à imaginer des exercices. Non pour des raisons techniques, l'environnement permettant aisément la construction d'exercices, mais plutôt pour des raisons pédagogiques, les enseignants éprouvant de réelles difficultés à imaginer une exploitation pertinente des simulations : quels buts fixer aux élèves ? Quels contrôles effectuer ? Comment enchaîner les exercices ? etc. Après réflexion, nous pensons que le cadre de l'expérimentation, basé sur la participation d'enseignants volontaires pour expérimenter un

outil de production, a beaucoup influé sur ce résultat. En effet, si les enseignants imaginaient facilement des simulations dans leur domaine, ils ne s'étaient pas encore réellement posé le problème de leur utilisation par leurs élèves seuls, en auto-formation. Ceci les a amenés à beaucoup s'interroger sur leurs pratiques pédagogiques. Les simulations pouvaient pour eux constituer un plus, qu'il leur fallait encore évaluer. Leur idée était souvent d'introduire la simulation (libre) dans un cours, et d'explorer en direct avec leurs élèves son utilisation. Notre hypothèse est qu'à l'issue d'une telle démarche exploratoire, ils auraient vu l'intérêt de proposer des exercices et auraient pu facilement en spécifier.

Identification des avantages de la méthodologie

La méthodologie proposée a eu plusieurs effets favorables sur le processus de développement. La séparation du modèle, de la présentation et des scénarios a permis de localiser très facilement les sources des problèmes dans la phase de test de la simulation. La séparation du modèle et de la présentation a permis d'étendre le modèle sans changer ou modifier les objets de présentation et vice-versa. La méthodologie a imposé une construction incrémentale de la simulation, ce qui a permis aux auteurs de maîtriser facilement leurs applications.

Un autre aspect positif de la méthodologie a été la possibilité de travailler en parallèle. Tandis que les auteurs développaient les modèles et testaient l'interface en utilisant des objets simples de la bibliothèque d'OASIS, nous développions de nouveaux objets spécialisés. Nous avons ensuite utilisé une fonctionnalité d'OASIS pour remplacer les objets initiaux par les nouveaux objets.

Suggestions pour l'amélioration du système

L'expérimentation a permis de trouver quelques erreurs non détectées dans les tests initiaux du système. Par ailleurs, les enseignants ont suggéré des améliorations du système et des fonctionnalités souhaitables. Quelques-unes de ces propositions ont été implémentées, par exemple des modifications de l'interface. Parmi leurs autres propositions, citons la construction d'un débogueur basé sur la méthodologie, la génération automatique d'aides pour l'apprenant, et la génération automatique d'exercices en utilisant des modèles d'exercices définis par l'auteur.

Validation des niveaux de développement des auteurs

Pour le niveau élémentaire, nous avons validé notre hypothèse : avec une formation minimale à l'éditeur de scénarios, un auteur est capable d'ajouter des exercices sur une simulation déjà développée. Pour le niveau intermédiaire, nous avons trouvé que des connaissances de ToolBook et de programmation ne suffisent pas; il faut aussi des notions sur les types abstraits de données ou sur l'approche par objets. Pour le niveau avancé, il ressort qu'une expertise de modélisation dans la spécialité est indispensable pour la réalisation du modèle.

Formation à la méthode et à l'outil

Au vu des résultats obtenus, nous avons défini de nouveaux contenus pour les différents niveaux de formation (cf. Figure 4-9).

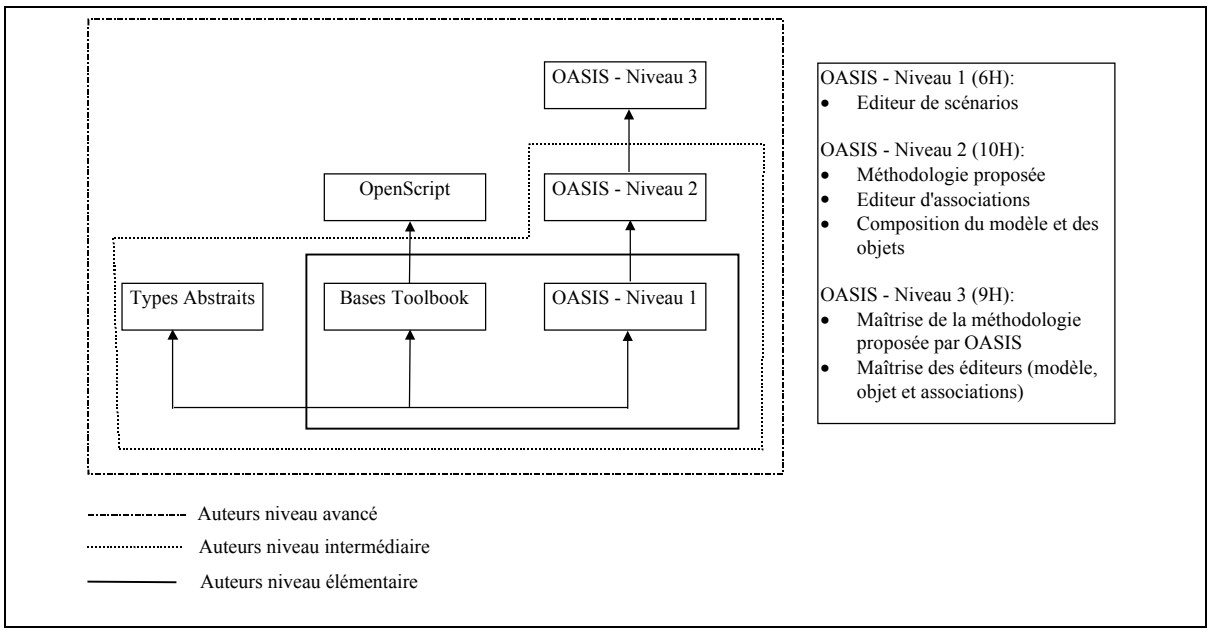


Figure 4-9: Contenus des formations OASIS par niveau d'auteur

4.3 Conclusions des expérimentations

Nous venons de présenter les résultats des deux expérimentations faites avec les systèmes de réalisation de simulations pédagogiques basés sur MARS. Nous pouvons en dégager les conclusions suivantes :

- Les systèmes ont permis le développement effectif de simulations dans les deux contextes.
- Les systèmes ont permis de réduire les temps de développement. Les auteurs qui avaient déjà réalisé des simulations avec d'autres outils ont réduit leurs temps de développement.
- La maîtrise des systèmes par les auteurs dépend fortement de leurs connaissances en informatique. Si cet aspect est crucial quand l'auteur doit maîtriser l'ensemble du système, il est moins important quand les tâches des auteurs sont réduites à la définition du contrôle pédagogique et à la définition de l'interface. De ce fait, les systèmes sont utilisables par des auteurs de différents niveaux.
- Nous avons constaté que la modélisation du système simulé est la tâche la plus difficile pour les auteurs. Il est nécessaire d'avoir des capacités d'abstraction importantes. La solution fournie à ce problème par GeneSimu nous semble très intéressante. GeneSimu propose en effet des interfaces qui correspondent directement aux formalismes utilisés par les auteurs dans leurs domaines spécifiques. Ceci facilite la tâche de modélisation. C'est dans ce sens que nous faisons des propositions plus génériques dans le paragraphe 4.4.
- L'approche méthodologique a été un facteur important pour aboutir à la réalisation des simulations. La séparation Modèle-Représentation-Scénario imposée par la méthodologie a favorisé le développement incrémental, la mise au point des simulations, les tests et la production en parallèle.

4.4 Notre Proposition

Je voudrais ici proposer une solution possible à la difficulté profonde rencontrée par les auteurs dans la tâche de modélisation du système simulé.

Analysons d'abord quels étaient les facteurs qui ont rendu cette tâche complexe pour les auteurs. Pour l'expérimentation chez Hewlett-Packard, il a été évident que la quantité de données nécessaires pour modéliser avec MELISA les simulations de dépannage constituait un facteur de complexité. Dans le cas des simulations de diagnostic, les auteurs

ont eu des difficultés pour "traduire" le problème selon les formalismes fournis par MELISA.

Pour l'expérimentation dans le contexte académique, le problème fondamental pour la modélisation a été l'abstraction qu'exigent les formalismes d'OASIS. Encore une fois, l'auteur arrive normalement à bien exprimer son problème avec les formalismes habituels de son domaine, mais la traduction dans les formalismes du système a été le facteur critique de la modélisation.

Nous constatons que, dans la tâche de modélisation, il y a un processus d'abstraction qui n'est pas évident pour les auteurs. Les formalismes de MARS demandent une traduction des problèmes en termes de propriétés, méthodes et dynamique qui ne peut pas être réalisée par un auteur sans une formation spécialisée (cf. Figure 4-10).

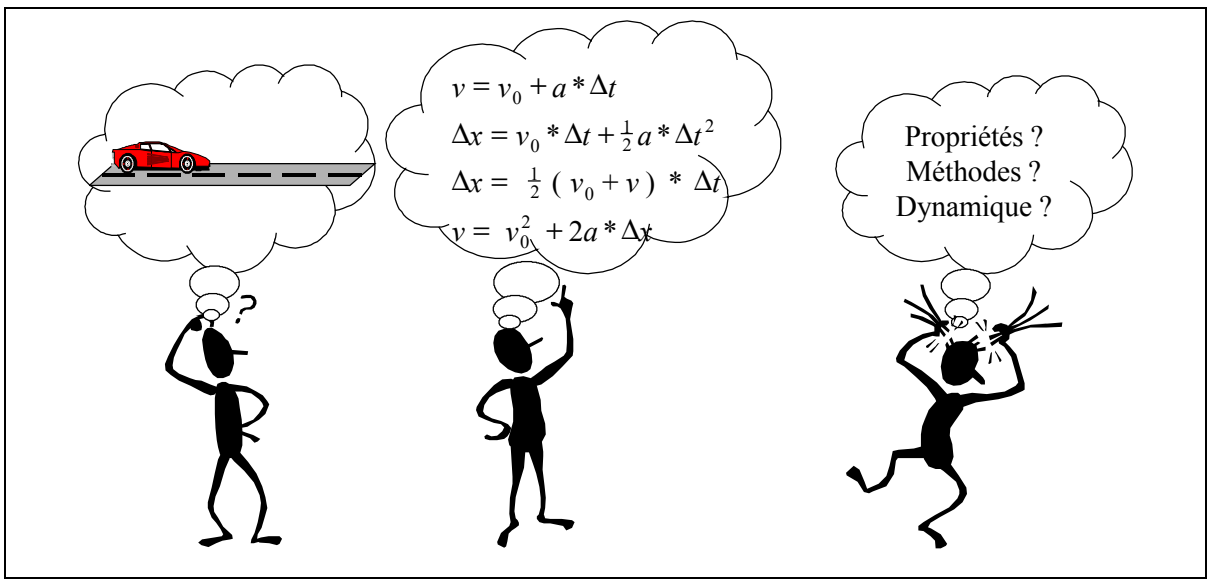


Figure 4-10: Processus d'abstraction avec MARS

Dans les expérimentations, nous avons adopté trois solutions différentes à ce problème :

- Pour les simulations de dépannage, construire un système spécialisé pour ce type de simulations.
- Pour les simulations de diagnostic, définir un ensemble de règles et de procédures pour l'utilisation du système. Ceci a été possible car il existait une certaine homogénéité entre ces simulations.
- Pour les simulations des enseignants, fournir un support personnel à chacun. Dans les cas où l'auteur ne possédait pas les compétences requises, c'est nous

qui avons finalisé le modèle et le rôle de l'auteur s'est alors limité au test du modèle.

La solution de GeneSimu s'est montrée la plus efficace et la mieux acceptée par les auteurs. Regardons en détail le processus de construction de cet outil.

Le premier pas vers la définition de GeneSimu a été la constatation de trois faits :

- Difficulté à gérer, avec les interfaces fournies par MELISA, la quantité de données nécessaire pour des simulations de dépannage.
- Besoin d'homogénéiser l'interface de l'apprenant pour toutes les simulations de dépannage.
- Existence de règles et de procédures de réparation standards, indépendantes de l'équipement à réparer.

Le deuxième pas a été l'identification précise des aspects partagés par toutes les simulations de dépannage. Ceci a permis d'établir un formalisme pour la description de ce type de problèmes, un vocabulaire commun, et des formulaires permettant d'exprimer les caractéristiques du système à simuler. Les auteurs ont participé activement à ce processus.

Le troisième pas a été la réalisation de GeneSimu. Une partie du développement a été la construction d'interfaces ressemblant aux formulaires définis avec les auteurs. L'autre partie a été consacrée à la construction du générateur de simulations. MARS a été utilisé comme modèle de base et certaines fonctionnalités de MELISA ont été réutilisées; ceci a permis un développement rapide du générateur.

Par ailleurs, l'étude des simulations produites pendant l'expérimentation et les tests d'OASIS ont révélé la présence implicite de patrons dans la modélisation en MARS de certains types de simulations. Nous avons constaté par exemple que les simulations basées sur des équations mathématiques avec une variable indépendante avaient la même structure en termes de MARS : une propriété MARS pour chaque variable, une méthode pour chaque variable modifiable par l'apprenant, et un graphe d'états similaire.

Partant de ces observations, je propose de reproduire le type de solution de GeneSimu pour des familles de simulations bien identifiées. Il s'agit de produire un système spécialisé où les interfaces offrent directement les notions et concepts propres à une famille et de générer

automatiquement les simulations en se basant sur MARS. Le processus générique pour la production d'un système spécialisé est décrit dans la Figure 4-11.

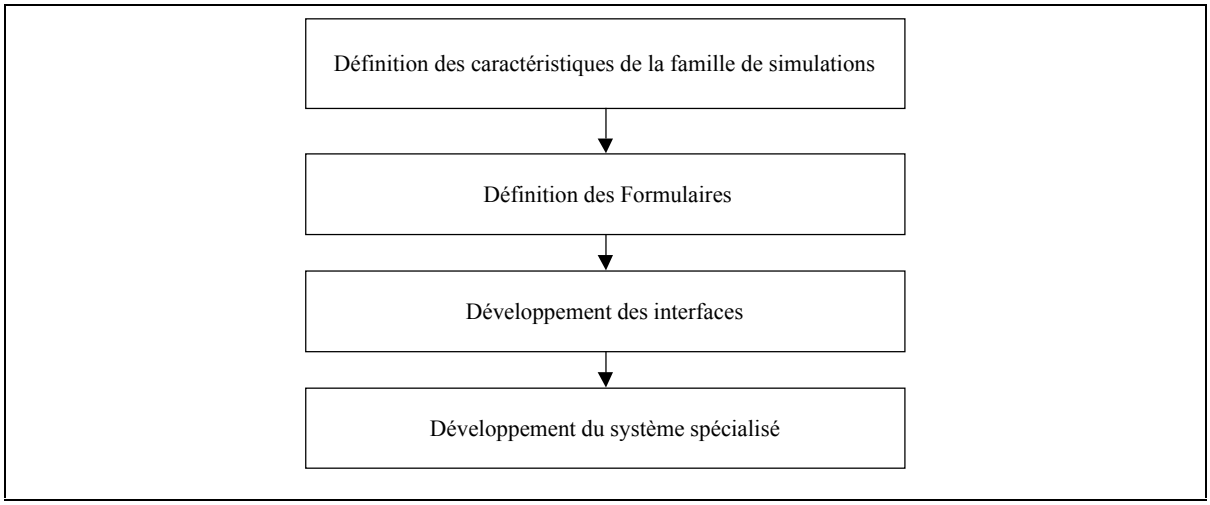


Figure 4-11: Processus de développement d'un système spécialisé

L'idée est de réutiliser le savoir-faire et les fonctionnalités d'OASIS pour développer rapidement des systèmes adaptés aux domaines des auteurs ou aux types de simulation qu'ils produisent.

J'ai entrepris le développement d'une telle solution en utilisant les techniques des frameworks et patrons. Les résultats de ce travail constituent la deuxième partie de cette thèse.

Partie 2
Systèmes dédiés à la production de simulation pédagogiques et adaptés aux domaines : Proposition d'un framework

Nous avons constaté le besoin d'adapter l'outil de construction de simulation aux domaines spécifiques des auteurs. Notre objectif est de permettre de développer rapidement des outils de production de simulations adaptés aux domaines des auteurs. Pour cela, je propose dans cette partie un framework basé sur le modèle MARS.

Le chapitre 5 présente brièvement les techniques de réutilisation dans les systèmes orientés objets aux niveaux architecture, conception et implémentation. Ces techniques seront ensuite utilisées pour la conception et le développement de notre framework.

Le chapitre 6 décrit le framework proposé, présente le processus de développement utilisé et décrit deux outils réalisés avec ce framework.

5 ÉLEMENTS DE REUTILISATION

Démarrer à partir de rien le développement de systèmes complexes s'avéré une approche coûteuse et longue. De nos jours, le développement de logiciels passe de plus en plus par la conception, la réutilisation et l'assemblage de composants. Les logiciels de grande taille se composent de plusieurs sous-systèmes, tels que des dépôts de données, des interfaces utilisateur, des noyaux de communication, des systèmes de sécurité, etc. La réutilisation d'éléments de conception, ainsi que de code existant, est une technique efficace pour rendre les projets plus rentables et moins sujets à erreurs.

Dans ce chapitre, je présente les éléments que j'ai utilisés pour concevoir et développer le framework qui sera décrit dans le chapitre suivant.

5.1 Pourquoi la réutilisation ?

Depuis les années 60, le génie logiciel essaie de résoudre les problèmes de la célèbre "*Crise du Logiciel* " :

- Les coûts élevés de développement, qui dépassent toujours les prévisions
- Le non-respect des délais
- La faible qualité des logiciels
- Les coûts élevés de maintenance

Dans son article *"Mass-produced software components"* [MIL 68], présenté en 1968 à la conférence de Génie Logiciel de l'OTAN, McIllroy faisait naître un espoir. Il décrivait une bibliothèque de composants réutilisables dotés de fonctionnalités d'adaptation. L'aspect clé de cet article a été de permettre d'envisager le développement de logiciels comme une industrie capable de réaliser des produits à partir d'un ensemble de composants réutilisables. Dès lors, la réutilisation de logiciel a été considérée comme un des facteurs nécessaires pour passer de l'artisanat à l'ingénierie.

La démarche de réutilisation peut être mise en œuvre dans toutes les phases du cycle de vie d'un logiciel. L'architecture peut réutiliser des styles tels que tubes et filtres, architectures en couches, invocation implicite, etc. [SHA 96], [JAC 97], [BAS 98]. La conception peut s'appuyer sur des patrons de conception [BUS 96], [COP 92], [GAM 95]. Pour le code, il existe des bibliothèques et des composants [NIE 92], [COU 96], [KOZ 98]. Plus récemment, sont apparus les frameworks qui permettent de réutiliser en même temps des éléments d'architecture, de conception et de code [JOH 91], [NEM 97], [JOH 97], [ROB 97].

5.2 Niveaux de réutilisation

Un premier niveau de réutilisation porte sur le processus de construction. De nombreux modèles ont été proposés : le modèle en cascade [ROY 70], le modèle en spirale [BOE 88], le modèle orienté prototype [BIS 92], etc. La réutilisation consiste à reproduire ces processus de construction de logiciel, en identifiant des étapes et un ordre d'exécution.

Quelque soit le cycle de vie utilisé pour le développement d'un système, on y retrouve des activités communes: définition de l'architecture, conception et implémentation. Dans ce qui suit, nous présentons brièvement les éléments de réutilisation au niveau de ces activités.

5.2.1 Styles d'architectures : réutilisation au niveau architecture

L'architecture d'un système est l'équivalent du plan de l'architecte pour la construction d'une maison. Elle constitue le document qui guide la réalisation. Elle décrit la structure du système en termes de ses composants et des interactions entre composants. L'architecture du système fixe des décisions importantes, telles que l'affectation de fonctionnalités aux

composants, leurs protocoles d'interaction et des propriétés globales du système comme la performance [SHA 96]. Dans [GAR 95], l'architecture d'un système est définie comme :

> *La structure des composants d'un système, leurs relations, et les principes et les orientations qui gouvernent leur conception et leur évolution.*

Les recherches dans ce domaine visent à fournir aux concepteurs des notations et des outils qui permettent de spécifier et de décrire l'architecture des systèmes [GAR 95a]. De nombreux langages de description d'architectures (*Architecture Description Language* ou *ADL*), ainsi que des outils basés sur ces langages, ont été proposés ces dernières années. Citons par exemple :

- UniCon (Universal Connection Langage) est un langage qui permet de définir une architecture à partir de ses composants ainsi que les connecteurs entre ces composants. UniCon a un outil associé qui permet de compiler une description pour produire le système décrit [ZEL 96].
- Rapide, un langage qui permet de spécifier et de simuler le comportement d'une architecture définie par ses composants et par les messages et les événements qu'ils échangent. Il ne permet toutefois pas de générer automatiquement une application [LUC 95].
- ACME, un langage générique pour la description d'architectures, qui facilite l'échange entre différents *ADLs* et outils [GAR 95b].

Dans la pratique, des styles d'architectures ont été identifiés et utilisés par les développeurs pour décrire des structures de systèmes : architecture client-serveur, architecture en couches, tableaux noirs, etc. Mais l'absence de définitions formelles rend difficile leur expression dans les *ADLs* et leur exploitation par les outils basés sur ces *ADLs*.

Un style d'architecture caractérise une famille de systèmes qui partagent des propriétés structurelles et sémantiques [MON 97]. Le style fournit un vocabulaire pour les composants et les connecteurs (pipes et filtres, serveur/client, …) et une sémantique associée au style (incluant des contraintes). Shaw et Garlan ont compilé un catalogue des styles d'architectures les plus répandus [SHA 96] (voir annexe 2).

Les styles d'architectures sont de fait des éléments de réutilisation. La référence à un style, ou à un ensemble de styles, établit un langage commun de communication entre les concepteurs de l'architecture et les développeurs. Ceci réduit les coûts de maintenance, cinquante pour cent du temps étant habituellement dédiés à la compréhension de l'architecture du système [SHA 95].

5.2.2 Patrons de conception : réutilisation au niveau conception

La réutilisation au niveau conception, dans le développement d'applications orientées objet, a été caractérisée par l'utilisation des patrons (ou schémas) de conception, que Peter Coad définit comme [COA 92] :

> *Une abstraction d'un doublet, triplet ou autre petit groupe de classes qui peut être utile encore et encore dans tout développement orienté objet.*

Un patron de conception décrit une solution générique et réutilisable, prouvée et acceptée, à un problème récurrent de conception. Il identifie les objets, leurs collaborations et la distribution de leurs responsabilités dans la solution. L'objectif principal des patrons est de permettre de réutiliser des solutions génériques proposées et acceptées par des développeurs experts.

Ce sont les écrits de l'architecte Christopher Alexander sur les patrons dans la construction de villes et de bâtiments qui ont inspiré les premiers travaux sur les patrons dans la conception orientée objets. Dans la conférence OOPSLA'87, Beck et Cunningham ont présenté cinq patrons résultant de l'application des idées d'Alexander, pour la conception d'applications en SmallTalk. Après cette conférence, une série d'événements ont contribué au renforcement de cette approche :

- En 1991, Jim Coplien a publié le livre *Advanced C++ Programming Sytles and Idioms*, contenant des solutions d'experts aux problèmes rencontrés par les programmeurs non experts [COP 92].
- En 1991 et 1992, à la conférence OOPSLA ont eu lieu deux workshops sur les patrons.

- En 1993, le groupe, connu comme le *Gang of Four* (Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides), présente une première compilation de patrons à la conférence ECOOP [GAM 93].
- En 1994 a lieu la première conférence sur les patrons *Patterns Languages of Programming*.
- En 1995, le *Gang of Four* publie le livre *Design Patterns - Elements of Reusable Object-Oriented Software,* un catalogue de patrons validés et répandus dans la communauté OO [GAM 94].

Il existe maintenant plusieurs autres catalogues de patrons de conception, par exemple ceux de Coad [COA 95], de Buschmann [BUS 96] et de Lea [LEA 97]. Front présente une comparaison entre les différentes notations utilisées dans ces catalogues pour décrire les patrons [FRO 97]. Le **Tableau 5-1** présente de façon résumée un exemple de patron, l'observateur.

Dans le milieu industriel, ces techniques de patrons de conception sont utilisées depuis quelques années. Dans une compilation d'expériences présentée dans [BEC 96], nous trouvons quelques leçons apprises par les entreprises :

- Les patrons de conception sont un excellent moyen de communication entre développeurs. Ils favorisent la communication effective de concepts complexes.
- Les patrons capturent l'essence des diagrammes de conception et permettent une réutilisation de cette conception dans d'autres projets.
- Les catalogues de patrons permettent de guider les concepteurs débutants vers des solutions appropriées. L'accès aux catalogues encourage ces développeurs à appliquer de bonnes pratiques dans leurs conceptions.
- L'existence dans l'entreprise d'un mentor qui aide à identifier les patrons et à les appliquer correctement accélère l'acceptation des patrons par les développeurs.
- L'identification et la spécification de nouveaux patrons prennent du temps, mais avec l'expérience ce temps diminue.

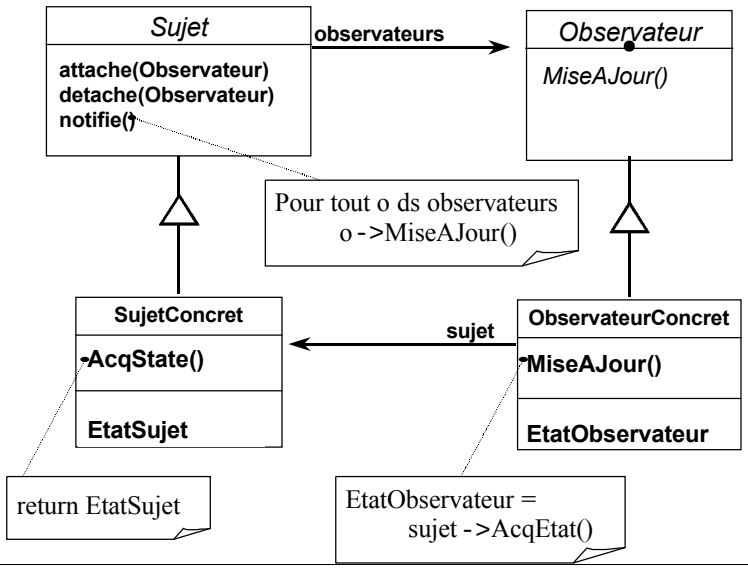
Nom	Observateur
Classification	Domaine: Objet, Rôle : Comportement
Intention	Définition de dépendances un à plusieurs. Quand un objet change d'état, tous ses dépendants sont informés.
Alias	Dépendants, Diffusion - Souscription
Motivation	La partition d'un système en une collection de classes coopérantes implique de maintenir la cohérence entre les objets sans passer par des solutions fortement couplées qui limiteraient la réutilisation. Un exemple typique est le tableur qui a besoin de maintenir la cohérence entre les données dans les feuilles et les différents graphiques (histogrammes, barres, secteurs) basés sur ces données. Les graphiques dépendent des données et ils doivent changer chaque fois que les données changent.
Indication d'utilisation	• Quand un concept a deux représentations, l'une dépendant de l'autre. • Quand la modification d'un objet doit être signalée à d'autres objets dont on ne connaît pas le nombre. • Quand des objets ne doivent pas être fortement couplés.
Structure	
Constituants	*Sujet*: maintient une liste des observateurs et fournit des méthodes pour ajouter et supprimer des observateurs. *Observateur*: définit une interface pour la mise à jour de l'objet observé. *SujetConcret*: mémorise l'état de l'objet et signale aux observateurs quand il y a un changement dans son état. *ObservateurConcret* : maintient une référence à un *SujetConcret*, stocke l'état qui doit être cohérent avec le *ObservateurConcret*, implémente la mise à jour de l'objet.
Collaborations	Le *SujetConcret* signale à ses observateurs quand il change, l'observateur fait une mise à jour de son état pour maintenir la cohérence.
Conséquences	• Ce patron permet de changer indépendamment les *Observateurs* et les *Sujets*. • Il réduit le couplage entre les *Observateurs* et les *Sujets*. • Il supporte la diffusion car une notification est automatiquement envoyée à chaque observateur enregistré. • Une mise à jour peut causer une cascade de mises à jour car les observateurs ne se connaissent pas entre eux. • Le protocole simple de mise à jour ne donne pas des informations détaillées sur les changements, ce qui peut demander plus de travail aux *ObservateurConcrets*.
Utilisations	L'exemple le plus connu de ce patron est le modèle MVC utilisé pour l'interface utilisateur de SmallTalk.
Voir aussi	Mediator, Singleton

Tableau 5-1: L'observateur, un exemple de patron de conception [GAM 94]

5.2.3 Bibliothèques et Composants : réutilisation au niveau implémentation

Après les propositions de McIllroy, les efforts de réutilisation ont donné des résultats principalement au niveau implémentation. De nombreuses bibliothèques ont été proposées, pour des problèmes divers: entrée-sortie, graphiques, traitement de texte, mathématiques, etc. Dans les applications orientées objet, les mécanismes d'héritage et de polymorphisme permettent de réutiliser des classes existantes.

Une autre approche de réutilisation au niveau implémentation est basée sur les composants. Les développeurs insèrent, dans l'application, des composants qui communiquent à travers leurs interfaces logicielles. Ces composants ne sont pas nécessairement développés dans le même langage de programmation que l'application, mais respectent une architecture et fournissent une interface. Des exemples de ces techniques sont les objets COM, les ActiveX et les Java Beans.

Chappell [CHA 98] définit ce type de composants comme *un morceau de logiciel, de taille réduite,* qui :

- est fourni en format exécutable,
- n'est pas exécutable par lui-même, mais s'exécute dans un conteneur (comme un navigateur WEB, Visual C++, etc.),
- est capable de donner au conteneur des informations sur certaines méthodes et propriétés,
- est capable d'envoyer des événements au conteneur,
- fournit une interface à l'utilisateur, interface qui est également manipulable par le développeur.

Si le génie logiciel basé sur les composants est un domaine récent et en évolution, des solutions techniques, comme les JavaBeans et les ActiveX, existent déjà sur le marché et permettent aux développeurs de réutiliser des composants. Ces solutions sont très répandues et sont utilisées quotidiennement [KOZ 98].

5.3 Les frameworks : des usines pour la construction d'applications

Nous venons de résumer les éléments de réutilisation les plus courants à différents niveaux dans le développement orienté objet. Ces éléments sont fortement utilisés dans la conception et le développement de frameworks. Un framework est un squelette d'application pour un domaine particulier, à partir duquel un développeur peut générer des applications. L'architecture, la conception et les composants sont réutilisés dans chaque application produite avec le framework.

5.3.1 Définition

Fayad et Schmidt définissent un framework comme **une application semi-complète, réutilisable, qui peut être spécialisée pour produire des applications adaptées** [FAY 97]. Un framework capture l'architecture d'une famille d'applications pour un domaine particulier; il exprime cette architecture avec un ensemble de classes abstraites et concrètes, et leurs collaborations; et il implémente cette conception dans un langage orienté objet. Un framework n'est pas une simple bibliothèque de classes : il incorpore en plus des connaissances sur la conception de cette famille d'applications et impose un schéma de collaboration entre les classes. Les frameworks appliquent une stratégie dite "d'inversion de contrôle", c'est-à-dire que c'est le framework qui appelle les méthodes définies par l'utilisateur. L'utilisateur (le développeur) spécialise le framework par héritage et par composition des classes.

La conception d'un framework est basée sur l'analyse des aspects stables (ou *frozen spots*) et des aspects variables (ou *hot spots*) des applications à produire.

Nous trouvons des exemples de frameworks dans plusieurs domaines : banque [BAU 97], composants multimédias [POS 97], *Computer Integrated Manufacturing* [DOS 97]), réseaux et communications [SCH 99], logiciels graphiques [RBB 98]; mais les plus connus sont ceux pour le développement d'applications dans des langages de programmation : MacApp sur Macintosh, MFC de Microsoft, AWT de Java, et les produits pour le développement rapide d'applications (Visual C++, Visual Café, C++ Builder, etc.). L'implémentation du modèle *Model/View/Controller* en SmallTalk est considérée comme l'un des premiers frameworks.

5.3.2 Classifications

Les frameworks peuvent être classifiés soit par le type d'applications qu'ils produisent, soit par la façon dont l'utilisateur peut les adapter pour construire une application [FAY 97].

La classification par type d'applications identifie trois catégories de frameworks :

- Les frameworks d'infrastructure de systèmes : frameworks de communication, frameworks pour les interfaces utilisateur, etc.
- Les frameworks d'intégration de logiciels dans les applications réparties: CORBA, systèmes de transactions de bases de données…
- Les frameworks pour des secteurs spécifiques, par exemple l'aéronautique, la fabrication ou les finances. Ces frameworks capturent l'essence de l'activité des entreprises de ce secteur.

La classification selon la technique d'extension permet de situer les frameworks entre deux extrêmes :

- Les frameworks boîte noire : la spécialisation de ces frameworks se fait exclusivement par composition de classes. La fonctionnalité du framework est adaptée à travers la création et l'intégration de composants qui respectent des interfaces définies par le framework.
- Les frameworks boîte blanche : la spécialisation se fait à l'aide des techniques orientées objets classiques, comme l'héritage et la liaison dynamique. L'extension est obtenue par héritage des classes abstraites du framework et par surcharge des méthodes.

Les frameworks boîte blanche demandent une connaissance de la hiérarchie des classes, et aboutissent à des frameworks fortement couplés avec cette hiérarchie et difficiles à utiliser. Les frameworks boîte noire sont, eux, plus faciles à étendre et à utiliser, mais ils sont plus difficiles à réaliser. Normalement, les frameworks ne sont pas totalement boîte blanche ou boîte noire; ils combinent les deux techniques, en donnant la préférence au style boîte blanche pour les hiérarchies de composants et au style boîte noire pour la composition de composants.

5.3.3 Méthodes de construction

Bien qu'il n'existe pas de méthode formelle pour le développement des frameworks, quelques approches ont été tirées de l'expérience. L'approche évolutive propose une construction itérative d'un framework. L'approche constructive propose la construction d'un framework à partir de l'analyse du domaine.

L'approche évolutive

Dans cette approche, proposée par Roberts et Johnson [ROB 97], un framework est développé itérativement à partir du développement successif d'applications du domaine du framework (le minimum conseillé est de trois, afin d'assurer une bonne prise en compte des particularités de ce type d'applications). Chaque application s'appuie sur la précédente, en étendant la solution avec des mécanismes tels que :

- La généralisation en utilisant l'héritage et les patrons de conception
- L'identification et la construction de composants communs aux applications
- L'identification et la séparation des aspects stables des applications (séparation entre le code qui change et le code qui ne change pas).

Ces itérations conduisent à un framework de type boîte blanche.

Pour aller vers un framework de type boîte noire, il est nécessaire de :

- Identifier et construire des objets *pluggables* qui encapsulent des ensembles de classes similaires. Ces objets implémentent les aspects variables du framework.
- Identifier et construire de petits objets, jusqu'au point où une division supplémentaire d'un objet produirait des objets sans signification dans le domaine.
- Organiser les composants en une hiérarchie de composants.
- Favoriser la composition, plutôt que l'héritage, pour combiner les composants.

Avec un framework de type boîte noire, il est possible de construire des langages pour décrire les applications et des outils visuels pour construire ces applications.

L'approche constructive

Cette approche est celle suivie par l'entreprise Taligent pour la construction de plusieurs frameworks d'infrastructure de systèmes [NEM 97]. Le développement d'un framework suit les étapes classiques du cycle de vie d'un logiciel avec certains objectifs additionnels :

- Identification et caractérisation du problème (Analyse de besoins) : à partir de l'analyse du domaine et des applications, identifier les abstractions que le client du framework utilise pour exprimer ses problèmes et fournir une logique pour résoudre les problèmes avec ces abstractions.
- Identifier, pendant la conception du framework, comment le client va utiliser ce framework.

  Quels sont les processus que le framework réalise ? Quels sont les processus que le client réalise ?

  Quelles sont les classes que le client utilise ?

  Quelles sont les méthodes que le client utilise ?

  Comment interagissent les classes et les méthodes du framework avec les classes et les méthodes du client ?

  Le but est d'arriver à un équilibre entre la flexibilité du framework et la simplicité d'utilisation pour le client.

- Implémentation et tests : La documentation du code est spécialement importante pour la réussite d'un framework. Les utilisateurs doivent comprendre comment le framework fonctionne et comment produire une application avec. Dans cette étape, il est nécessaire de produire des descriptions du framework et des exemples d'utilisation bien documentés à l'aide de diagrammes d'architecture.
- Raffinement de la conception : au vu de l'utilisation du framework par les clients, le framework est complété en ajoutant des fonctionnalités. A cette étape, le processus devient itératif.

5.3.4 Avantages et inconvénients des frameworks

Les avantages des frameworks, quand ils sont bien construits, sont soulignés par plusieurs auteurs [TAL 93], [JOH 97], [ROB 97], [FAY 97], [NEM 97] :

- Le principal avantage des frameworks est qu'ils permettent de réutiliser en même temps du code et des éléments de conception.
- Un framework fournit une infrastructure et un guide architectural au développeur. Cela permet d'améliorer la qualité du logiciel car le développeur ne doit pas tout construire et la structure du framework guide le développement.
- Un framework permet de réduire les efforts de codage, de test et de mise au point des applications car la plupart des fonctionnalités sont fournies, ce qui limite la quantité de code à écrire par le développeur.
- La productivité des développeurs augmente car ils peuvent se concentrer uniquement sur les caractéristiques particulières de l'application qu'ils développent.
- Les efforts de maintenance sont réduits car si une erreur est corrigée dans un framework, ou si une nouvelle fonctionnalité y est ajoutée, ces améliorations sont facilement répercutées sur les applications construites avec ce framework. Il est également plus facile de passer d'une application à une autre, ce qui réduit encore l'effort de maintenance.
- Les frameworks permettent de réduire les délais de mise sur le marché des produits des entreprises.

Mais "si développer des applications orientées par objets est difficile, développer des applications orientées par objets réutilisables est plus difficile encore" [GAM 94]. Que dire alors du développement des frameworks qui sont des applications complexes, extensibles et réutilisables ? On peut faire les remarques suivantes :

- L'apprentissage de l'utilisation efficace d'un framework exige du temps et des efforts. Pour devenir un expert de frameworks tel que MacApp ou MFC, par exemple, il est nécessaire d'investir de 6 à 12 mois [FAY 97]. Il est indispensable ici d'avoir une bonne documentation et des processus de formation bien établis.

- Lors de la mise au point d'une application, il est difficile de distinguer entre les erreurs dues au framework et les erreurs propres à l'application.
- La maintenance d'un framework exige des développeurs compétents avec une connaissance profonde du framework.
- Le développement d'un framework est coûteux. Il est nécessaire d'avoir des développeurs experts, de faire des analyses en profondeur des domaines et de développer des versions successives.
- L'utilisation d'un framework oblige à recourir au langage de programmation employé pour construire le framework lui-même.

5.4 Résumé

Nous avons présenté brièvement dans ce chapitre les approches de réutilisation à différents niveaux pour le développement de logiciels.

Les styles d'architecture servent à guider la conception de haut niveau d'un système. L'utilisation d'un style particulier permet d'identifier globalement les types de composants du système et les relations entre ces composants.

Les patrons de conception permettent de réutiliser des techniques validées par l'expérience pour résoudre des problèmes récurrents dans la conception de systèmes orientés objets. Leur utilisation contribue à l'amélioration de la qualité des logiciels.

Les composants constituent à l'heure actuelle la technique de réutilisation la plus répandue dans le milieu industriel. La possibilité d'utiliser des bibliothèques de composants, adaptées aux besoins de développement spécifiques, est une aide immense pour les développeurs.

Finalement, les frameworks utilisent toutes les techniques signalées ci-dessus pour permettre la réutilisation à la fois d'architectures, de conceptions et de code, pour générer des applications d'un domaine particulier. Un framework capture le savoir-faire de la construction des applications qui appartient à un domaine spécifique. La conception et le développement d'un framework sont des tâches longues, complexes et coûteuses.

Le chapitre suivant va montrer comment nous avons exploité concrètement ces divers éléments de réutilisation.

6 VERS UN FRAMEWORK BASE SUR MARS (-S)

Nous voulons maintenant aborder le développement des outils-auteurs eux-mêmes. Nous nous intéressons à des développeurs d'outils-auteurs. Ces développeurs veulent créer des outils adaptés a une famille[5] spécifique de simulations à construire. Notre idée est de leur fournir un framework afin de faciliter et d'accélérer le processus de développement de ces outils.

Après un rappel de nos motivations, nous présentons en détail MARS-S[6], la partie de MARS qui sert de base à notre framework, ainsi que les mécanismes indispensables à son exécution. Nous rappelons ensuite le processus général de développement d'un outil-auteur. Puis, nous discutons l'approche que nous avons utilisée pour réaliser notre framework. Nous donnons d'abord une vision globale du framework, puis son architecture en insistant sur l'utilisation de patrons de conception. Nous montrons, pour finir, l'état de développement de deux outils-auteurs construits en appliquant notre framework.

---

[5] Nous appelons *famille de simulations* un groupe de simulations de même type (par exemple, les simulations MediSchool présentées dans le chapitre 4) ou qui sont des simulations pour un domaine spécifique (par exemple des simulations de Mécanique)

[6] Lire MARS moins S

6.1 Motivations et objectifs

L'étude du processus de développement de simulations pédagogiques a conduit notre équipe à proposer un modèle de conception, MARS [PER 96], et à développer des outils-auteurs, MELISA et OASIS, basés sur ce modèle.

Les expérimentations de ces outils ont conduit aux constats suivants (cf. chapitre 4) :

- Les auteurs ont eu du mal à modéliser avec les formalismes proposés et ils ont eu besoin d'être assistés dans cette tâche par notre équipe.
- Les auteurs préfèrent exprimer leurs modèles avec des formalismes propres à leur domaine ou au type de simulation à développer.
- On a identifié des processus de création pour des types particuliers de simulations.

On peut ajouter que la performance des simulations pédagogiques produites est faible lorsque les systèmes modélisés deviennent complexes. Cela est dû en partie au fait que MELISA et OASIS sont construits comme des surcouches de ToolBook.

Ces constats ont conduit à identifier quelques besoins :

- Réaliser un outil indépendant de la plate-forme ToolBook pour améliorer les performances des simulations produites.
- Permettre qu'une simulation créée puisse être reliée à un contrôle pédagogique indépendant. (Cet aspect sera développé dans la troisième partie.) De ce fait, il suffit de prendre comme base le modèle MARS sans la partie scénario (ce que nous désignerons par MARS-S), et non pas le modèle MARS complet.
- Permettre d'adapter rapidement l'outil aux types de simulations développées et aux besoins d'expression des auteurs. Les adaptations souhaitables repérées dans les expérimentations vont de l'ajout de nouveaux objets de présentation jusqu'au changement des interfaces (auteur et apprenant) et des concepts manipulés. Une telle variété ne peut être satisfaite par un seul outil car nous ne pouvons pas prévoir toutes les adaptations possibles. Il s'agit donc de permettre de développer rapidement un outil adapté.

L'objectif global est d'avoir une sorte d'atelier de développement pour la production d'outils basés sur MARS-S.

En réponse à ces besoins, ma proposition de est schématisée dans la Figure 6-1 :

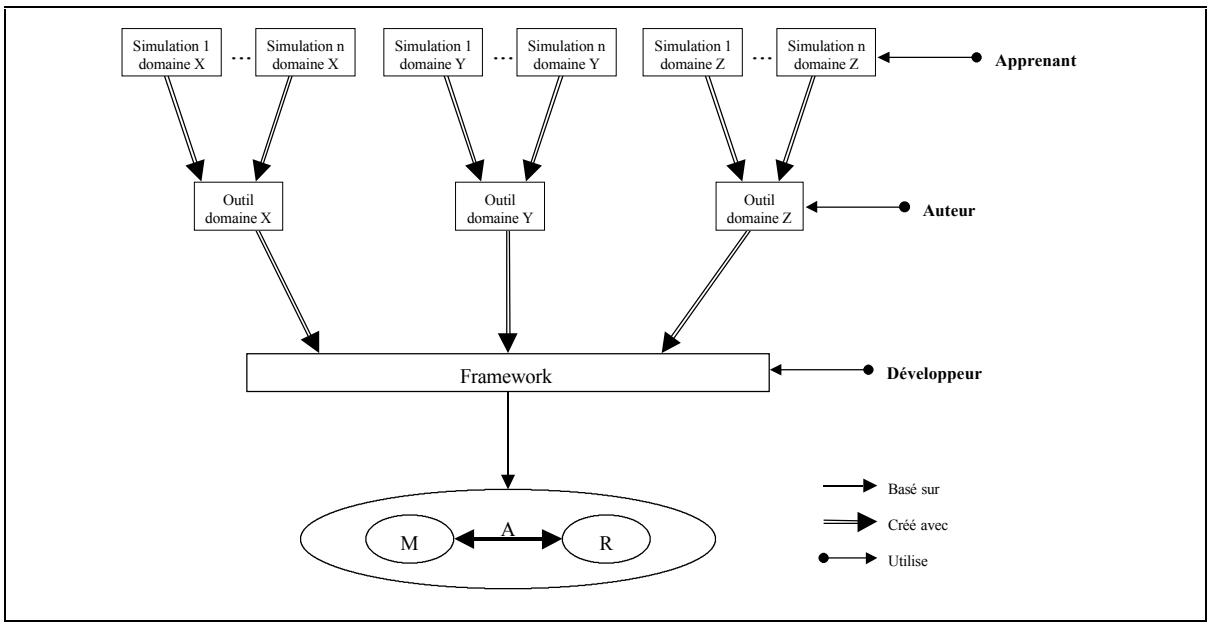


Figure 6-1: Objectif du framework

Comme nous l'avons vu dans le chapitre 5, les frameworks sont une technique de réutilisation qui capture le savoir-faire de conception et de développement pour une famille d'applications. Dans notre cas, cette famille est celle des outils de production de simulations basés sur MARS-S. Si les développeurs ne souhaitent pas d'adaptations, le framework doit permettre de créer "par défaut" un équivalent de l'outil OASIS (sans l'éditeur de scénarios).

6.2 MARS-S le modèle de base

Comme nous l'avons vu dans la partie 1, le modèle MARS complet propose quatre espaces de travail pour le développement d'une simulation pédagogique : l'espace modèle, l'espace représentation, l'espace associations et l'espace scénario. Seuls les trois premiers sont pris en compte dans MARS-S.

Une explication détaillée s'avère nécessaire avant d'aborder le framework. Pour concrétiser, nous prendrons un exemple simple de simulation et nous montrerons comment l'exprimer avec les formalismes de chaque espace. Nous analyserons, pour finir, les aspects liés à l'exécution d'une simulation.

6.2.1 Exemple: Simulateur de mouvement rectiligne uniformément accéléré

En Physique, les concepts de base de la cinématique sont introduits avec le mouvement rectiligne uniformément accéléré. L'élève a parfois des difficultés pour associer la réalité et le modèle physique du mouvement. Pour aborder cet aspect, nous voulons simuler une voiture qui se déplace avec un mouvement rectiligne uniformément accéléré.

Spécification de la simulation

La simulation devra permettre à l'apprenant de : changer l'accélération de la voiture, voir la vitesse de la voiture, initialiser la simulation, démarrer / arrêter la simulation.

La figure suivante est une maquette de l'interface désirée :

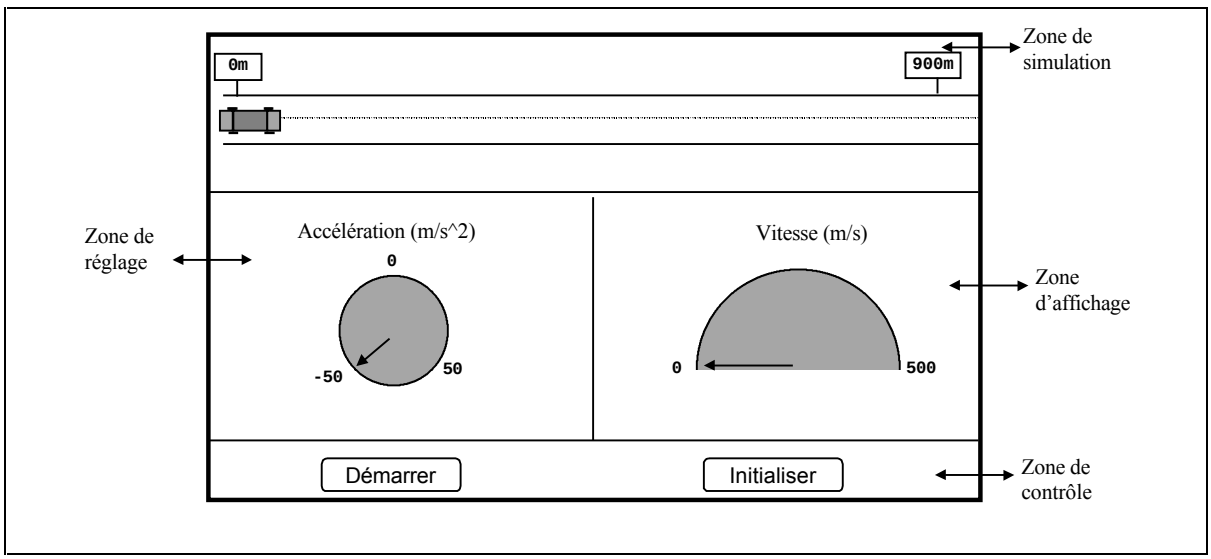


Figure 6-2: Maquette de l'interface de la simulation

Le Tableau 6-1 décrit les effets des actions de l'utilisateur.

Action	**Effet**
Déplacement de l'aiguille du réglage de l'accélération	La valeur de l'accélération change
Clic sur le bouton *Go*	La voiture se déplace avec les paramètres donnés par l'utilisateur. La zone d'affichage montre les modifications de la vitesse. Le nom du bouton *Go* change en *Pause*.
Clic sur le bouton *Pause*	La simulation s'arrête, sans changement des valeurs des variables. Le nom du bouton *Pause* change en *Go*.
Clic sur le bouton *Initialize*	Les valeurs de la position et de la vitesse s'initialisent à 0.

Tableau 6-1: Description des actions de l'utilisateur dans la simulation

Le mouvement de la voiture est modélisé par deux équations :

$\Delta V = a * \Delta t$	ΔV: Variation de la vitesse pendant un intervalle de temps Δt
$\Delta X = (V_0 + a * \Delta t / 2) * \Delta t$	ΔX: Distance parcourue pendant cet intervalle de temps
	a: accélération
	V_0: la vitesse initiale

6.2.2 Le modèle abstrait

Un modèle abstrait est défini par une structure, un comportement et une dynamique :

- La *structure* du modèle comporte un ensemble de propriétés (ou attributs) ayant chacune un type et une valeur initiale.
- Le *comportement* est défini par un ensemble de méthodes applicables au modèle.
- La *dynamique* est définie par un graphe d'états et de transitions. Le graphe comporte un ensemble d'états, une liste d'événements de transition et un ensemble de transitions. Un état a un nom, une méthode à exécuter avant d'entrer dans l'état, une méthode à exécuter pendant l'état et une méthode à exécuter avant de sortir de l'état. Une transition a un nom, un état d'origine, un état de destination, le nom de l'événement de transition qui déclenche la transition, et une condition qui est évaluée avant d'effectuer la transition. Un des états est désigné comme état initial.

Dans notre exemple concret, le modèle correspond au modèle physique qui décrit le mouvement de la voiture. Il comporte aussi des éléments de pilotage de la simulation (démarrer/arrêter et initialiser).

Les propriétés du modèle

Les propriétés du modèle sont ici les variables du système d'équations du modèle physique. L'accélération est l'unique variable d'entrée, la vitesse et la position sont des variables de sortie calculées par le modèle. Le temps est une variable indépendante.

Pour chaque variable, nous définissons une propriété. Le Tableau 6-2 décrit les propriétés et leurs caractéristiques.

Nom	Description	Type	Valeur initiale
x	position de la voiture par rapport au point de départ.	réel	0
v	vitesse instantanée de la voiture.	réel	0
a	accélération de la voiture	réel	0
t	temps écoulé depuis le début de la simulation	réel	0
dt	valeur de l'intervalle de temps utilisé dans les calculs	réel	0.2

Tableau 6-2: Propriétés du modèle de la simulation

Les méthodes du modèle

Pour chaque variable d'entrée, il faut une méthode pour modifier sa valeur, car on doit pouvoir la modifier depuis la représentation. Ainsi pour changer la valeur de *a*, nous définissons la méthode *set_a*.

Pour réinitialiser la simulation, nous définissons la méthode *initialize*, qui met les propriétés *x*, *v,a* et *t* à 0.

Le graphe d'états du modèle

La Figure 6-3 montre le graphe d'états et de transitions de la simulation.

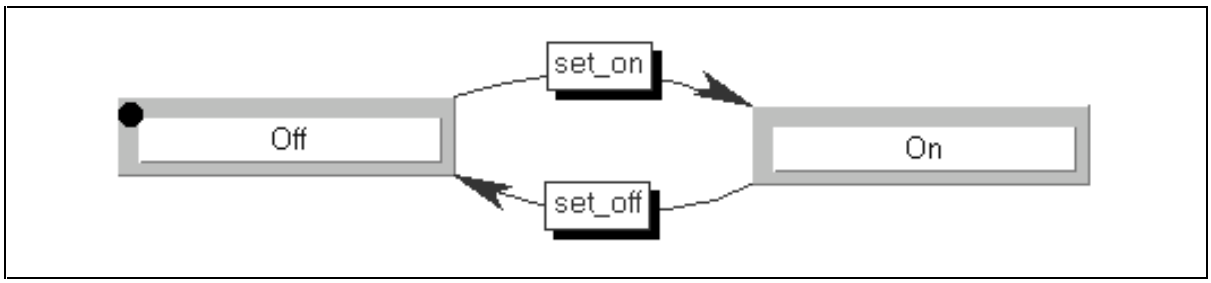


Figure 6-3: Graphe d'états et de transitions de la simulation

Les états *Off* et *On* permettent d'exprimer le fait que la simulation est en marche ou arrêtée. Le point qui marque l'état *Off* indique l'état initial. Dans l'état *On,* le modèle doit calculer constamment la position de la voiture. Pour cela, nous définissons une méthode à exécuter (ou activité) pendant que la simulation est dans cet état; cette méthode fait les calculs correspondant aux équations du modèle physique :

```
x = x + (v * dt) + 1/2 * (a * dt * dt)
v = v + a * dt
t = t + dt
```

6.2.3 La présentation

La présentation d'une simulation est définie par un ensemble d'objets de représentation. Chaque objet a lui-même un modèle (contenant les mêmes types d'éléments que le modèle défini ci-dessus), plus une interface (ou représentation) permettant à l'utilisateur de le manipuler ou de visualiser son état interne.

Cet espace offre le concept de bibliothèques d'objets réutilisables. Des objets élémentaires d'entrée et de sortie y sont prédéfinis et disponibles. L'auteur peut les importer et les paramétrer pour construire la présentation de la simulation.

Dans notre exemple, nous avons cinq objets :

- Un objet permettant de régler l'accélération.
- Un objet permettant d'afficher la vitesse.
- Le bouton *Initialize*.
- Le bouton *Go/Pause*.
- L'objet permettant d'afficher la position.

Pour les quatre premiers, nous pouvons utiliser des objets de la bibliothèque. Pour la position, il a été nécessaire de définir un nouvel objet symbolisant une voiture, ce type d'objet ne figurant pas dans la bibliothèque.

Le Tableau 6-3 donne pour chaque objet utilisé, son nom, sa visualisation, son type et une courte description de son fonctionnement.

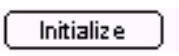
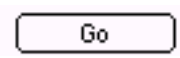
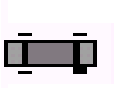
Nom	Visualisation	Type / Fonctionnement
acceleration		Objet d'entrée : La flèche peut être déplacée par l'utilisateur en la faisant glisser à l'aide de la souris. La valeur choisie est sauvegardée dans la propriété CurrentVal de cet objet.
speed		Objet de sortie: La valeur à afficher peut être changée avec la méthode SetValue (dont le paramètre indique la valeur désirée).
initialize		Objet d'entrée: Quand l'utilisateur clique sur le bouton, la propriété Trigger de cet objet met sa valeur à TRUE et après un moment se remet à False.
GoPause		Objet d'entrée: Quand l'utilisateur clique sur le bouton, la propriété booléenne ActualState change de valeur. Quand la valeur de ActualState est TRUE, le bouton a l'étiquette Pause; si la valeur est FALSE, l'étiquette est Go.
auto		Objet de sortie: La position à l'écran de la voiture peut être changée avec la méthode SetPos (dont le paramètre indique la valeur désirée).

Tableau 6-3: Objets de représentation de la simulation

6.2.4 Les associations

Une association en MARS sert à lier un objet de représentation avec le modèle ou vice-versa. Les associations du modèle vers les objets de représentation permettent de modifier les objets de représentation quand les propriétés ou l'état du modèle changent. Les associations des objets de représentation vers le modèle permettent de modifier les valeurs des propriétés du modèle ou l'état du modèle quand les propriétés des objets de représentation changent.

Les associations sont basées sur des règles événement-condition-action, et lient un objet d'origine (le modèle ou un objet de représentation) avec un objet de destination (respectivement un objet de représentation ou le modèle). L'événement correspond à la modification d'une propriété de l'origine ou à l'occurrence d'un événement interne. La condition correspond à un prédicat sur les propriétés de l'origine. L'action consiste à exécuter une méthode de la destination, ou à lui envoyer un événement de transition. La spécification d'une association consiste à déterminer l'origine, la destination, la propriété de l'origine à observer, la condition à évaluer, et la méthode à exécuter ou l'événement de transition à envoyer.

Pour notre exemple, nous devons spécifier deux associations allant du modèle vers la représentation (cf. Tableau 6-4) :

- Une association pour spécifier que lorsque la propriété vitesse *v* du modèle change, l'objet de représentation *speed* doit modifier la valeur qu'il affiche.
- Une association pour spécifier que lorsque la propriété position *x* du modèle change, l'objet de représentation *auto* doit modifier sa position.

Condition sur une propriété ou sur l'état du modèle		**Objet**	**Action sur l'objet**
Si v change	⇒	*speed*	exécute SetValue (v)
Si x change	⇒	*auto*	exécute SetPos (x)

Tableau 6-4: Associations modèle – objet

Nous devons aussi spécifier quatre associations (cf. Tableau 6-5) allant des objets de représentation vers le modèle:

- Une association pour spécifier que lorsque l'utilisateur modifie la valeur de l'objet de représentation *acceleration*, la valeur de la propriété *a* du modèle doit être modifiée.
- Une association pour spécifier que lorsque l'utilisateur appuie sur le bouton *Initialize*, le modèle doit initialiser les propriétés *v, t et x*.
- Une association pour spécifier que quand l'utilisateur appuie sur le bouton *GoPause* si la valeur de la propriété *ActualState* de l'objet est *true*, alors il faut envoyer au modèle l'événement de transition *Set_On*.
- Une association pour spécifier que quand l'utilisateur appuie sur le bouton *GoPause* si la valeur de la propriété *ActualState* de l'objet est *false*, alors il faut envoyer au modèle l'événement de transition *Set_Off*.

Objet	**Condition sur une propriété ou sur l'état de l'objet**		**Action sur le modèle**
acceleration	Si CurrentVal change	⇒	Exécute set_a (CurrentVal)
GoPause	Si ActualState = true	⇒	Evénement de transition Set_On
GoPause	Si ActualState = false	⇒	Evénement de transition Set_Off
Initialize	Si Trigger = true	⇒	Exécute initialize ()

Tableau 6-5: Associations Objet –Modèle

6.2.5 Exécution de la simulation

Pour faire fonctionner une simulation ainsi exprimée, il est nécessaire de disposer d'un moteur capable d'exécuter une simulation à partir de sa description. Ce moteur doit prendre en charge la dynamique du modèle et des objets, les associations, ainsi que les méthodes du modèle et des objets.

Reprenons notre exemple pour mieux comprendre le fonctionnement d'un tel moteur. Supposons que l'apprenant change la valeur de l'objet *acceleration*. Le moteur doit se rendre compte qu'il y a un changement et vérifier alors s'il existe des associations objet–modèle ayant comme origine l'objet *acceleration*. Comme il en existe une, le moteur doit exécuter l'association. Le moteur vérifie d'abord si la condition de l'association est vraie (ici, la condition est vérifiée: il suffit d'un changement de valeur); puis il exécute alors l'action prévue: dans ce cas, il exécute la méthode *set_a* avec comme paramètre la valeur de la propriété *CurrentVal* de l'objet *acceleration*.

Supposons maintenant que l'apprenant appuie sur le bouton *Go*. Le moteur doit se rendre compte que la valeur *ActualState* de l'objet *GoPause* a changé à *true* et doit examiner les associations qui partent de l'objet *GoPause*. Il existe deux associations; l'une n'est pas exécutée car sa condition n'est pas vraie (*ActualState* = *false*), mais l'autre doit être exécutée: envoi au modèle de l'événement de transition *Set_On*. Le moteur doit alors exécuter la méthode de sortie de l'état *Off*, puis la méthode d'entrée de l'état *On* et changer l'état du modèle à l'état *On*. Comme il existe une méthode à exécuter pendant que le modèle est dans l'état *On*, le moteur doit exécuter cette méthode à intervalles réguliers. Notons que cette méthode provoque la modification des valeur des propriétés *x* et *v* de la simulation qui vont à leur tour déclencher les associations modèle–objet dépendant de ces propriétés.

On voit bien que le travail de ce moteur est tout à fait essentiel à la bonne exécution d'une simulation.

Après cette explication, certes un peu longue mais indispensable, des détails des formalismes de MARS-S et du moteur d'exécution correspondant, nous pouvons revenir vers notre objectif, le framework pour développeur d'outils-auteurs (revoir ici la Figure 6-1).

6.3 Processus de construction des outils-auteurs

Étant donnée une famille de simulations, le framework doit faciliter la démarche de construction d'un outil pour cette famille, démarche que nous proposons ci-dessous :

- Étape 1 : Étudier la réalisation des simulations de la famille en s'appuyant sur le modèle MARS-S. Cette étape implique l'équipe de développement et un expert connaissant bien les concepts et les formalismes de cette famille de simulations. Le but est d'identifier un mécanisme de traduction des formalismes propres à la famille vers les formalismes MARS-S. Pour cela, il convient de réaliser, concrètement, avec OASIS un ou plusieurs exemples afin de dégager une méthode pour construire ce type de simulations selon les espaces MARS-S et de définir si nécessaire une collection d'objets de présentation complémentaires (objets spécifiques à la famille et objets génériques indépendants de la famille).
- Étape 2 : Spécifier les interactions les mieux adaptées à la famille de simulations. Il faut identifier les interfaces utilisateurs qui permettront aux

auteurs de décrire les informations de la simulation, ainsi que les aspects communs des interfaces qui seront proposées aux apprenants. Cette étape implique l'équipe de développement et un expert du domaine.

- Étape 3 : Développer les interfaces d'interaction avec l'auteur, les objets et le mécanisme de traduction. Cette étape concerne l'équipe de développement.
- Étape 4 : Test des fonctionnalités de l'outil, et test de fonctionnement des simulations produites avec l'outil. Cette étape implique l'équipe de développement et un expert du domaine.

Cette démarche implique certaines contraintes sur le framework :

- Le framework doit fournir à l'équipe de développement un support pour la manipulation et l'exécution du modèle MARS-S, qui reste le modèle de base pour tous les outils.
- Le framework doit faciliter la création de nouveaux objets de présentation et leur intégration à la bibliothèque d'objets.
- Le framework doit guider l'équipe de développement dans ses tâches d'analyse, de conception et de développement des aspects qui changent entre les applications, c'est-à-dire des aspects propres à la famille de simulations : les interfaces, la traduction en MARS-S et les objets de présentation.

6.4 Processus de développement du framework

Dans une première analyse de besoins pour notre framework, nous différencions deux parties : l'une implémente le moteur correspondant au modèle MARS-S, et l'autre implémente les interactions avec l'auteur.

Nous connaissions bien le domaine d'application du sous-framework portant sur MARS-S. Nous avons bénéficié en effet de l'expérience retirée de la construction de deux outils, de la réalisation de plusieurs simulations avec ces outils, et de réalisation d'autres simulations directement avec des langages de programmation (mais en appliquant le modèle). Nous pouvons donc proposer de construire pour cette partie un sous-framework, en adoptant une approche constructive.

Pour l'autre partie (interactions avec l'auteur), nous avions l'expérience des simulations avec les outils généralistes MELISA et OASIS, et avec l'outil spécialisé GeneSimu. Ceci a permis d'identifier le besoin d'adaptation, de lister au moins trois outils à produire et d'entrevoir d'autres outils envisageables. Nous pouvons donc proposer de construire un sous-framework pour cette partie, en adoptant cette fois une approche évolutive.

Nous présentons dans la Figure 6-4, le processus de construction du framework qui combine les approches constructive et évolutive dont les étapes sons commentées ci-dessous.

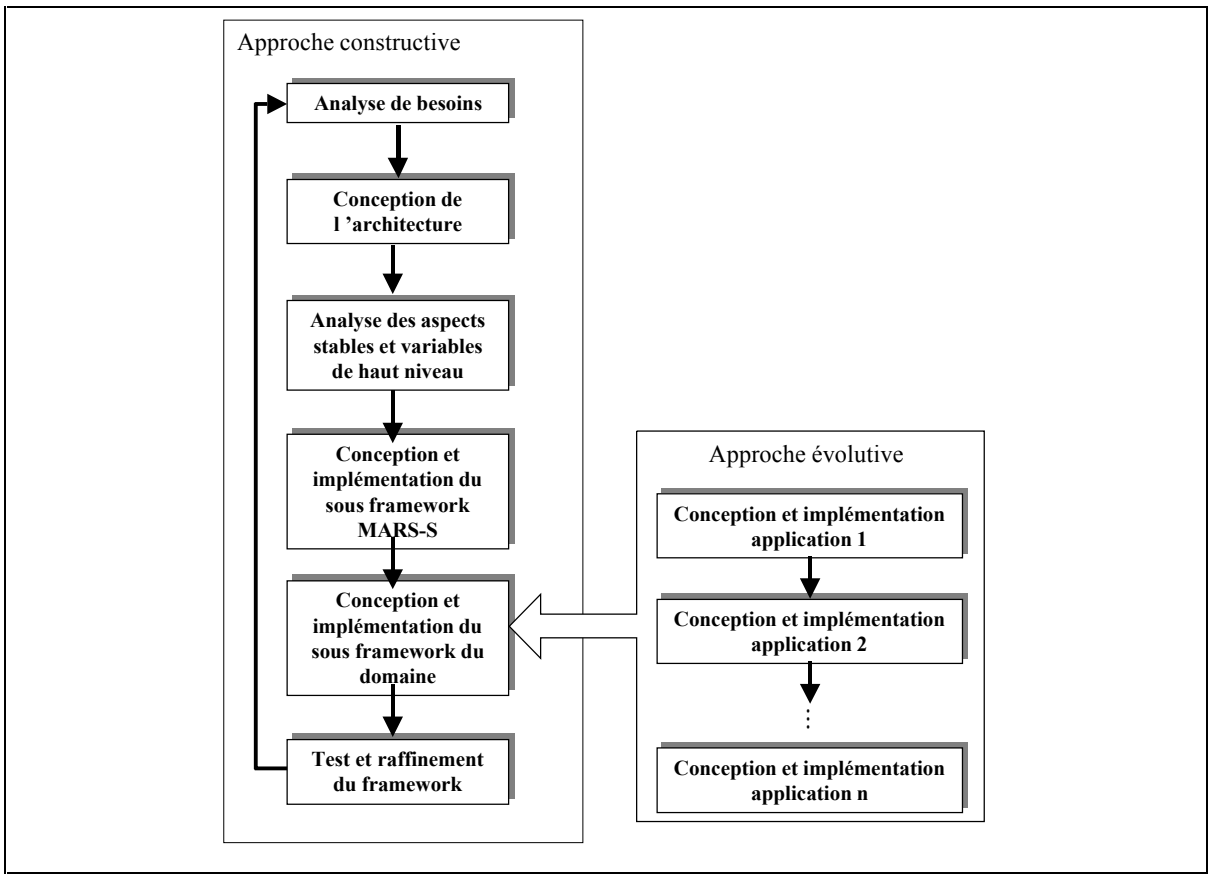


Figure 6-4: Processus de développement du framework

- Analyse de besoins : nous analysons ici le processus de construction des outils basés sur MARS-S, nous identifions les abstractions de ce domaine et nous soulignons les aspects susceptibles de variations.
- Définition de l'architecture du framework : à l'aide de styles existants, nous concevons l'architecture du système, en identifiant ses composants et leurs relations. Nous établissons en particulier les relations entre les deux sous-frameworks.

- Analyse des aspects stables et variables de haut niveau : nous identifions les aspects stables (qui ne doivent pas être modifiés) et variables (à modifier ou à compléter par le développeur) de chaque composant de l'architecture.
- Conception du sous-framework MARS-S : à l'aide de patrons de conception, nous définissons les classes et leurs relations pour les aspects stables de l'architecture. Nous établissons ici comment le développeur va utiliser la partie stable du framework.
- Implémentation et test du sous-framework MARS-S.
- Conception et implémentation du sous-framework du domaine : pour ce sous-framework, nous adoptons une approche évolutive. Nous construisons une suite d'outils en utilisant les composants stables déjà construits et nous faisons évoluer petit à petit les composants variables.
- Test et raffinement du framework : réalisation des nouveaux outils avec le framework et évolution en fonction des nouveaux besoins.

La construction d'un framework n'est pas identique à la construction d'une application quelconque. Avant d'arriver à une version stable, il faut construire plusieurs applications avec le framework. En ce sens, le développement d'un framework est un processus long et complexe. J'ai déjà franchi les quatre premières étapes évoquées ci-dessus et j'ai développé deux applications de la cinquième étape. Dans ce qui suit, je présente d'abord une vision globale du framework, puis une vision détaillée et finalement les deux outils produits.

6.5 Vision globale du framework

Le framework doit apporter une aide dans le processus de développement, ce qui implique essentiellement trois aspects :

- Les outils eux-mêmes. Nous distinguons deux tâches, l'édition et l'exécution, de la simulation. L'auteur édite une simulation à l'aide d'interfaces spécialisées pour la famille de simulations. Au moment de l'exécution, l'outil traduit les informations capturées par l'éditeur en une représentation de données en MARS-S, puis exécute la simulation MARS-S. Les éditeurs et le processus de

traduction sont des **aspects variables,** tandis que la représentation MARS-S et les fonctionnalités d'exécution sont des **aspects stables**.

- Les formalismes du modèle MARS-S utilisés pour exprimer et exécuter une simulation. La représentation de données et l'exécution d'une simulation MARS-S sont des aspects stables du framework.
- Les formalismes propres aux familles de simulations. Ces abstractions font partie du processus de développement de l'outil, car elles sont nécessaires pour définir la traduction du formalisme de la famille au formalisme MARS-S. Ces abstractions sont **variables** pour des simulations de différentes familles, mais sont **stables** pour des simulations d'une même famille. La traduction des formalismes de la famille au formalisme MARS-S est **variable** pour chaque famille.

L'analyse de ces aspects a permis d'établir de façon globale les aspects stables et les aspects variables à considérer dans le framework :

Aspects stables	Aspects variables
Représentation d'une simulation MARS-S Exécution d'une simulation MARS-S Gestion des simulations MARS-S Objets de présentation génériques	Interfaces adaptées aux familles de simulations Traduction entre formalismes Formalismes des familles de simulations Objets de présentation des familles de simulations

Tableau 6-6: Aspects stables et variables du processus

Nous appelons « FREAK » le sous-framework qui s'occupe des aspects stables et « Outil d'édition » celui qui s'occupe des aspects variables (cf. Figure 6-5).

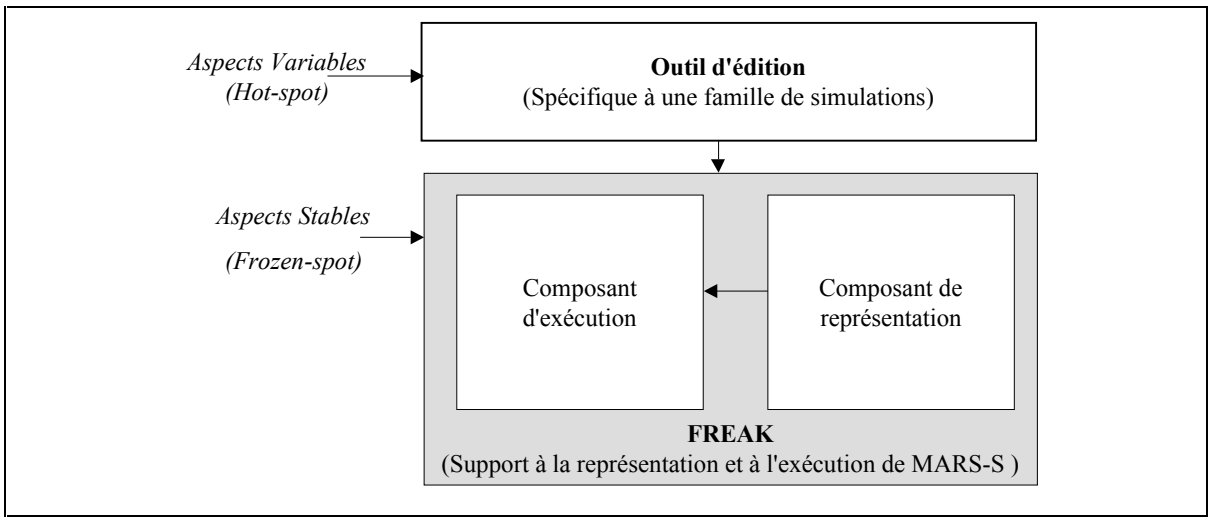


Figure 6-5: Structure globale du framework

Le sous-framework FREAK

FREAK offre des services pour la manipulation abstraite du modèle MARS-S et pour l'exécution des simulations utilisant ce modèle, avec un composant spécialisé pour chaque aspect.

Le composant de représentation fournit des services pour :

- La création des simulations MARS-S.
- La gestion de modèles : ajout/édition/suppression de propriétés, de méthodes, d'états, d'événements et de transitions.
- La gestion d'associations : ajout/édition/ suppression.

Le composant d'exécution est un moteur pour l'exécution des simulations MARS-S.

Nous reviendrons plus loin sur les détails de conception et d'implémentation d'une première version de FREAK.

Le sous-framework outil d'édition

Pour réaliser ce sous-framework, nous avons choisi une approche évolutive, car c'est à partir d'expériences de développement d'outils que nous pourrons arriver à finaliser ce framework.

Selon cette approche, le framework est construit itérativement en développant une série d'applications de même famille. Une première phase consiste à arriver à un framework de type boîte blanche, après la construction d'au moins trois applications. Une deuxième phase consiste à réaliser de nouvelles applications en utilisant le framework de la première phase et à le faire évoluer en un framework de type boîte noire. Une troisième phase consiste à construire des outils visuels pour la manipulation du framework.

Nous sommes actuellement dans la première phase. Nous développons en parallèle deux outils pour la production de simulations : SIMARS, un outil qui correspond à OASIS (le cas par défaut du framework) et FENIX, un outil pour des simulations basées sur des modèles mathématiques simples. Ces deux cas nous serviront pour commencer l'analyse

des aspects stables du sous-framework. L'état actuel du développement de SIMARS et de FENIX est présenté plus loin.

6.6 Vision détaillée du framework

6.6.1 Architecture du framework

La Figure 6-6 montre l'architecture détaillée du framework. L'outil d'édition utilise FREAK pour :

- Construire un fichier *.mar* qui contient la représentation MARS-S de la simulation.
- Générer et compiler la simulation en Java à partir du fichier *.mar*.
- Instancier et exécuter la simulation Java
- Utiliser des objets de présentation génériques.

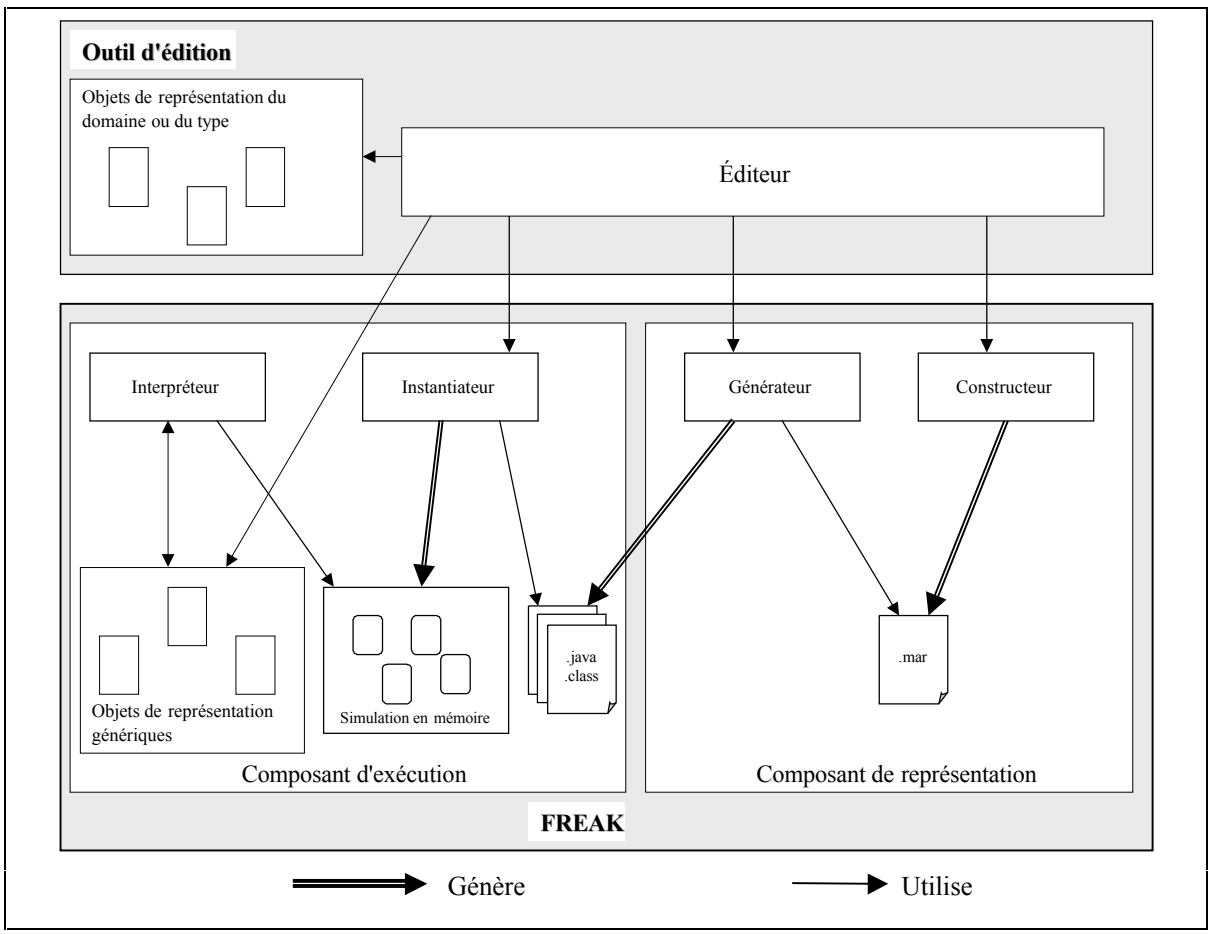


Figure 6-6: Architecture générale du framework

Les points de communication entre les sous-frameworks sont la représentation de la simulation (le fichier *.mar*), les objets de présentation génériques et la simulation elle-même (les fichiers *.class*).

Le sous-framework outil d'édition

L'architecture des outils d'édition est de type MVC [GOL 84], avec un traducteur du modèle de la famille de simulations vers MARS-S (cf. Figure 6-7). Si l'application n'a pas de famille de simulation spécifique, FREAK devient la partie modèle de l'architecture MVC.

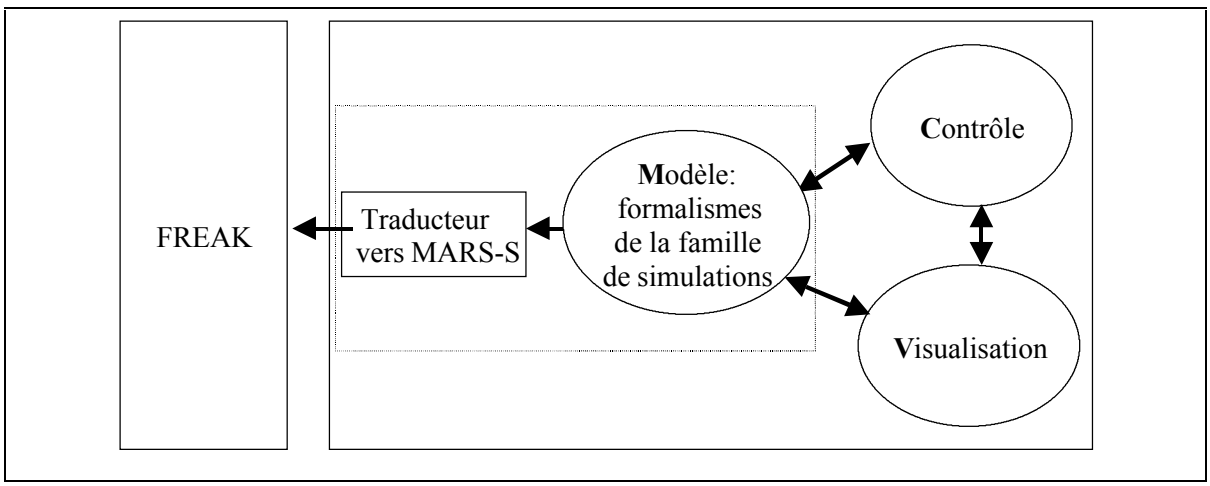


Figure 6-7
Architecture d'une application du sous-framework du domaine

Le sous-framework FREAK

Une simulation MARS-S est une application Java (dont le schéma de classes est présenté dans l'annexe 3). Le sous-framework FREAK est responsable de la création et de l'exécution de la simulation. Son fonctionnement dépend de la représentation des abstractions du modèle MARS-S.

Comme indiqué auparavant, FREAK comporte un composant de représentation et un composant d'exécution. Examinons-les en détail.

Le composant de représentation a lui-même deux composants:

- Le **constructeur** fournit des services pour gérer la simulation au niveau abstrait: créer la simulation, ajouter une propriété, ajouter une association, etc.

Cette description est sauvegardée dans un fichier `.mar` avec un format spécifique.

- Le **générateur** fournit des services pour générer et pour compiler les classes java qui constituent la simulation.

Le composant d'exécution a également deux composants:

- L'**instantiateur** a la responsabilité d'instancier en mémoire les objets qui représentent la simulation, à partir des classes Java. Il offre aussi des services liés à l'exécution.
- L'**interpréteur** est responsable de l'exécution d'une simulation déjà instanciée. Il capture les interactions avec l'utilisateur et exécute les associations et la dynamique des modèles et des objets.

Nous avons appliqué le style d'architecture *Interpréteur* (cf. annexe 2). Le programme à interpréter étant ici une simulation MARS-S, nous avons établi les correspondances suivantes :

- L'état du programme est déterminé par les valeurs des propriétés de la simulation.
- L'état du contrôle de l'interpréteur correspond à l'état de la dynamique du modèle.
- Le programme interprété correspond à la dynamique même du modèle.
- Le moteur d'interprétation utilise la machine virtuelle de JAVA pour exécuter les méthodes de la simulation MARS-S. Ces méthodes sont activées à travers la dynamique du modèle ou par les associations (après la validation des conditions, qui nécessite aussi la machine Java pour son interprétation).
- Les entrées sont produites par l'utilisateur à travers les objets de représentation qui déclenchent les associations. Les sorties sont produites par les associations activées par le changement du modèle.

Ces correspondances sont à l'origine de l'interpréteur montré dans la Figure 6-8.

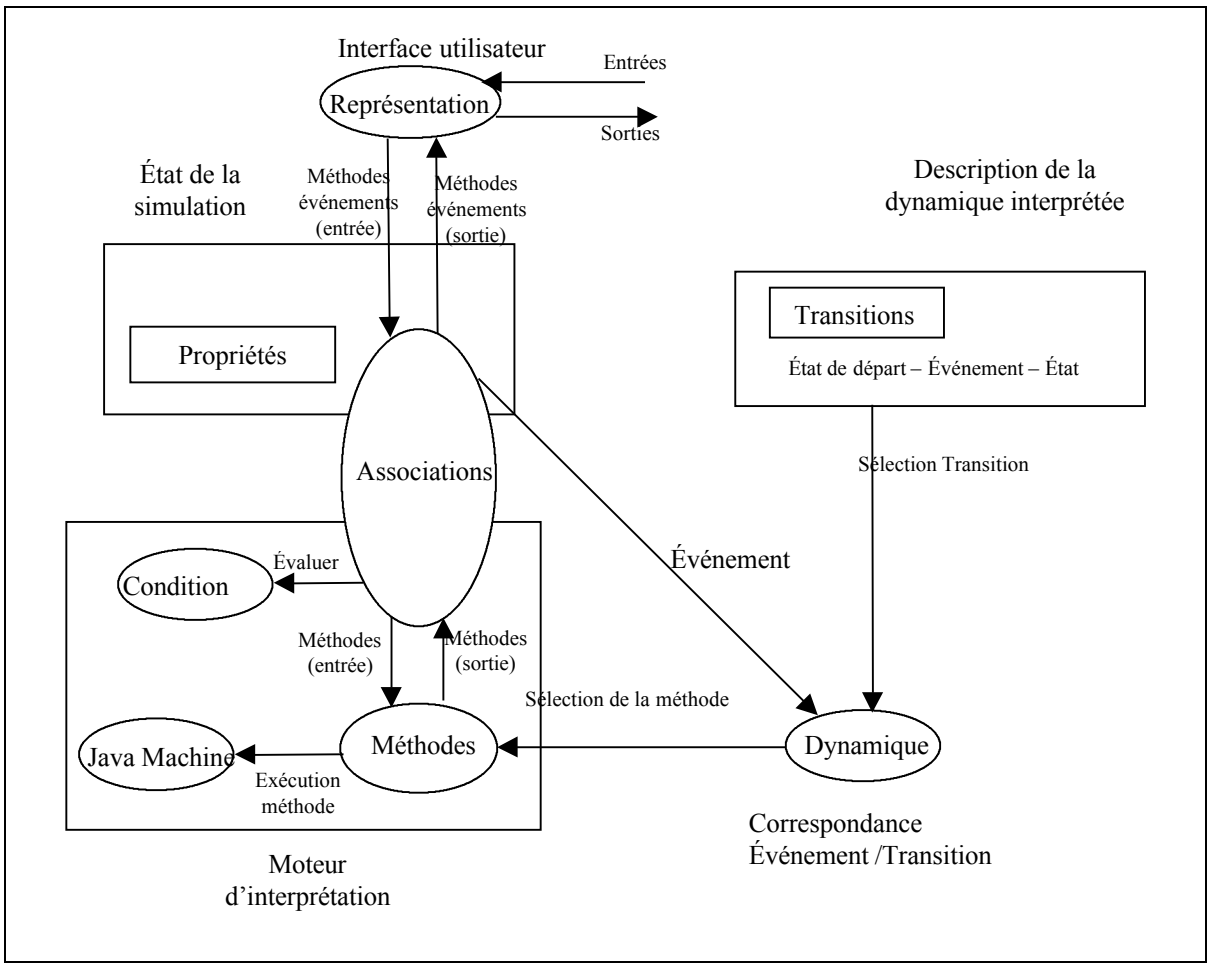


Figure 6-8
Architecture de l'interpréteur de simulations MARS-S

6.6.2 Conception et implémentation de FREAK

L'outil d'édition ne doit pas connaître la structure interne de FREAK. Pour cette raison, FREAK a été implémenté comme un framework de type boîte noire. La conception détaillée des composants de l'architecture a comme objectif d'identifier les classes et leurs relations afin de définir la logique interne des composants. Dans le développement orienté objets, la plupart des décisions sont prises dans cette phase et c'est ici que les patrons de conception prennent toute leur importance.

Nous montrons dans la suite sur quelques exemples comment les patrons de conception ont guidé la prise de décisions de conception pour obtenir une architecture et une conception

réutilisables. Nous indiquons également comment le développeur utilise FREAK. La conception détaillée complète est décrite dans les rapports [ARI 98] et [CJJ 98][7].

6.6.2.1 Application de patrons

Nous avons utilisé des patrons pour la conception des simulations produites par FREAK et pour la conception du framework. Nous résumons ici les principales décisions de conception qui ont bénéficié de l'apport des patrons. L'application détaillée des patrons est présentée dans l'annexe 3.

Dans la conception des simulations produites, nous avons retrouvé des cas classiques: par exemple, le patron *Mediateur* [GAM 95, p.273-291] pour la gestion des éléments de l'interface de la simulation (les objets de représentation, le menu, la barre d'outils et la fenêtre, et le patron *Commande* [GAM 95, p.233-242] pour la gestion de commandes de l'utilisateur à travers le menu, le clavier ou la barre d'outils.

D'autres utilisations, moins évidentes, nous ont servi pour prendre des décisions importantes. C'était le cas pour décider comment gérer les associations de MARS-S qui jouent un rôle essentiel. Elles sont la base de l'exécution de la simulation (cf. description de l'interpréteur). Deux aspects sont fondamentaux pour leur bon fonctionnement: comment notifier les changements du modèle et des objets de présentation afin d'activer les associations et comment déclencher toutes les associations pertinentes lors de l'arrivée d'une notification. Bien que les utilisations les plus connues du patron *Observateur* [GAM 95, p.293-303] concernent les outils d'interface, nous l'avons utilisé pour implémenter la gestion des associations. Les associations se comportent comme des observateurs du modèle et les objets de représentation, sont eux les sujets observables.

Nous avons aussi utilisé des patrons pour faciliter certaines évolutions du framework que nous avions envisagées. Par exemple, dans la première version, nous avons décidé de n'avoir, dans le modèle d'une simulation, que des propriétés élémentaires, en envisageant de traiter plus tard des vecteurs et peut-être d'autres types d'objets. Cet aspect a été important pour le traitement des conditions des associations car elles dépendent du type de

[7] La conception et développement du framework a était fait par un équipe dirigé par moi-même. Cette équipe a été composée par des étudiants de l'Université des Andes en Colombie.

la propriété sur laquelle la condition est appliquée. Pour pouvoir faire évoluer le framework pour traiter d'autres types, nous avons conçu les conditions des associations avec le patron *Stratégie* [GAM 95, p.315-323]. L'association évalue la condition à travers une interface commune à tous les types de propriétés.

6.6.2.2 Utilisation des composants stables par le développeur

Le développeur utilise FREAK par composition de classes. FREAK fournit deux classes pour construire et exécuter des simulations MARS-S :

- La classe *SimulationDescription* avec laquelle le développeur peut créer et gérer la description d'une simulation MARS-S et générer le code Java de la simulation.
- La classe *Simulation* avec laquelle le développeur peut instancier une simulation pour l'exécuter.

Dans l'outil d'édition, le développeur peut créer des classes pour gérer la construction et l'exécution de la simulation en utilisant les classes du framework. La Figure 6-9 montre un exemple.

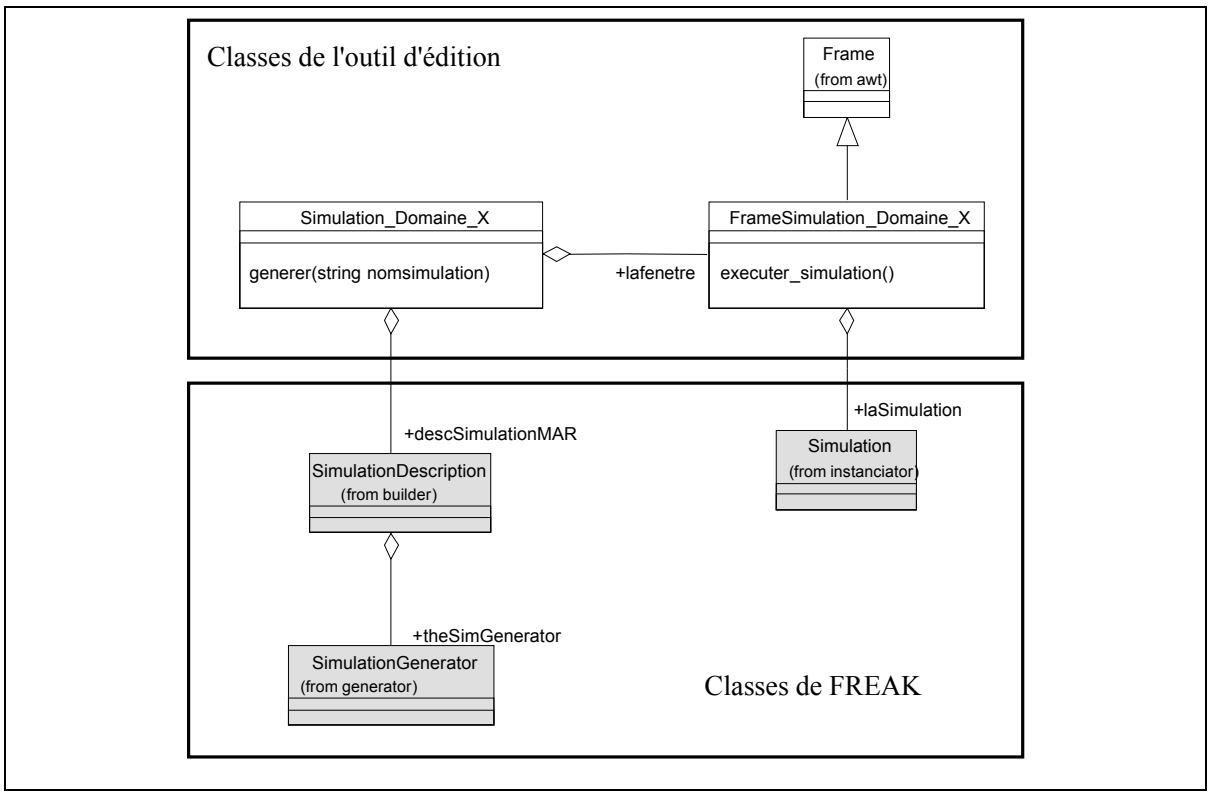


Figure 6-9: Exemple d'utilisation de la partie stable

Dans cet exemple, l'outil d'édition utilise une classe *Simulation_Domaine_X* pour gérer la description et la génération de la simulation. Elle contient une référence à un objet de type *SimulationDescription* et une autre à la fenêtre de la simulation, de type *FrameSimulation_Domaine_X*. Cette dernière est responsable de l'exécution de la simulation. Pour cela, elle contient une référence à un objet de la classe *Simulation*. La classe *Simulation_Domaine_X* délègue à la classe *SimulationDescription* les responsabilités de gestion et de génération de la simulation (cf. Figure 6-10).

```
public generer(string nomsimulation){

  // Création de la description de la simulation
  descSimulationMar = new SimulationDescription();

      ...

       // Addition de propriétés
      p_x = new PropertyDescription("x","Position","double","0.0");
      descSimulationMar.getModel().addProperty(p_x);

      ...
      // Création du fichier .mar
      DescSimulationMar.guardar(nomsimulation);


      // Génération des classes java et compilation
      // le fichier de la simulation est
      // Simulation_<nomsimulation>
      descSimulationMar.generate(nomsimulation);
 …
}
```

Figure 6-10: Exemple du code FREAK : création et génération d'une simulation

Pour sa part, la classe *FrameSimulation_Domaine_X* délègue la responsabilité d'exécution de la simulation à la classe *Simulation* (cf. Figure 6-11).

```
public executer_simulation(){
     lasimulation.execute(
}
```

Figure 6-11: Exemple du code FREAK : exécution d'une simulation

En ce qui concerne les objets de présentation, l'outil d'édition utilise les objets génériques comme composants en se basant sur l'architecture Beans de Java.

6.7 Outils produits

Nous donnons ici deux exemples d'outils-auteurs produits avec le framework.

6.7.1 SIMARS : un outil d'édition générique sur MARS-S

Le premier, SIMARS, est un outil-auteur généraliste qui n'apporte pas de fonctionnalités originales; il représente en effet un équivalent de ce qu'offrent en OASIS les espaces de travail Modèle, Représentation et Association (il ne comporte aucun élément de scénario et ne permet donc que de produire des simulations libres). Il a comme principal intérêt de montrer ce qu'on peut obtenir avec un effort minimal, puisqu'il repose sur les mécanismes par défaut du framework.

SIMARS est en phase d'implémentation; nous avons déjà construit un premier prototype avec des interfaces simples pour tester FREAK. La Figure 6-12 montre l'interface de prototype:

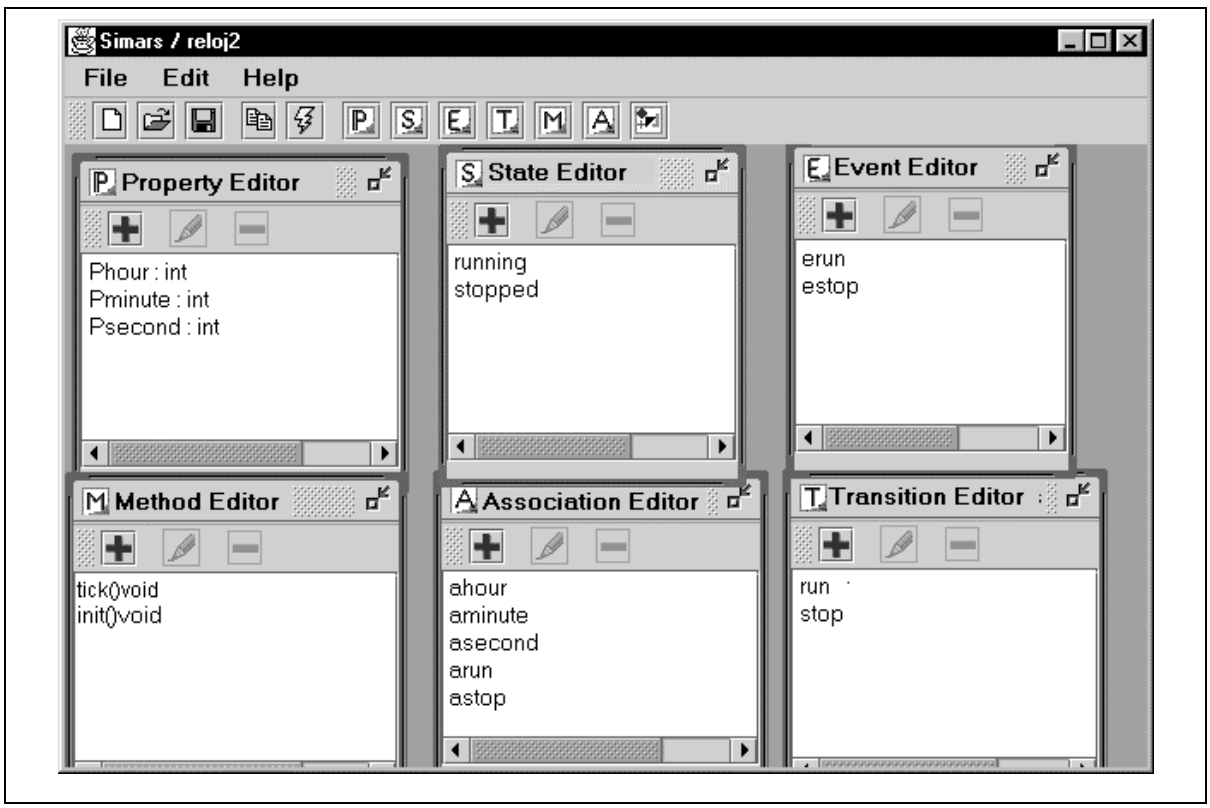


Figure 6-12: Interface du prototype de SIMARS

6.7.2 FENIX : Outil pour des simulations de modèles mathématiques

Nous avons choisi la simulation de modèles mathématiques simples comme premier outil à développer avec FREAK. Cet outil a été inspiré par des simulations de ce type réalisées avec OASIS et par le logiciel Modellus de l'entreprise *Knowledge Revolution* [MOD 99].

Les abstractions du domaine

Un modèle est exprimé avec une liste d'équations de la forme :

```
<variable> = <expression>
```

avec la contrainte qu'une variable ne peut pas être définie par plus d'une équation. Les expressions utilisent les opérateurs et les fonctions listées dans le Tableau 6-7.

Les expressions peuvent inclure des variables indépendantes, ainsi que la variable indépendante **t** qui représente le temps. Pour chaque variable indépendante, il faut fixer sa valeur. Pour le temps, il faut fixer une valeur minimum, une valeur maximum et un delta t. La simulation du modèle est effectuée entre le temps minimum et le temps maximum.

		Sémantique
Opérateurs	**+,-,*,/**	Opérations de base
Constantes	**pi,e**	
Fonctions	**SQRT(x)**	Racine carrée
	EXP(x)	**e^ x**
	LN(x)	Logarithme naturel
	SIN(x),COS(x),TAN(x)	Fonctions trigonométriques
	ASIN(x),ACOS(x),ATAN(x)	Fonctions trigonométriques inverses
	ABS(x)	valeur absolue
Substitution	**SUBST(a,x,e)**	Substitue dans l'expression **a** la sous-expression **x** par la sous-expression **e**
Dérivation	**DIFF(f,x)**	Dérivée de **f** par rapport à **x**
	DIFF(f,x,n)	Dérivée d'ordre **n** de **f** par rapport à **x**
Intégration	**INT(f,x)**	Intégrale indéfinie de **f** par rapport à **x**
	INT(f,x,i1,i2)	Intégrale définie de **f** par rapport à **x** sur l'intervalle **i1,i2**
Approximation de Taylor	**TAYLOR(f,x,x0,n)**	Approximation de Taylor d'ordre **n** de **f** par rapport à **x** en **x=x0**
Série de Fourier	**FOURIER(f,x,t1,t2,n)**	Série de Fourier d'ordre **n** de **f** par rapport à **x** sur l'intervalle **t1,t2**

Tableau 6-7: Opérateurs et fonctions de FENIX

Nous identifions quatre abstractions dans ce domaine :

- Les équations, décrites ci-dessus
- Les variables indépendantes : ce sont celles qu'on trouve uniquement dans la partie expression des équations. Elles n'ont pas de valeur dérivée et leur valeur doit être fixée par l'utilisateur.

- Les variables indépendantes associées au temps : ce sont quatre variables réservées au système. t représente le temps, min_t et max-t fixent les bornes de l'intervalle d'exécution de la simulation, et dt permet d'accélérer ou de ralentir le passage (simulé) du temps. L'utilisateur doit fixer les valeurs de min-t, max_t, et dt. La variable t est calculée par le système.
- Les variables dépendantes : ce sont celles qui apparaissent dans la partie <variable> des équations. La valeur d'une telle variable est définie par l'expression associée qui est calculée automatiquement par la simulation, sans que l'utilisateur puisse fixer cette valeur.

L'interface des simulations

Une simulation a toujours trois boutons pour gérer l'exécution: démarrer, mettre en pause, ou terminer. Elle a des objets de représentation pour l'entrée des valeurs initiales des variables indépendantes et pour la visualisation des valeurs des variables. Ces objets, pour l'instant, sont des objets génériques.

La traduction du modèle des équations au modèle MARS-S

Pour définir la traduction du modèle du domaine au modèle MARS-S, nous devons décrire une procédure de conversion des abstractions du domaine en éléments du modèle MARS-S.

La construction du modèle

1. Pour chaque variable, définir une propriété du modèle.
2. Pour chaque expression, construire une méthode du modèle pour évaluer l'expression et affecter le résultat à la propriété correspondante.
3. Le graphe d'états du modèle est le même pour toutes les simulations (cf. Figure 6-13).
4. Ajouter les méthodes associées aux états :

Dans la méthode *Pendant* de l'état S_en_execution : appeler les méthodes d'évaluation des équations et incrémenter le temps de delta_t. Si le temps est supérieur au temps maximum, déclencher l' événement E_fin.

Dans la méthode d'*entrée* de l'état S_Init et dans la méthode de *sortie* de l'état S_fin, prévoir l'initialisation des variables dépendantes.

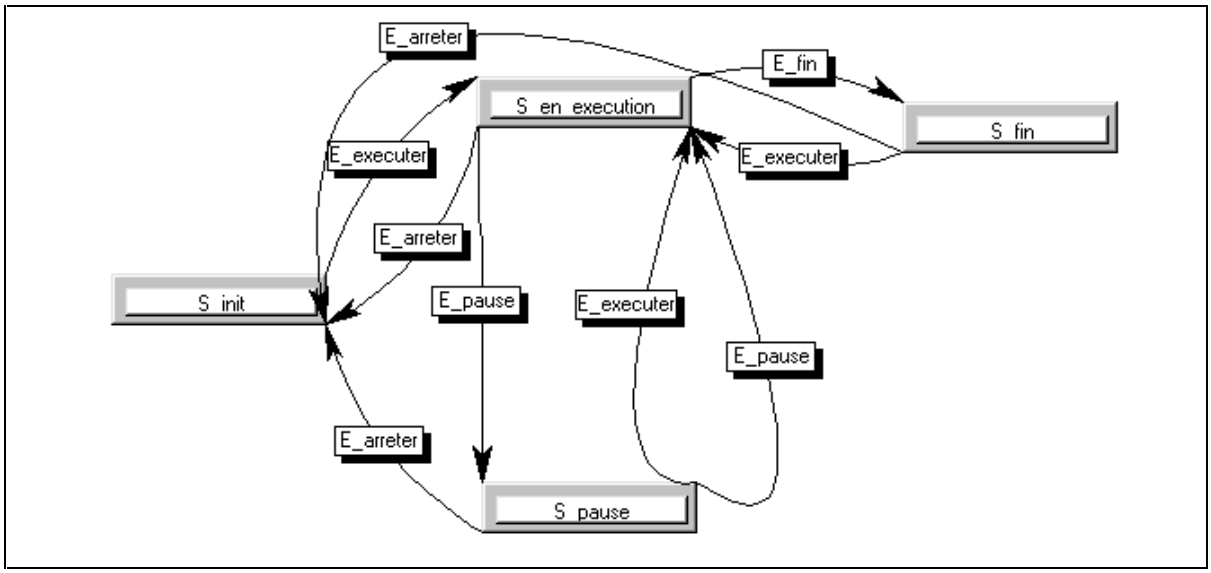


Figure 6-13: Graphe d'état des simulations en FENIX

Les associations

1. Pour chaque bouton de l'interface, construire l'association pour lier l'action sur le bouton avec l'événement correspondant.

2. Pour chaque objet d'entrée, créer une association qui relie une propriété de cet objet à une variable indépendante. Pour chaque objet de sortie, créer une association qui relie la valeur d'une variable à une propriété de cet objet. Les associations entre les objets de présentation et le modèle sont simples et ne nécessitent pas d'appels de conditions ou de méthodes particulières.

La définition des interfaces d'interaction avec l'auteur

Les interfaces d'interaction avec l'auteur doivent permettre d'exprimer le modèle en termes d'équations, d'ajouter des objets d'entrée et de sortie, et de faire les associations correspondantes. Les figures ci-dessous montrent les interfaces proposées. L'outil fournit un éditeur d'équations, un tableau pour fixer les valeurs initiales des variables

indépendantes, un tableau pour fixer les valeurs initiales des variables de temps et une fenêtre pour la construction de l'interface de la simulation. (cf. Figure 6-14). L'interface de la simulation comporte les trois boutons qui permettent d'exécuter, de mettre en pause et d'arrêter la simulation. Des objets de présentation sont disponibles à travers une palette d'objets.

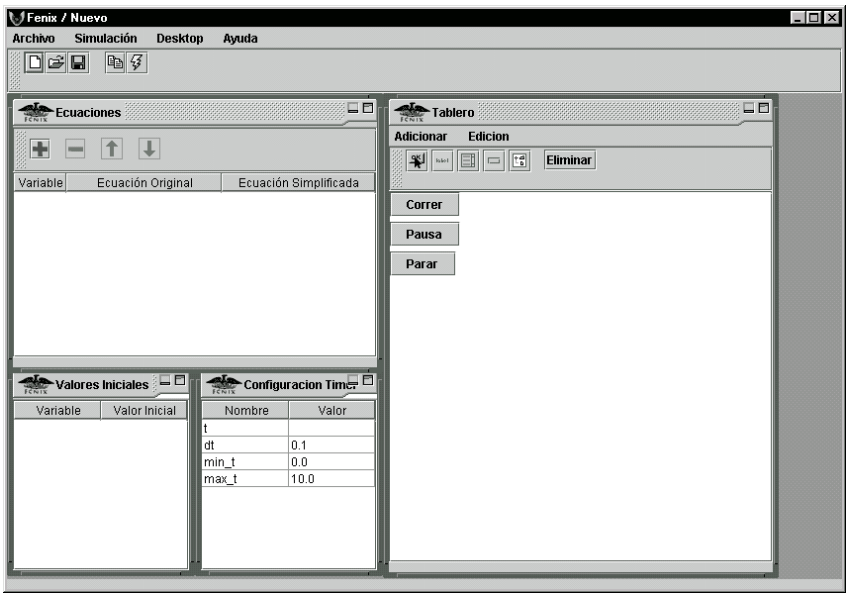


Figure 6-14: Interface d'interaction de FENIX

La Figure 6-15 montre l'interface pour associer les objets de présentation au modèle. L'auteur accède à cette interface à travers un menu local de l'objet de présentation à associer.

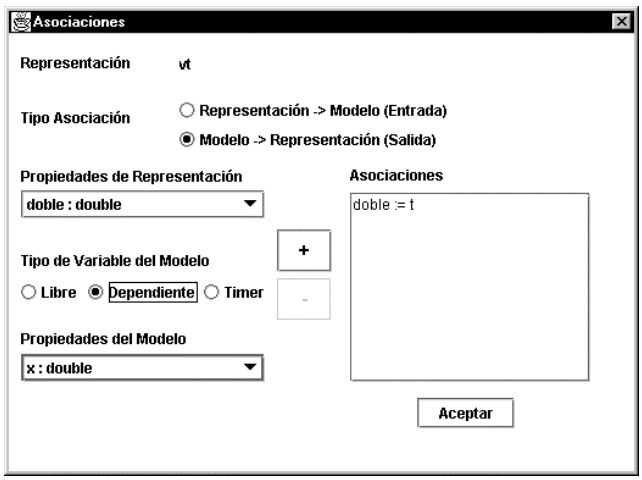


Figure 6-15: Interfaces pour la définition d'associations de FENIX

6.8 Bilan

Les résultats obtenus jusqu'à maintenant sont très encourageants. Tout d'abord nous disposons déjà avec le sous-framework FREAK d'un composant réutilisable pour la construction de différents outils-auteurs basés sur MARS-S.

Nos premières expériences d'utilisation de FREAK ont été satisfaisantes. D'une part, la conception et le développement des prototypes FENIX et SIMARS ont été réalisés dans un temps très court : un mois pour chacun. D'autre part, la maintenance et l'évolution du FREAK pendant le développement des outils ont été aisées, grâce à l'approche de conception basée sur l'architecture et les patrons.

Nous avons constaté par ailleurs que l'utilisation des patrons de conception a permis une meilleure communication entre les concepteurs, les développeurs et les utilisateurs du framework. Les patrons ont fourni un langage commun pour prendre des décisions de conception et pour comprendre le fonctionnement du framework.

Notre expérience de développement de FREAK nous a permis aussi de constater que la réutilisation est un problème essentiellement de *conception*. Le paradigme de programmation par objets ne garantit pas, à lui seul, des conceptions et des codes réutilisables. Notre approche a consisté à définir une architecture orientée vers la réutilisation, puis à continuer avec une conception basée sur les patrons ; cette démarche a été fondamentale dans le développement de FREAK. Nous pensons que ceci favorise grandement l'élaboration de logiciels mieux conçus et ayant de meilleures possibilités de réutilisation, de maintenance et d'évolution.

Partie 3

Une architecture de communication entre simulations et contrôles pédagogiques

Cette partie a pour objectif de proposer une architecture générale de communication permettant de coupler une simulation à une application de contrôle pédagogique.

Le chapitre 7 est consacré à l'identification des services que doit offrir la simulation pour être pleinement exploitée au niveau du contrôle pédagogique. Des travaux similaires sont passés en revue. La démarche suivie afin de recenser ces services consiste à définir un cadre d'analyse et à l'appliquer aux environnements pour la production de simulations pédagogiques présentés dans la partie 1.

A partir des résultats ainsi obtenus, nous proposons dans le chapitre 8 une architecture générale, appelée ARGOS, pour l'observation et le contrôle de simulations pédagogiques. Nous présentons ses composants, et les protocoles mis en œuvre.

7 UNE TYPOLOGIE DES ELEMENTS DE COMMUNICATION

Dans les chapitres 2 et 3, l'état de l'art des systèmes de développement de simulations pédagogiques a permis d'établir que ces simulations s'articulent autour de trois composants : le modèle de la simulation, l'interface de la simulation et le composant pédagogique. Dans ce chapitre, nous regrouperons le modèle et l'interface sous le terme simulation afin de bien souligner la séparation qui existe entre ces deux composants et le composant pédagogique.

Cette séparation prend toute son importance dans ce chapitre puisqu'elle nous permet d'envisager la coopération entre le contrôle pédagogique et la simulation comme étant celle entre un client et un serveur. Outre l'intérêt d'offrir un cadre général à la perception d'un système de développement de simulations, cette approche permet de mettre en évidence la nécessité d'identifier clairement les services que doit offrir une simulation pour être couplée à un contrôle pédagogique, et ce, indépendamment des modèles sur lesquels reposent ces composants.

L'objectif de ce chapitre est d'aboutir à la définition de ces services. Nous décrivons, auparavant, deux projets qui font des propositions reposant sur des principes similaires. Le premier fait l'objet du travail du groupe LTSC de l'IEEE (Learning Technology Standards Committee), rassemblant chercheurs et entreprises, et dont le but est d'établir des standards pour les systèmes d'enseignement par ordinateur [RAD 97]. Le deuxième est le projet SAM étudié dans la première partie. Pour le recensement des services, nous avons suivi

une démarche consistant à analyser, en nous appuyant sur la séparation entre simulation et contrôle pédagogique, un ensemble de systèmes existants. Ce recensement nous permettra ensuite d'établir une classification des services, en fonction du type d'information requise.

7.1 Deux essais de standardisation

Le problème de l'intégration des simulations avec d'autres outils, tels que les outils auteur, n'est pas nouveau. Déjà en 1992, le système SAM, développé dans le projet DELTA D2010, se donnait comme objectif principal l'intégration des outils existant à l'époque, pour construire des environnements d'apprentissage basés sur des simulations [ROS 92].

Dans des domaines particuliers de simulation, nous trouvons des éléments de réponse comme, par exemple, les guides d'interopérabilité des simulations du comité CBT de l'industrie de l'aviation [AIC 95]. Le propos de ces guides est d'établir les conditions nécessaires pour combiner des simulations portant sur le domaine de l'aviation, avec différents systèmes CBT (*Computer Based Training*) afin d'améliorer le processus d'apprentissage et de donner plus de choix aux producteurs de systèmes de formation industrielle.

Dans un spectre plus vaste, le groupe LTSC (*Learning Technology Standards Committee*) travaille depuis 1996 sur la définition de standards sur la technologie pour l'apprentissage [RAD 97]. Le sous-groupe *Tool/Agent Communication* est chargé d'établir des protocoles d'interaction entre des applications.

Dans cette section, nous présentons et analysons l'état d'avancement du groupe *Tool/Agent Communication* du LTSC et les propositions du système SAM.

7.1.1 LTSC (Learning Technology Standards Committee)

Ce comité, sponsorisé par l'IEEE, comprend plusieurs groupes de travail. Son objectif est de développer des standards techniques, des recommandations pratiques et des guides pour les composants logiciels, les outils, les technologies et les méthodologies de conception afin de faciliter le développement, la diffusion, la maintenance et l'interaction des systèmes d'apprentissage/enseignement par ordinateur (systèmes d'EAO).

Le sous-groupe Architecture et Modèle de Référence propose un modèle/architecture de référence pour les systèmes d'EAO basée sur des composants. Une version préliminaire

[FAR 98] définit l'architecture LTSA (*Learning Technology System Architecture*), une architecture abstraite qui tente de spécifier des composants généraux et d'identifier des caractéristiques génériques au vu des systèmes existants.

L'architecture LTSA identifie dans un système d'EAO quatre composants de processus, deux de stockage et sept types de flux d'information entre composants (cf. Figure 7-1). L'apprenant et l'instructeur du système négocient le style d'apprentissage. L'apprenant est observé et évalué. L'évaluation produit des données sur la performance. L'instructeur du système vérifie l'évaluation de l'apprenant, ses résultats historiques et les objectifs d'apprentissage futurs. L'instructeur du système cherche dans la base de connaissances le contenu à délivrer à l'apprenant.

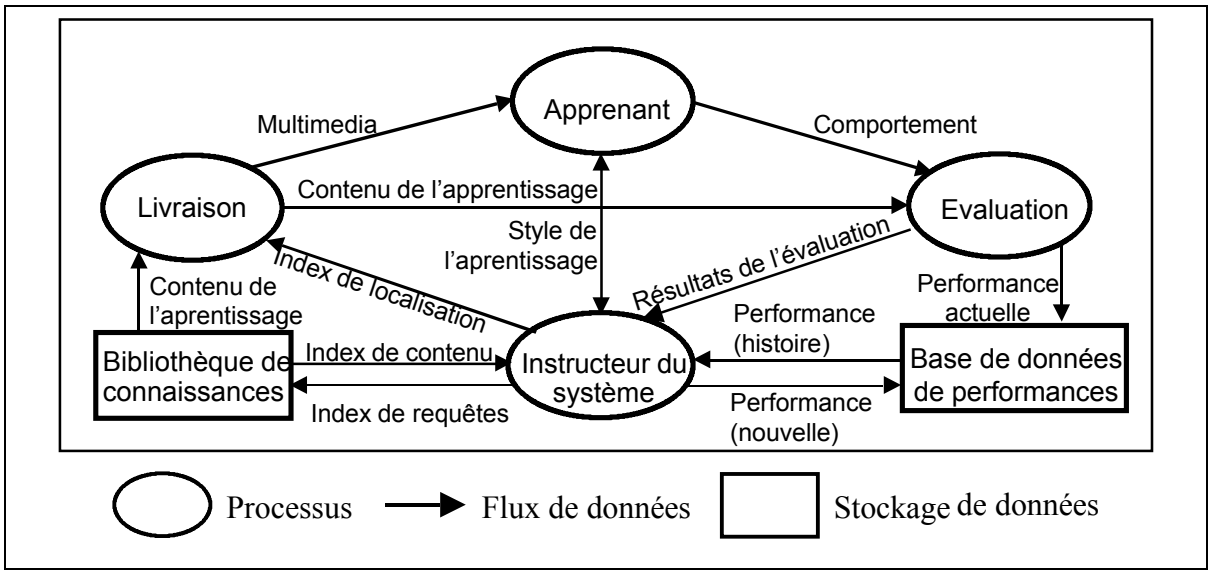


Figure 7-1: Composants de l'architecture LTSA

Cette architecture peut décrire la plupart des architectures des systèmes EAO existants. Dans la mesure où il s'agit d'une architecture abstraite, la correspondance peut ne pas être parfaite. Un système peut combiner des portions de composants en un seul composant et/ou des parties de processus peuvent être sous la responsabilité des personnes et non pas du système. Néanmoins, il est possible de faire une séparation conceptuelle des composants.

Cette architecture sert de cadre de référence à tous les groupes de travail du LTSC. Nous nous intéressons particulièrement au travail du groupe Communication Outil/Agent, qui étudie les systèmes EAO dans lesquels interviennent des outils informatiques et des agents d'instruction. Un agent d'instruction est une application capable de guider un apprenant

vers un but d'apprentissage. Les outils informatiques sont, par exemple, les outils de bureautique, les logiciels éducatifs, les simulations, etc.

Dans ces systèmes, l'apprenant utilise différents outils pour accomplir un objectif d'apprentissage. Ces outils peuvent être complétés par des logiciels spéciaux qui permettent d'expliquer les objectifs à l'apprenant, de délivrer des consignes, d'introduire des concepts, de donner de l'aide en ligne ou de guider l'apprenant pendant la résolution d'un problème.

Voici quelques exemples de tels environnements d'apprentissage: l'apprenant utilise un tableur pour résoudre des problèmes algébriques et un agent d'instruction, externe au tableur, guide l'apprenant [RIT 96]; l'apprenant utilise une calculatrice financière pour calculer les intérêts d'un emprunt et un agent d'instruction l'assiste dans l'utilisation de la calculatrice; l'apprenant utilise une simulation d'un système pour inférer ses propriétés, et un agent d'instruction lui suggère des tests à faire.

Le groupe de travail Communication Outil/Agent a pour but de spécifier un protocole pour l'interaction entre des outils (tableurs, simulations, accessoires, logiciels d'EAO, etc.) et des agents d'instruction. Le protocole de communication en cours de spécification annonce les objectifs ci-dessous :

- Permettre à l'agent d'instruction d'observer ce que l'apprenant fait dans les outils.
- Permettre à l'agent d'instruction de donner un feed-back à l'apprenant.
- Transmettre à l'outil les demandes de l'agent d'instruction.
- Permettre aux agents d'instruction de communiquer entre eux, pour partager leurs informations dans le but d'évaluer ou de guider l'apprenant.
- Établir un standard pour référencer les objets des outils.

Dans l'architecture LTSA, ce groupe est concerné principalement par les processus évaluation et instructeur du système, par le stockage de registres et par les flux de données de comportement, de résultats d'évaluation, de performance et de localisation (cf. Figure 7-2).

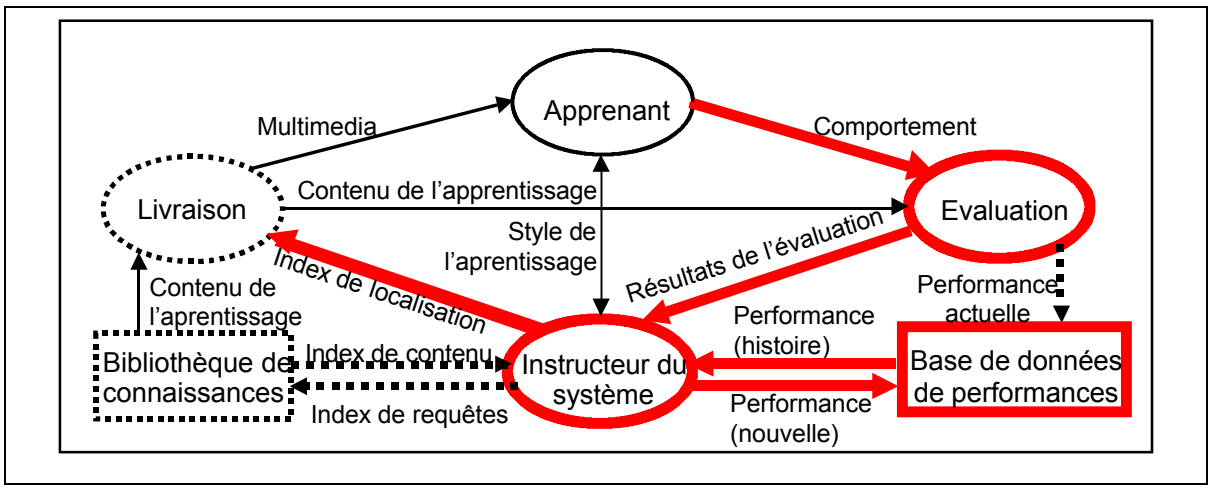


Figure 7-2: Aspects de l'architecture LTSA concernant le groupe Communication Outil/Agent

Les trois caractéristiques que le protocole cherche à implanter sur les outils et les agents sont :

- La capacité d'être observable : capacité d'un outil ou agent d'informer automatiquement des événements survenants en son sein.
- La capacité d'être scriptable : capacité d'être modifiable en utilisant un langage de commande.
- La capacité d'être inspectable : capacité d'un outil ou agent de délivrer une information précise suite à une demande explicite d'un autre outil ou agent.

La différence entre la capacité d'être observable et la capacité d'être inspectable est que dans le premier cas l'outil ou l'agent fournissent automatiquement l'information alors que dans le deuxième cas l'information est donnée suit à une demande explicite.

Pour garantir l'indépendance entre les outils et les agents d'instruction, il est nécessaire de satisfaire trois hypothèses :

- L'architecture interne de l'agent d'instruction n'est pas connue.
- Les outils ne font pas de suppositions sur la façon dont l'agent d'instruction utilise les informations.
- Le fonctionnement des outils ne dépend pas des agents d'instruction.

Les outils et les agents d'instruction sont conçus comme des applications qui contiennent des objets[8]. Un objet peut contenir d'autres objets. Les messages échangés entre des outils et des agents font références aux objets qu'ils contiennent. Ainsi, la problématique du groupe LTSC tourne autour de deux nécessités essentielles : La première est d'avoir un standard pour référencer les objets des outils et des agents. La deuxième est d'établir un standard pour les messages que peuvent échanger outils et agents d'instruction. Pour l'instant, le groupe a une proposition pour référencer les objets et une liste de messages pour les outils et pour les agents d'instruction. Décrivons-les brièvement.

7.1.1.1 Proposition de standard de référence d'objet

L'objectif principal est de définir une méthode pour référencer un objet qui permette à un agent d'instruction d'identifier sans ambiguïté les objets des outils. Pour cela, ils ont défini la notion d'amorce d'objet (*object proxy* en anglais) et des notations standards. Une amorce d'objet est une représentation d'un objet dans le domaine conceptuel d'une application.

Une amorce d'objet permet d'identifier un objet par une référence unique (OPID) ou par une description (OPD). Les références uniques sont générées à partir des informations comme le temps, la machine, des compteurs, etc. Les adresses URL dans le WEB sont des exemples de ce type de référence. Les références par description identifient un objet par son nom ou par sa position dans l'objet qui le contient: par exemple, le paragraphe 3 dans le document 2, la cellule 2 de la colonne 3 ou le rectangle « Boîte A» du dessin 1.

Les outils respectant ce standard doivent être capables de fournir des OPIDs pour tous leurs objets et doivent être capables de comprendre des OPDs qui décrivent des objets en suivant ces règles.

7.1.1.2 Messages Agent-Outil et Outil-Agent

Le Tableau 7-1 résume les messages proposés, en fonction des caractéristiques que le protocole désire donner aux agents et aux outils.

---

[8] La notion d'objet dans le contexte de ce groupe est une abstraction du domaine de l'outil et n'implique pas nécessairement une implémentation orientée objets.

Pour observer un outil, l'agent d'instruction envoie le message `Start Observing` à l'outil en spécifiant le type de déclencheur, la granularité et la période. Le type de déclencheur spécifie ce que l'outil doit notifier à l'agent d'instruction : les activités de l'utilisateur, les changements dans le modèle de données et le temps. La granularité spécifie le niveau des événements à observer : s'ils sont du niveau interface utilisateur (*mouse click at 254,320*) ou s'ils sont de niveau sémantique, c'est-à-dire s'ils font référence aux objets du domaine de l'outil (*property gravity of object moon set to 5*). La période spécifie l'intervalle d'observation quand le type de déclencheur est le temps. Après le message `Start Observing`, chaque fois qu'un événement observé arrive, l'outil envoie un message `Note` à l'agent d'instruction.

Pour modifier les objets de l'outil, le groupe a pour l'instant défini un ensemble de messages de base que l'outil doit savoir traiter. Ces messages permettent à un agent d'instruction de :

- Réaliser des opérations sur l'outil : quitter l'application et annuler la dernière action.
- Réaliser des opérations de base sur les objets de l'outil : créer, éliminer, changer la valeur d'une propriété, incorporer dans un autre objet, ouvrir, fermer, imprimer, sauvegarder et sélectionner.
- Donner un feed-back en utilisant l'outil : signaler un objet, mettre en évidence un objet sur l'écran (flag/unflag) et montrer un message à l'apprenant.

Pour inspecter l'outil, l'agent d'instruction utilise le message *get property*.

Pour sa part, l'agent d'instruction peut être observé par une autre application (un autre agent d'instruction ou une application de gestion de curriculum). Pour cela, l'agent d'instruction envoie des messages pour actualiser l'évaluation, pour informer que l'apprenant a fini un problème et pour informer que l'agent a fini de répondre à un événement observé.

	Outil	Agent
Capacité d'être observable	**Start Observing** (level, period, granularity) **Stop Observing** **Note Creation** (object, parameters) **Note Deletion** (object) **Note Property Set**(property, object, value) **Note Insert** (object, container, position) **Note Remove**(id, object, container, position) **Note Open** (object) **Note Print**(object, printer) **Note Quit** **Note Save**(object) **Say** (message)	**Update Assesment**(mesure, amount) **End State Reached** **Reponse Completed**(reponse id)
Capacité d'être scriptable	**Create**(object, parameters) **Delete**(object) **Set Property** (property, object, value) **Insert** (object, container, position) **Remove**(id, object, container, position) **Open** (object) **Print**(object, printer) **Quit** **Save**(object) **Flag**(object) **Unflag**(object) **Point To**(object) **Undo**(object) **Send Message**(message, objects to point, format, attach object) **Select**(object) **Start Activity**(activity)	***Start Problem****(user name,initial state, goal state)* **Get Hint**(user name, type, selected object)
Capacité d'être inspectable	**Get Property** (property,object)	**Get next step**(user name) **Get Property** (property,object)

Italique: méthodes des outils **Non italiques: méthodes de l'agent d'instruction**

Tableau 7-1: Messages Outil/Agent proposés

7.1.1.3 Similarités entre nos travaux et le travail du LTSC

Nous trouvons plusieurs similarités entre nos travaux et le travail du LTSC. La première est l'identification de deux types d'applications : les guidages pédagogiques (nos applications de contrôle pédagogique et ses agents d'instruction) et les outils pour l'apprenant (nos simulations et ses outils d'application).

D'autre part, comme nous, le groupe Outil/Agent s'occupe des aspects d'intégration des applications et des outils pédagogiques. La différence essentielle est la portée des travaux. Nous nous intéressons seulement aux simulations et aux applications de contrôle pédagogique de simulations, et non à tous les types d'applications. Dans ce sens, il nous semble qu'il faudra encore beaucoup de temps pour que le groupe Outil/Agent arrive à des résultats concrets et applicables.

Un troisième aspect est la similarité dans les objectifs de la communication : l'observation et le contrôle, dans un but pédagogique, d'outils existants. Établir des standards qui permettent de faciliter la communication apportera des bénéfices à long terme pour la configuration d'environnements d'apprentissage plus adaptés aux besoins des utilisateurs.

7.1.2 SAM

Nous avons décrit dans la partie 1 l'architecture proposée par le système SAM pour ajouter un support pédagogique aux simulations (cf. §2.2.1). Nous décrirons ici seulement la technique utilisée pour intégrer la simulation dans l'environnement SAM.

Tout d'abord, SAM offre à l'auteur du contrôle pédagogique un ensemble réduit d'actions qu'il peut faire sur une simulation [ROS 93] :

- `event <nomev>` : Génère l'événement <nomev> dans la simulation.
- `get <nomvar>` : Obtient la valeur de la variable <nomvar> de la simulation.
- `set <nomvar> = <valeur>` : Met la valeur <valeur> dans la variable <nomvar> de la simulation.

- monitor <nomvar> : Démarre la surveillance des changements de la variable <nomvar> de la simulation.
- unmonitor <nomvar> : Arrête la surveillance des changements de la variable <nomvar> de la simulation.
- save <state.id> : Fait une sauvegarde de l'état de la simulation sous le nom <state.id>.
- load <state.id> : Remet la simulation dans l'état enregistré sous le nom <state.id>.

Rappelons-nous que pour construire des contrôles pédagogiques en SAM, l'auteur écrit des scripts dans un langage spécial (cf. §3.3.4). Dans un script, l'auteur peut inclure les actions listées ci-dessus. L'exécution des scripts est faite par un interpréteur. Cet interpréteur interagit avec la simulation en utilisant d'une part une table de contrôle et communication (table C&C) et d'autre part une bibliothèque dynamique (DLL) spécifique. La table C&C contient la traduction de chaque action sur la simulation en appels à des fonctions DLL. La DLL, spécifique pour l'outil avec lequel la simulation est faite, est chargée de faire la communication avec la simulation. Chaque fois que l'interpréteur interprète une action sur la simulation, il consulte la table C&C et invoque la séquence de fonctions DLL spécifiée pour l'action. Ce processus est décrit dans la Figure 7-3.

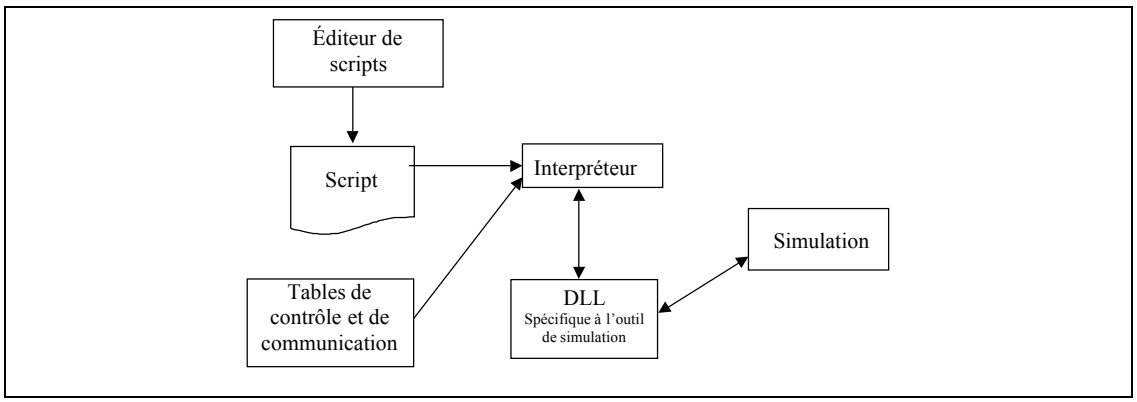


Figure 7-3: SAM : Réalisation et exécution de programmes pour le contrôle de simulations

L'architecture proposée par SAM a démontré la faisabilité d'intégrer des simulations avec autres applications. Pour chaque outil de simulation, il suffit de construire une entrée dans la table C&C et une DLL. Néanmoins, l'architecture présente deux contraintes importantes :

1. Le mécanisme de communication utilisé est lié au système d'exploitation et à une technique spécifique (DLLs sur Windows et AppleEvents sur Macintosh).

2. L'ensemble réduit de services utilisables par la simulation pédagogique.

L'idée de traduction des services à travers une table et un langage concret nous semble néanmoins intéressante pour réutiliser des simulations qui fournissent des services spécifiques.

7.2 Une analyse de besoins basée sur l'existant

Pour établir l'inventaire des services qu'une simulation fournit à un contrôle pédagogique, il nous faut définir un cadre d'analyse qui, d'une part, couvre toutes les situations d'interconnexion entre ces deux composants et, d'autre part, fournisse les critères permettant d'identifier les services. Nous allons d'abord définir ce cadre, puis nous l'appliquerons dans les sections suivantes aux systèmes SAM, OASIS, SimQuest, Rides, SimBest, Interact et Instructional Simulator.

Environnements où intervient un contrôle pédagogique

Afin de mener à bien notre étude, il nous faut observer le fonctionnement de chaque système. Or de tels systèmes peuvent être exploités à des fins différentes, et de ce fait mettre à contribution le contrôle pédagogique de diverses façons. Il convient donc pour l'analyse de distinguer clairement les phases d'exploitation d'un contrôle pédagogique.

Nous distinguerons deux types d'environnements possibles pour le contrôle pédagogique: l'environnement auteur du contrôle pédagogique et l'environnement de l'apprenant (cf. 3.5). Ceci nous permet d'établir une démarche générale pour notre analyse. En effet, pour un système donné, recenser les services offerts au contrôle pédagogique par la simulation revient à observer :

- Premièrement, la phase de création du contrôle pédagogique.
- Deuxièmement, la phase d'exécution du contrôle pédagogique.

Il reste ensuite à rassembler et à synthétiser les résultats ainsi obtenus.

La démarche générale étant précisée, il convient maintenant de fixer la méthode d'observation qui nous permettra d'identifier les services impliqués dans chaque phase.

Caractéristiques observées pour l'identification des services

Dans chacune des phases, il est nécessaire de s'interroger sur les types d'informations et de services qu'une simulation doit rendre publics pour pouvoir être contrôlée pédagogiquement par une autre application. Nous les classerons selon les caractéristiques proposées par le groupe Outil/Agent du comité LTSC : la scriptabilité, l'observabilité et l'inspectabilité. Nous ajoutons une quatrième caractéristique pour les services qui permettent au contrôle pédagogique de référencer les différents composants de la simulation.

7.2.1 SAM

La philosophie du système SAM est de faciliter l'intégration de différents outils pour créer des environnements d'apprentissage basés sur des simulations (Simulation based learning environments, SBLE). Les types d'outils que SAM cherche à intégrer sont des outils auteurs, des outils de simulation, des outils de planification et de contrôle de l'instruction, des bases de données, des outils de traitement de textes, etc.

L'architecture de SAM, présentée en détail dans la partie 1, définit explicitement les services que doit fournir une simulation pour qu'elle puisse être intégrée dans un SBLE [ROS 93] [ROS 94]. Nous identifions le type de chaque service, et la phase dans laquelle il est utilisé.

Service	Phase d'utilisation	Type de service
1. Générer un événement. SAM définit deux événements de base : démarrer la simulation et arrêter la simulation.	Exécution	Scriptabilité
2. Capturer la valeur d'une variable de la simulation.	Exécution	Inspectabilité
3. Changer la valeur d'une variable de la simulation.	Exécution	Scriptabilité
4. Initialiser la surveillance d'une variable de la simulation, c'est-à-dire être notifié chaque fois que la variable change.	Exécution	Observabilité
5. Stopper la surveillance d'une variable.	Exécution	Observabilité
6. Capturer l'état de la simulation.	Exécution	Inspectabilité
7. Changer l'état de la simulation.	Exécution	Scriptabilité

Tableau 7-2: Services requis par le contrôle pédagogique de SAM

7.2.2 OASIS [CAG 97] [COR 97] [COR 97ª] [COR 98]

Comme nous l'avons vu dans la partie 1, le contrôle pédagogique d'OASIS est basé sur la notion de scénario (ou exercice). On rappelle qu'un scénario est composé d'une situation initiale, d'un ensemble d'étapes, d'un ensemble de contrôles globaux et d'une réactivité.

Une étape est composée d'une situation, d'une réactivité et d'un ensemble de contrôles. Les contrôles globaux et les contrôles d'étapes sont composés d'une situation et d'une réactivité. Une situation est une description du modèle de la simulation, en termes des valeurs des propriétés du modèle.

7.2.2.1 Définition du contrôle pédagogique

L'environnement auteur de contrôles pédagogiques fourni par OASIS est l'éditeur de scénarios.

En premier lieu, pour définir un contrôle pédagogique, l'auteur doit sélectionner les propriétés du modèle qui sont pertinentes pour le scénario. La simulation doit donc lui fournir la liste de ses propriétés.

En second lieu, l'auteur doit définir les situations correspondant à l'état initial, à chaque étape, et à chaque contrôle (global ou interne à une étape). La Figure 7-4[9] décrit la séquence d'actions que l'auteur doit effectuer pour définir les différents composants du scénario, ainsi que la séquence d'actions réalisées par l'éditeur de scénarios.

La méthode adoptée pour la définition de scénarios repose sur un mécanisme de démonstration : l'auteur doit agir sur la simulation pour la mettre dans la situation souhaitée puis enregistrer la situation grâce à une commande ou un bouton fournis par l'environnement. Cet enregistrement capture automatiquement les valeurs des propriétés du modèle pertinentes pour le scénario.

---

[9] Pour décrire les séquences d'actions, nous utilisons une notation similaire aux schémas de séquence d'UML. Chaque diagramme montre les acteurs (Simulation, Contrôle Pédagogique, Auteur et Apprenant) et les interactions entre eux (flèches) dans le temps.

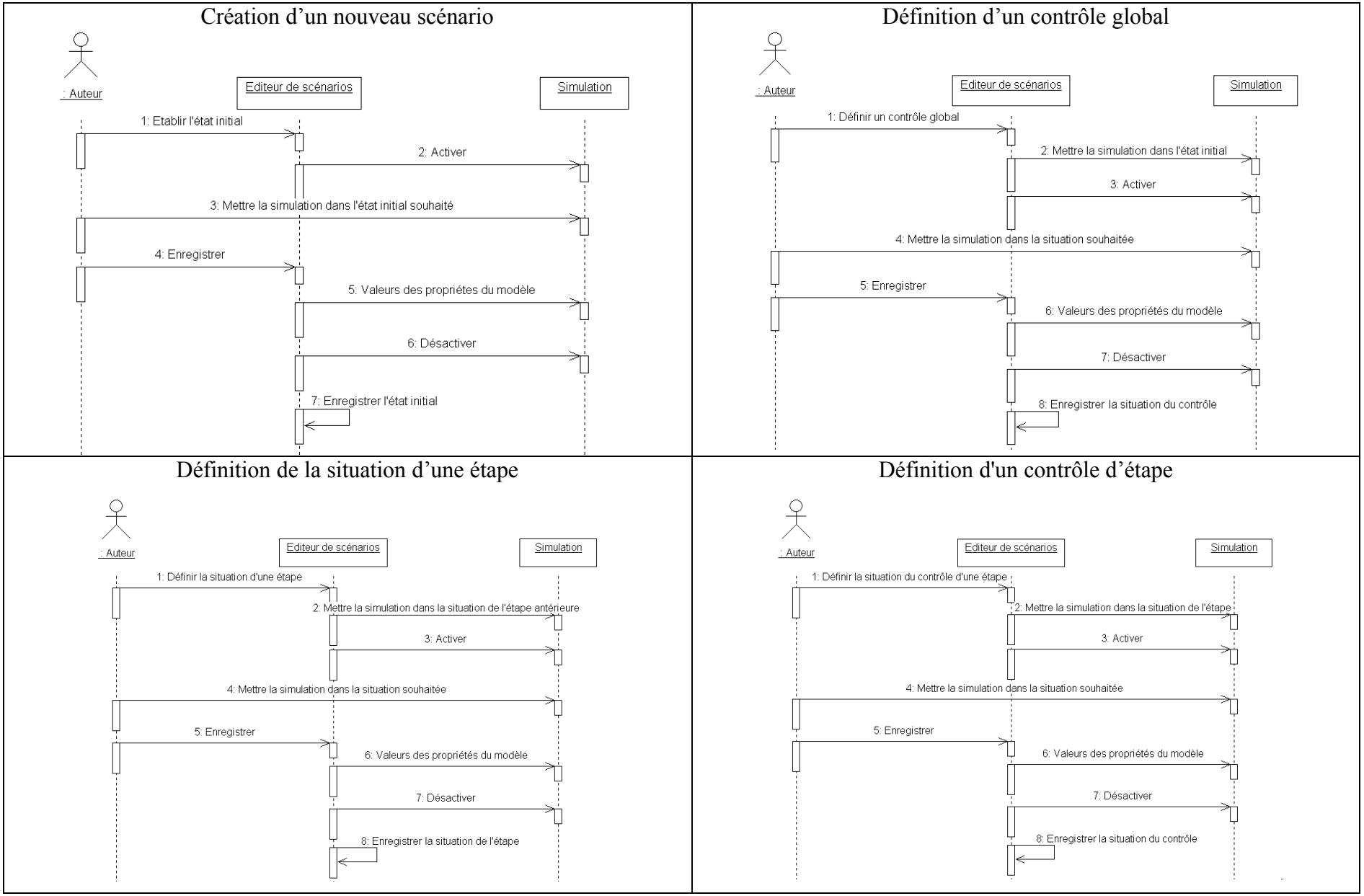


Figure 7-4: Diagrammes de séquence pour la définition de situations en OASIS

Il apparaît donc que deux types de coopération existent entre l'éditeur et la simulation. L'éditeur de scénarios doit rendre la simulation active ou inactive pour l'auteur. Il doit enregistrer la situation courante de la simulation à la demande de l'auteur ou remettre la simulation dans une situation déjà enregistrée.

Nous détectons ici trois besoins au niveau de l'éditeur de scénario :

- Obtenir une liste de propriétés du modèle de la simulation.
- Activer/désactiver la simulation.
- Capturer/restaurer une situation de la simulation à travers les valeurs de propriétés du modèle.
- Capturer/restaurer l'état de la simulation, les valeurs des propriétés et l'état de l'interface.

7.2.2.2 Exécution du contrôle pédagogique

Comme nous l'avons expliqué dans la partie 1, chaque scénario défini par l'auteur peut être exécuté par l'apprenant soit en mode entraînement, soit en mode évaluation.

Rappelons la différence essentielle entre ces deux modes: dans le mode entraînement, le déroulement de l'exercice est assisté étape par étape; en mode évaluation l'apprenant est laissé libre d'agir comme il veut avec la seule contrainte d'une limite de temps. Le mode entraînement n'est autre qu'une exécution incrémentale du scénario ; il permet à l'apprenant de pouvoir être évalué à chaque étape et de répéter cette étape s'il le souhaite en cas d'échec. Le mode évaluation, quant à lui, délivrera simplement les résultats de l'évaluation de l'apprenant à la fin de l'exercice sous forme d'un rapport sur le travail effectué.

La Figure 7-5 montre le déroulement des scénarios dans les deux modes. Notons que dans tous les cas, l'application de contrôle pédagogique inspecte périodiquement les valeurs des propriétés du modèle, pour vérifier si la situation de l'étape courante a été atteinte ou si un contrôle a été déclenché.

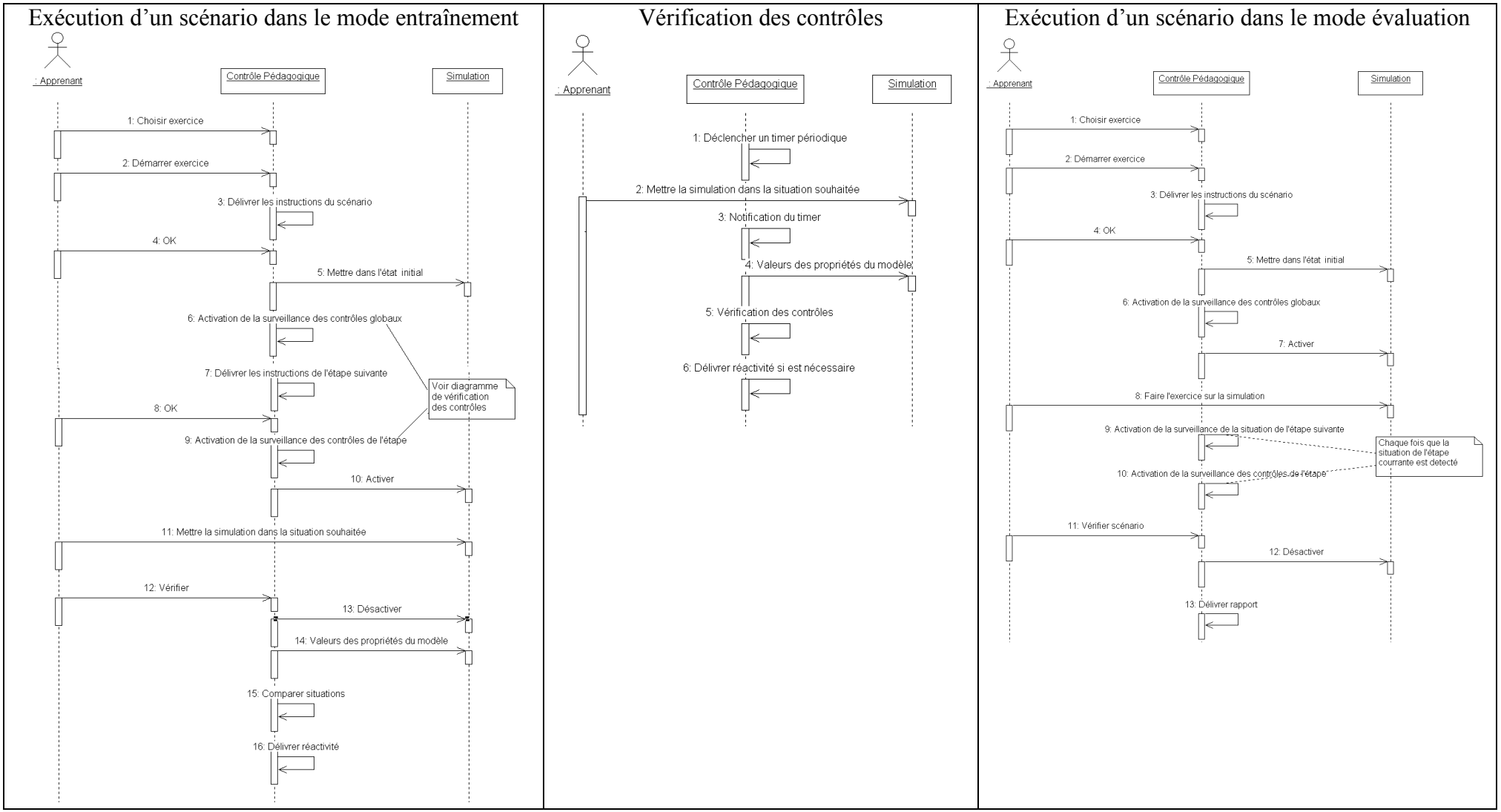


Figure 7-5: Séquence d'activités pour l'exécution d'un scénario en OASIS

La coopération entre la simulation et le contrôle pédagogique consiste à remettre la simulation dans une situation (état initial ou situation d'une étape), au moment de l'activation (ou de la désactivation) de la simulation, au moment de la vérification des situations et au moment de la vérification des contrôles.

Résumons les besoins pour l'exécution d'un scénario:

1. La capacité d'activer/désactiver la simulation depuis l'application de contrôle pédagogique.
2. La capacité de capturer/restaurer une situation de la simulation à travers les valeurs de propriétés du modèle.

7.2.2.3 Synthèse des services

Nous venons d'analyser la définition et l'exécution des scénarios en OASIS. Nous pouvons conclure que, pour le type de contrôle pédagogique proposé, la simulation doit fournir cinq services à un environnement de contrôle pédagogique (cf. Tableau 7-3).

Service	Phase d'utilisation	Type de service
1. Fourniture d'une liste de propriétés du modèle que le contrôle pédagogique peut observer et modifier.	Création	Référence
2. Activation de la simulation par le contrôle pédagogique.	Création Exécution	Scriptabilité
3. Désactivation de la simulation par le contrôle pédagogique.	Création Exécution	Scriptabilité
4. Récupération par le contrôle pédagogique des valeurs des propriétés du modèle qui appartiennent à la liste.	Création Exécution	Inspectabilité
5. Changement par le contrôle pédagogique des valeurs des propriétés du modèle qui appartiennent à la liste.	Création Exécution	Scriptabilité

Tableau 7-3: Services requis par le contrôle pédagogique d'OASIS

7.2.3 SimQuest

Comme nous l'avons vu dans la première partie, SimQuest propose un contrôle pédagogique basé sur des composants d'instruction (IM : instructional measures) que l'auteur peut importer d'une bibliothèque, puis instancier. L'originalité de SimQuest est qu'il inclut explicitement dans son architecture un module (*Model Wrapper*) dédié à la

communication entre les composants d'instruction, les modèles de simulation et les éléments d'interface.

Le *Model Wrapper* est conçu afin de fournir l'accès aux informations du modèle aussi bien à l'interface qu'aux composants d'instruction [DEJ 96]. Pour ce faire, il met à disposition une liste des variables du modèle et une description de la composition du modèle en sous-composants. Les noms qui figurent dans le *Model Wrapper* sont des noms externes au modèle. Ils ne correspondent pas nécessairement aux noms internes des éléments et des variables du modèle.

7.2.3.1 Définition du contrôle pédagogique

L'environnement auteur de SimQuest est constitué par l'éditeur de composants d'instruction.

Pour définir un composant d'instruction, l'auteur doit d'abord sélectionner un *Model Wrapper*. L'éditeur de composants d'instruction extrait du *Model Wrapper* la description des variables de la simulation et la composition du système. Avec cette information, l'éditeur permet la définition de tous les types de composants d'instruction. Le modèle et l'interface de la simulation ne sont pas nécessaires pour la phase de définition, car le *Model Wrapper* fournit les informations requises.

Ayant choisi un *Model Wrapper*, l'auteur spécifie les variables du modèle qui vont être utilisées par le composant d'instruction. Chaque type de composant d'instruction utilise les données avec différents propos (cf. Tableau 7-4) [DEJ 96].

Nous pouvons observer que dans la phase de définition du contrôle pédagogique de SimQuest, les besoins d'information sont minimaux : d'une part, avoir accès à la liste de variables du modèle et d'autre part, avoir une description en composants du système.

	Composant d'instruction	Information utilisée	Mode d'utilisation
Exercices (*Assignments*)	**Recherche** Exploration libre de l'élève avec une question à choix multiple à répondre. L'auteur définit la question et les réponses.	Les noms des variables du modèle.	L'auteur inclut les noms des variables dans la question et dans les réponses.
	Explicitation Exploration de l'élève avec une question à choix multiple plus un ensemble d'états initiaux pour démarrer l'exploration. L'auteur définit la question, les réponses et les états initiaux.	Les noms des variables du modèle.	L'auteur inclut les noms des variables dans la question et dans les réponses. Pour définir un état initial, l'auteur donne une valeur à chaque variable du modèle et spécifie si la variable est visible, invisible ou modifiable par l'élève.
	Prédiction L'élève doit prédire les valeurs des variables quand la simulation s'arrête. L'état initial est défini par l'auteur.	Les noms des variables du modèle.	L'auteur définit un état initial. L'auteur choisit les variables que l'élève doit prédire. Pour chaque variable, il définit les déviations permises à l'élève.
	Opération A partir d'un état initial, l'élève doit atteindre un état final, en respectant les contraintes fixées par l'auteur.	Les noms des variables du modèle.	L'auteur définit un état initial (Cf. explicitation). L'auteur choisit les variables considérées pour l'état final. Pour chaque variable choisie, il définit la valeur maximale et minimale qu'elle peut prendre dans l'état final. L'auteur procède de même pour les variables sur lesquelles portent des contraintes.
	Explications Explications pour l'élève sous différents formats (sonore, graphique ou textuel).	Les noms des variables du modèle. Les noms des composants du système.	L'auteur inclut les noms des variables ou des composants dans l'explication. Les explications peuvent êtres associés aux réponses des autres exercices.

Tableau 7-4: Utilisation d'information par les composants d'instruction de SimQuest

7.2.3.2 Exécution du contrôle pédagogique

Pendant l'exécution de l'environnement, une version d'exécution du *Model Wrapper* gère l'interaction entre les différents modules de l'application [DEJ 96]. Elle est chargée, en particulier, de mémoriser une valeur pour chaque variable du modèle et d'informer les composants d'instruction des changements de valeurs.

Bien que la bibliographie [DEJ 94] [DEJ 96] [JOO 96] ne fournisse pas les détails du fonctionnement lors de l'exécution, nous nous sommes efforcés de le reconstituer dans la Figure 7-6. Chaque diagramme décrit, pour un type de composant d'instruction, les interactions devant exister lors de l'exécution entre la simulation, le contrôle pédagogique et l'apprenant.

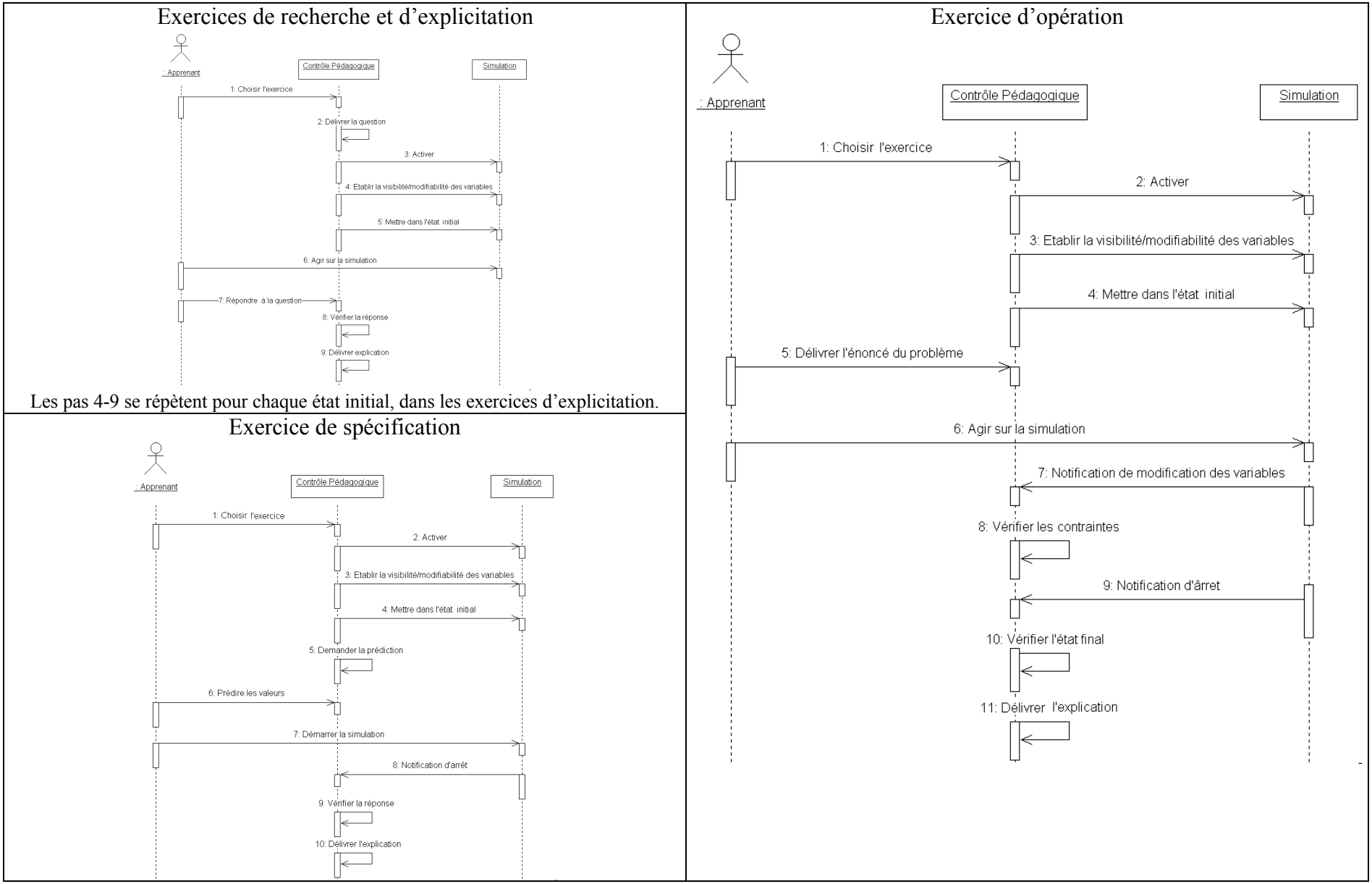


Figure 7-6: Diagrammes d'interaction en exécution des composants d'instruction de SimQuest

Nous pouvons observer que les services requis par le contrôle pédagogique sont :

- Mettre la simulation dans un état initial, à partir des valeurs des propriétés du modèle, depuis le contrôle pédagogique.
- Activer la simulation depuis le contrôle pédagogique.
- Etre notifié de changements de valeur des variables du modèle.
- Etre notifié de certaines actions de l'apprenant sur l'interface, par exemple un clic sur le bouton Stop.
- Préciser la visibilité dans l'interface d'une variable du modèle depuis le contrôle pédagogique.
- Préciser si une variable du modèle est modifiable par le contrôle pédagogique.

7.2.3.3 Synthèse des services

Les services qu'une simulation doit fournir à une application de contrôle pédagogique similaire aux composants d'instruction de SimQuest sont résumés dans le Tableau 7-5.

Service	Phase d'utilisation	Type de service
1. Fourniture d'une liste de propriétés du modèle que le contrôle pédagogique peut observer et modifier.	Création	Référence
2. Fourniture de la composition du système (éléments, parties, etc).	Création	Référence
3. Activation de la simulation par le contrôle pédagogique.	Exécution	Scriptabilité
4. Changement par le contrôle pédagogique des valeurs des propriétés du modèle qui appartiennent à la liste fournie par le service 1.	Exécution	Scriptabilité
5. Changement par le contrôle pédagogique de la visibilité dans l'interface d'une propriété du modèle.	Exécution	Scriptabilité
6. Changement par le contrôle pédagogique de la modifiabilité d'une propriété du modèle.	Exécution	Scriptabilité
7. Notification des modifications des propriétés du modèle.	Exécution	Observabilité
8. Notification des actions de l'élève sur l'interface de la simulation.	Exécution	Observabilité

Tableau 7-5: Services requis par le contrôle pédagogique de SimQuest

7.2.4 RIDES

Dans la première partie, nous avons vu que RIDES reprenait partiellement les principes du processus de génération d'entraînement par ordinateur (appelé CGT pour *Computer Generated Training*) proposé par Towne [TOW 95]. A la différence d'OASIS et de SimQuest, RIDES donne à l'auteur la possibilité de générer automatiquement certains types d'exercices.

En nous basant sur la documentation du système [MUN 95] et [MUN 95a], nous allons maintenant analyser les processus de définition et d'exécution des exercices.

7.2.4.1 Définition du contrôle pédagogique

Selon le principe CGT, l'auteur doit, pour générer automatiquement des exercices, fournir à RIDES des informations complémentaires à celles de la simulation. De ce fait, le processus de définition du contrôle pédagogique englobe une phase de capture de connaissances d'expert et une phase de création d'exercices.

L'acquisition de connaissances complémentaires

Pour réaliser cette première étape, RIDES fournit deux outils: un éditeur d'unités de connaissance (Knowledge Units editor) et un éditeur de relations d'influence (Affect Relationships editor).

Une unité de connaissance contient une référence à un objet de la simulation, une collection de discussions (textes) créées par l'auteur et classées par thèmes, une liste d'exercices associés, une liste d'unités de connaissances relatives et une liste de mots clés.

L'éditeur de relations permet de définir deux listes pour un objet: une liste des objets qui sont influencés par cet objet, et une liste des objets qui agissent sur lui.

Cette phase d'acquisition nécessite seulement une liste des objets de la simulation.

La création des exercices

RIDES permet de créer différents types de support pédagogique pour une simulation : exercices types (Patterned Exercises), exercices créés de façon libre et leçons (Custom Lessons).

La création de chaque type s'appuie sur la notion de leçon qui est, dans RIDES, une séquence d'items d'instruction. Un item d'instruction est un groupe d'items ou un item d'instruction primitif; ce dernier constitue l'unité élémentaire de transaction entre RIDES et l'apprenant.

Dans le cas des exercices générés automatiquement, la définition d'une leçon consiste à définir la séquence d'items qui la composent, en précisant les caractéristiques de chaque item. Citons quelques caractéristiques d'un item :

- Le mode d'exécution : démonstration, pratique ou évaluation. L'auteur peut définir pour un item les modes d'exécution où l'item doit apparaître.
- Un texte à montrer à l'apprenant pour chaque mode d'exécution. Ces textes peuvent inclure des références aux attributs des objets de la simulation.
- Des caractéristiques telles que la note de l'item, le nombre d'essais que peut faire l'apprenant, le fait que l'apprenant doive ou non appuyer sur un bouton spécifique pour continuer, etc.
- Selon l'item, l'auteur peut définir des caractéristiques supplémentaires.

Le Tableau 7-6 montre les vingt-cinq items d'instruction primitifs de RIDES, le comportement de chaque item dans les trois modes et les caractéristiques supplémentaires que doit définir l'auteur. En analysant cette table, nous pouvons observer que, pour la définition des items, le contrôle pédagogique doit requérir l'information suivante de la simulation :

- La composition de la simulation : la liste des scènes et la liste des objets.
- Une liste d'attributs pour chaque objet et pour chaque scène.
- Une liste d'événements que le contrôle pédagogique peut observer et générer.

Type d'item	Item	Caractéristiques additionnelles définies par l'auteur	Comportement en mode démonstration	Comportement en mode pratique et en mode évaluation
Instruction	Set Control*	La valeur d'un attribut d'un objet du modèle (Par démonstration).	Donne à l'attribut la valeur fixée par l'auteur.	Demande à l'apprenant de donner à l'attribut la valeur fixée par l'auteur.
	Find Object*	Une référence à un objet de la simulation (Par démonstration).	Signale l'objet dans l'interface.	Demande à l'apprenant de signaler l'objet.
	Read Indicator*	Une référence à un objet et la valeur d'un attribut.	Signale la valeur de l'attribut dans l'interface.	Demande à l'apprenant de signaler la valeur de l'attribut de l'objet.
	Highlight Object*	Une référence à un objet de la simulation.	Met en évidence l'objet dans l'interface.	Met en évidence l'objet dans l'interface.
Présentation	Keyboard Question	Une réponse textuelle.	Montre le texte défini pour le mode démonstration.	Pose la question à l'apprenant et attend une réponse.
	Keypad Question	Une réponse numérique, avec un pourcentage de tolérance.	Montre le texte défini pour le mode démonstration.	Pose la question à l'apprenant, montre un clavier numérique et attend une réponse.
	Menu Question	Une liste d'options, les réponses correctes possibles et le nombre de réponses que doit donner l'apprenant	Montre le texte défini pour le mode démonstration.	Pose la question à l'apprenant, montre un menu avec une liste d'options et attend que l'apprenant sélectionne une réponse.
	Text Presentation		Montre le texte défini pour le mode démonstration.	Montre le texte défini pour le mode pratique ou test.
	Topic Presentation	Une unité de connaissance et un thème.	Montre le texte du thème de l'unité de connaissance.	Montre le texte du thème de l'unité de connaissance.
	Clear Text Window		Efface le texte de la fenêtre d'instruction.	Efface le texte de la fenêtre d'instruction.
	Move Text Window	Coordonnées de la fenêtre d'instruction.	Déplace la fenêtre d'instruction à la position établie par l'auteur.	Déplace la fenêtre d'instruction à la position établie par l'auteur.
	Close Text Window		Ferme la fenêtre d'instruction.	Ferme la fenêtre d'instruction.
	Wait for Student	Un temps limite	Attend que l'apprenant appuie sur le bouton continuer ou la limite de temps.	Attend que l'apprenant appuie sur le bouton continuer ou la limite de temps.

Type d'item	Item	Caractéristiques additionnelles définies par l'auteur	Comportement en mode démonstration	Comportement en mode pratique et en mode évaluation
Action	Install Configuration*	Le nom d'une configuration.	Met la simulation dans la configuration établie par l'auteur.	Met la simulation dans la configuration établie par l'auteur.
	Show Object *	Une référence à un objet de la simulation.	Change l'attribut de visibilité de l'objet à TRUE.	Change l'attribut de visibilité de l'objet à TRUE.
	Hide Object *	Une référence à un objet de la simulation.	Change l'attribut de visibilité de l'objet à FALSE.	Change l'attribut de visibilité de l'objet à FALSE.
	Open Scene *	Le nom d'une scène de la simulation.	Ouvre la scène spécifiée par l'auteur.	Ouvre la scène spécifiée par l'auteur.
	Close Scene *	Le nom d'une scène de la simulation.	Ferme la scène spécifiée par l'auteur.	Ferme la scène spécifiée par l'auteur.
	Free Play *	Un temps limite.	Permet à l'apprenant d'agir sur la simulation pendant le temps fixé par l'auteur ou jusqu'à que l'apprenant appuie sur le bouton Done.	Permet à l'apprenant d'agir sur la simulation pendant le temps fixé par l'auteur ou jusqu'à que l'apprenant appuie sur le bouton Done.
	Pause	Un temps limite.	Arrête l'exécution de l'instruction le temps fixé par l'auteur.	Arrête l'exécution de l'instruction le temps fixé par l'auteur.
	Start Simulator *		Démarre l'exécution de la simulation.	Démarre l'exécution de la simulation.
	Stop Simulator *		Arrête l'exécution de la simulation.	Arrête l'exécution de la simulation.
	Set Attribute *	Référence à un attribut d'un objet de la simulation et une valeur	Donne à l'attribut de l'objet la valeur spécifiée par l'auteur.	Donne à l'attribut de l'objet la valeur spécifiée par l'auteur.
	Execute Event *	Le nom d'un événement défini dans la simulation.	Exécute l'événement.	Exécute l'événement.
	Wait for Event*	Une condition sur les objets de la simulation.	Attend que la simulation atteigne la condition fixée par l'auteur.	Attend que la simulation atteigne la condition fixée par l'auteur.

* Items qui au moment de l'exécution demandent des informations ou des services à la simulation.

Tableau 7-6: Items d'instruction primitifs de RIDES

Les noms fournis par la simulation sont les noms donnés par l'auteur aux composants et aux attributs de la simulation; ils sont significatifs dans le contexte de la simulation.

Pour les exercices générés automatiquement, l'auteur doit définir d'abord une configuration de départ de la simulation, c'est-à-dire un état initial. La définition des configurations est faite par démonstration : l'auteur donne un nom à la configuration, met ensuite la simulation dans l'état souhaité et finalement demande à l'éditeur de configurations d'enregistrer l'état. Puis, l'auteur définit l'exercice.

Nous n'analysons pas ici la création libre d'exercices de type leçon, car les besoins d'information de la simulation sont couverts par les besoins évoqués pour la définition des items de base. Néanmoins, la définition de configurations demande à la simulation plusieurs services:

- Activer la simulation depuis le contrôle pédagogique pour permettre à l'auteur de faire sa démonstration.
- Capturer l'état de la simulation depuis le contrôle pédagogique.
- Désactiver la simulation depuis le contrôle pédagogique.

7.2.4.2 Exécution du contrôle pédagogique

Dans le Tableau 7-6, les items marqués d'un * demandent des informations ou des services à la simulation au moment de l'exécution. Les besoins identifiables pour ces items sont les suivants:

- Changer la valeur d'un attribut d'un objet depuis le contrôle pédagogique (items set control, set attribute et install configuration).
- Capturer la valeur d'un attribut d'un objet depuis le contrôle pédagogique (items set control en modes pratique et évaluation).
- Sélectionner un objet dans l'interface depuis le contrôle pédagogique (items Find Object et Read Indicator).
- Notifier que l'apprenant choisit un objet dans l'interface de la simulation (items Find Object et Read Indicator en mode pratique et évaluation).
- Mettre en évidence un objet depuis le contrôle pédagogique (item highlight).

- Montrer et cacher un objet depuis le contrôle pédagogique (items show object et hide object).
- Ouvrir et fermer une scène depuis le contrôle pédagogique (items open scene et close scene).
- Activer/Désactiver la simulation depuis le contrôle pédagogique (items Free play, Start simulator, et Stop simulator).
- Démarrer/Arrêter la simulation depuis le contrôle pédagogique (items Start simulator, et Stop simulator).
- Générer un événement depuis le contrôle pédagogique (item execute event).
- Etre notifié qu'un événement arrive dans la simulation (item wait event).

7.2.4.3 Synthèse des services

Pour pouvoir créer et exécuter un contrôle pédagogique similaire au contrôle proposé par RIDES, une simulation doit fournir les services suivants :

Service	Phase d'utilisation	Type de service
1. Composition de la simulation en termes de scènes et d'objets.	Création	Réference
2. Une liste des attributs des objets que le contrôle pédagogique peut observer et changer.	Création	Réference
3. Une liste d'événements que le contrôle pédagogique peut observer et exécuter.	Création	Réference
4. Activer/Désactiver la simulation depuis le contrôle pédagogique.	Création Exécution	Scriptabilité
5. Récupérer l'état de la simulation, c'est-à-dire récupérer les valeurs des attributs des objets de la simulation.	Création Exécution	Inspectabilité
6. Changer l'état de la simulation, c'est-à-dire changer les valeurs des attributs des objets de la simulation.	Exécution	Scriptabilité
7. Récupérer la valeur d'un attribut d'un objet de la simulation.	Création Exécution	Inspectabilité
8. Changer la valeur d'un attribut d'un objet de la simulation.	Exécution	Scriptabilité
9. Signaler un objet de la simulation dans l'interface.	Création Exécution	Scriptabilité
10. Notifier si l'apprenant signale un objet dans l'interface de la simulation.	Exécution	Observabilité
11. Mettre en évidence un objet dans l'interface.	Exécution	Scriptabilité
12. Montrer/cacher un objet dans l'interface.	Exécution	Scriptabilité
13. Démarrer/Arrêter la simulation.	Exécution	Scriptabilité
14. Générer un événement.	Exécution	Scriptabilité
15. Notifier l'arrivée d'un événement.	Exécution	Observabilité

Tableau 7-7: Services requis par le contrôle pédagogique de RIDES

7.2.5 Autres systèmes

D'autres systèmes proposent aussi un composant pour le contrôle pédagogique. Néanmoins, leur documentation s'est avérée insuffisante pour nous permettre d'établir précisément les services fournis par la simulation au contrôle pédagogique.

Par exemple, SimBest a un éditeur d'instruction, dédié à la définition d'un guide pédagogique pour l'apprenant, mais ce n'est qu'un outil auteur [MAR 96] [MAR 97]. Cet éditeur ne peut pas se connecter à la simulation.

Dans le projet Interact, une simulation est entourée de support pédagogique avec la possibilité de l'inclure dans une page HTML. L'unique fonctionnalité fournie à l'auteur pour contrôler la simulation est l'option d'enregistrer et d'exécuter une interaction particulière avec la simulation. Ceci nécessite que la simulation soit capable d'informer sur son état et que l'outil de contrôle puisse changer l'état de la simulation.

7.3 Une classification des services de communication

Après cet inventaire des services fournis dans chaque système étudié, cette section se propose d'établir une classification des services qu'une simulation devrait fournir à une application de contrôle pédagogique.

En premier lieu, nous nous proposons de synthétiser dans le Tableau 7-8 l'ensemble de tous les services mis en évidence, tous systèmes confondus. Pour ce faire, nous avons considéré que deux services provenant de systèmes différents et remplissant le même objectif représentaient le même service. Par exemple, les services 5 d'OASIS, 9 de SimQuest, 7 de RIDES et 2 de SAM sont réunis en un même service, à savoir le service 7, car ils ont tous pour objectif le changement des valeurs d'une liste de variables de la simulation ; cette liste pouvant être réduite au cas particulier d'une liste à un seul élément.

En second lieu, afin de compléter notre classification, nous nous intéressons aux caractéristiques communes des services.

Un premier critère de classification peut être dégagé en constatant que la plupart des services opèrent sur la simulation à un niveau sémantique, c'est-à-dire qu'ils font

référence à des variables, à des événements et à des éléments de description de la simulation. De ce fait, le premier critère de classification que nous allons utiliser est le type d'éléments de simulation auquel fait référence un service. Quatre types d'information sont à considérer lors des échanges entre simulation et contrôle pédagogique :

- Données : variables du modèle de la simulation qui sont publiques pour les applications de contrôle pédagogique. Elles ont un nom et un type.
- Événements : événements ou actions sur la simulation. Il peut s'agir d'événements ayant un sens ou de simples événements d'interface. Ils ont un nom.
- États de la simulation : descriptions complètes de la simulation. Leur caractéristique principale est qu'ils peuvent servir à restaurer une situation précise de la simulation (modèle et interface). Les états dépendent de la simulation et doivent être traités comme un tout.
- Composants de la simulation : une description de la composition de la simulation en termes d'objets ou éléments. Elle peut être décrite comme une hiérarchie d'éléments qui représente la relation *composé par*. Lorsque la hiérarchie n'a qu'un niveau, elle correspond à une liste d'éléments. Chaque élément a un nom qui a un sens dans la simulation.

Lorsqu'il s'agit de surveiller la simulation, il faut considérer un deuxième critère de classification, le mode de surveillance que permet d'appliquer le service :

- Surveillance par scrutation périodique : l'application de contrôle pédagogique scrute périodiquement les données et les événements à surveiller. Ce type de surveillance est basé sur les services d'inspectabilité.
- Surveillance par notification : la simulation notifie à l'application de contrôle pédagogique quand les données changent ou quand les événements surviennent. Ce type de surveillance est basé sur les services d'observabilité.

Le Tableau 7-8 montre la classification de chaque service.

	Service	Type de service	Type d'information	Type de surveillance
1.	Fournir une liste des variables de la simulation qui sont observables et modifiables.	Référence	Données	
2.	Fournir la composition du système (éléments, parties, sous–parties, etc).	Référence	Evénements	
3.	Fournir une liste d'événements que le contrôle pédagogique peut observer et générer.	Référence	Composants	
4.	Récupérer l'état de la simulation, faire une photo de la simulation du point de vue modèle et interface.	Inspectabilité	Etats	
5.	Récupérer les valeurs des variables de la simulation qui appartiennent à la liste fournie par le service 1.	Inspectabilité	Données	Par scrutation
6.	Initialiser la surveillance des variables de la simulation qui appartiennent à la liste fournie par le service 1.	Observabilité	Données	Par notification
7.	Stopper la surveillance d'une variable de la simulation qui appartient à la liste fournie par le service 1.	Observabilité	Données	Par notification
8.	Notification de la modification des variables de la simulation qui appartiennent à la liste fournie par le service 1.	Observabilité	Données	Par notification
9.	Notification d'actions de l'élève sur l'interface de la simulation.	Observabilité	Evénéments	Par notification
10.	Notification des événements qui appartiennent à la liste fournie par le service 3	Observabilité	Evénements	Par notification
11.	Notifier si l'apprenant sélectionne un objet dans l'interface de la simulation.	Observabilité	Composants	Par notification
12.	Activer la simulation.	Scriptabilité	Evénements	
13.	Désactiver la simulation.	Scriptabilité	Evénements	
14.	Démarrer/Arrêter la simulation.	Scriptabilité	Evénements	
15.	Changer les valeurs des variables de la simulation qui appartiennent à la liste fournie par le service 1.	Scriptabilité	Données	
16.	Changer la visibilité dans l'interface des variables de la simulation qui appartiennent à la liste fournie par le service 1.	Scriptabilité	Données	
17.	Changer la modifiabilité des variables de la simulation qui appartiennent à la liste fournie par le service 1.	Scriptabilité	Données	
18.	Changer l'état de la simulation, remettre la simulation dans l'état correspondant à une photo enregistrée.	Scriptabilité	Etat	
19.	Produire un des événements qui appartient à la liste fournie par le service 3.	Scriptabilité	Evénements	
20.	Signaler un objet de la simulation dans l'interface.	Scriptabilité	Composants	
21.	Sélectionner/déselectionner (Highlight /Unhighlight) un objet dans l'interface.	Scriptabilité	Composants	
22.	Montrer/cacher un objet dans l'interface.	Scriptabilité	Composants	

Tableau 7-8: Synthèse des services de communication

7.4 Spécification d'une typologie des services de communication

Afin de fournir les bases à la proposition d'architecture du chapitre 8, nous précisons ici les notions dégagées précédemment. Nous spécifions les différents services de la simulation en fonction du type d'information mise à contribution. Nous ajoutons deux services de connexion, pour permettre au contrôle pédagogique d'établir et de finir la communication avec une simulation.

7.4.1 Services de connexion

init_connection

Établit une connexion entre la simulation et l'application de contrôle pédagogique. Ce service permet à la simulation de savoir quand elle est en train d'être observée/contrôlée par une autre application. Retourne une liste avec les noms des services fournis par la simulation.

close_connection

Termine une connexion entre la simulation et l'application de contrôle pédagogique.

7.4.2 Services sur les données

get_variables_list

Retourne une liste des descriptions des variables que l'application de contrôle pédagogique peut observer ou modifier. Pour chaque variable, fournit son nom, son type et une description de la signification de la variable dans le contexte de la simulation.

get_variables_values

Retourne une liste avec les valeurs des variables disponibles dans la simulation. Pour chaque variable, retourne son nom, son type et sa valeur.

get_list_variables_values

Retourne les valeurs d'une liste de variables fournie par l'application du contrôle pédagogique. Pour chaque variable, retourne son nom, son type et sa valeur.

set_variables_values

Change les valeurs des variables disponibles dans la simulation par les valeurs fournies par l'application de contrôle pédagogique. Pour chaque variable, l'application de contrôle pédagogique passe son nom, son type et sa nouvelle valeur.

set_list_variables_values

Change les valeurs d'une liste de variables par les valeurs fournies par l'application de contrôle pédagogique. Pour chaque variable, l'application de contrôle pédagogique passe son nom, son type et sa nouvelle valeur.

set_variable_visibility

Rend visible une variable dans l'interface de la simulation. L'application de contrôle pédagogique passe le nom de la variable et une valeur pour la visibilité (true si la variable doit être visible et false sinon).

set_variable_modifiability

Rend modifiable une variable par l'apprenant. L'application du contrôle pédagogique passe le nom de la variable et une valeur pour la modifiabilité (true si la variable peut être modifiée par l'apprenant et false sinon).

notify_variable

Envoie un message de la simulation à l'application de contrôle pédagogique quand une variable surveillée change. La simulation passe le nom de la variable, son type et la nouvelle valeur. Quand une simulation offre ce service, elle doit proposer aussi les services: init_monitoring_variable et end_monitoring_variable.

init_monitoring_variable

Initialise la surveillance d'une variable. L'application de contrôle pédagogique passe le nom de la variable. A partir du moment où ce service est invoqué par l'application de contrôle pédagogique, la simulation doit envoyer le message notify_variable chaque fois que la variable change.

stop_monitoring_variable

Termine la surveillance d'une variable. L'application de contrôle pédagogique passe le nom de la variable.

7.4.3 Services sur les événements

get_actions_list

Retourne une liste des actions que l'application de contrôle pédagogique peut observer ou exécuter. Pour chaque action, fournit son nom, son type (sémantique ou interface), une description de la signification de la variable dans le contexte de la simulation et si l'action est exécutable depuis l'application de contrôle pédagogique. Toute simulation doit traiter (observation et exécution) les actions suivantes :

- activate_simulation : active la simulation pour l'apprenant.
- deactivate_simulation : inactive la simulation pour l'apprenant. L'apprenant ne peut plus agir sur la simulation.
- init_simulation : démarre la simulation.
- stop_simulation : arrête la simulation.
- pause_simulation : met la simulation en pause.
- resume_simulation : remet la simulation en marche.

execute_action

Exécute une action dans la simulation. L'application de contrôle pédagogique passe le nom de l'action. L'action doit être une action exécutable depuis le contrôle pédagogique.

notify_action

Envoie un message à l'application de contrôle pédagogique quand une action se produit. La simulation passe le nom de l'action. Quand une simulation offre ce service, elle doit aussi disposer des services: init_monitoring_action et end_monitoring_action.

init_monitoring_action

Initialise la surveillance d'une action. L'application de contrôle pédagogique passe le nom de l'action. A partir du moment où ce service est invoqué par l'application de contrôle pédagogique, la simulation doit envoyer le message notify_action chaque fois que l'action se produit.

stop_monitoring_action

Termine la surveillance d'une action. L'application de contrôle pédagogique passe le nom de l'action.

7.4.4 Services sur la hiérarchie d'éléments

get_components_list

Retourne une liste des composants de la simulation. Pour chaque composant, la simulation retourne un nom et une liste des sous-composants du composant.

point_element

Signale un élément d'interface. L'application de contrôle pédagogique passe le nom de l'élément à signaler.

set_element_visibility

Rend visible un élément d'interface. L'application du contrôle pédagogique passe le nom de l'élément et une valeur pour la visibilité (true si l'élément doit être visible et false sinon).

set_element_highligthing

Établit s'il faut faire ressortir un élément d'interface. L'application du contrôle pédagogique passe le nom de l'élément et une valeur (true si l'élément doit être mis en évidence et false sinon).

7.4.5 Services sur les états

get_simulation_state

Retourne un état de la simulation.

set_simulation_state

Met la simulation dans l'état fourni par l'application de contrôle pédagogique.

7.5 Résumé

Nous avons étudié dans ce chapitre la façon dont les systèmes de développement de simulations pédagogiques interconnectent le composant pédagogique avec le reste de la simulation. De cette étude nous avons déduit une liste des services de

communication nécessaires pour pouvoir séparer la simulation et une application de contrôle pédagogique et nous proposons une typologie de ces services :

Types d'information	• Données • Evénements • Etats • Composition
Services sur les données	• get_variables_list • get_variables_values • get_list_variables_values • set_variables_values • set_list_variables_values • set_variable_visibility • set_variable_modifiability • notify_variable • init_monitoring_variable • stop_monitoring_variable
Services sur les événements	• get_actions_list • execute_action • notify_action • init_monitoring_action • stop_monitoring_action
Services sur la hiérarchie d'éléments	• get_components_list • point_element • set_element_visibility • set_element_highligthing
Services sur les états	• get_simulation_state • set_simulation_state

8 ARGOS : ARCHITECTURE GENERALE POUR L'OBSERVATION ET LE CONTROLE PEDAGOGIQUE DE SIMULATIONS

Une simulation peut être réalisée directement avec un langage de programmation comme C++ ou Java, avec des outils comme Rapid ou Labview, ou avec un système auteur comme Authorware ou Toolbook. Pour les simulations pédagogiques, divers systèmes de création ont été présentés dans la première partie. Du fait de cette diversité, les simulations existantes forment un ensemble hétérogène qui n'obéit à aucun standard. Dans de telles conditions, c'est à l'auteur d'un contrôle pédagogique qu'incombe la tâche de tenir compte des détails de réalisation de la simulation qu'il souhaite utiliser.

Afin d'éviter à l'auteur d'entrer dans de tels détails, des systèmes comme Rides, SimQuest et OASIS proposent un cadre de plus haut niveau pour la conception des contrôles pédagogiques. Mais ces systèmes sont fermés, c'est-à-dire qu'ils imposent une contrainte forte : les simulations doivent être conçues par le même système que le contrôle pédagogique.

On voit donc le besoin d'un système qui soit à la fois capable de proposer à un auteur un cadre de conception de haut niveau, tout en restant ouvert pour permettre l'utilisation de simulations d'origines diverses.

C'est à cette problématique qu'entend répondre ce chapitre en proposant une architecture générale pour le couplage entre contrôle pédagogique et simulation. Cette architecture doit pouvoir coupler n'importe quelle simulation avec n'importe quel contrôle pédagogique, et ce, avec un minimum de changements si les applications existent déjà, ou avec un minimum de contraintes si les applications doivent être développées.

La notion centrale sur laquelle s'appuie cette architecture est celle de réutilisation. D'une part, il faut permettre de réutiliser des simulations et des applications de contrôle pédagogique déjà existantes pour produire de nouvelles simulations pédagogiques. Et d'autre part, il faut favoriser la réalisation d'applications réutilisables.

Pour ce faire, nous proposons ARGOS (**Ar**chitecture **G**énérale pour l'**O**bservation et le contrôle pédagogique de **S**imulations), qui se focalise sur la définition de l'architecture logicielle du système à mettre en œuvre, et plus particulièrement sur la définition de l'interaction entre ses composants logiciels, à savoir la simulation et le contrôle pédagogique. L'intérêt de cette approche est de permettre un haut niveau d'abstraction qui évite d'entrer dans les détails d'implémentation.

L'étude présentée dans le chapitre 7 a constitué une première étape vers la conception d'une telle architecture. En effet, la table 7-8 présente une classification des services que doit fournir une simulation à un contrôle pédagogique. C'est autour de ces services que l'architecture ARGOS va s'organiser.

Dans ce chapitre, nous commençons par une présentation rapide du langage de description d'architectures, Unicon. Nous décrivons ensuite ARGOS, en nous servant de Unicon. Nous présentons enfin un exemple d'application d'ARGOS, au prototype SWIPS qui permet l'observation et le contrôle, à distance, de simulations.

8.1 La description graphique d'architectures en Unicon

L'architecture d'un système informatique décrit sa structure: ses composants et leurs relations, les patrons de composition et les contraintes de la composition [SHA 96].

UNICON est un langage de description d'architectures (ADL) proposé par la *School of Computer Science* de l'université Carnegie Mellon [SHA 96] [SHA 95] [ZEL 96].

Pour Unicon, les entités de base d'une architecture sont les composants et les connecteurs. Les composants sont des entités chargées des calculs et des états du système. Les connecteurs fournissent des règles pour relier des composants.

Un composant a une interface qui définit ses propriétés : type du composant (serveur, stockage de données, processus, filtre, etc.), sa fonctionnalité, ses caractéristiques d'exécution et sa signature. Les entités visibles d'un composant sont les acteurs (*players*). Un composant peut être primitif (c'est-à-dire un module de code) ou composé.

Un connecteur a un protocole qui définit ses propriétés : type du connecteur (*remote procedure call*, *pipeline*, événement, etc.), règles sur les interfaces qu'il supporte, caractéristiques d'interaction, etc. Les entités visibles d'un connecteur sont les rôles (*roles*). Un connecteur peut être primitif (fourni par le langage de programmation) ou composé.

UNICON comporte un langage textuel et un langage graphique. C'est ce dernier que nous allons utiliser ici comme moyen d'expression. Décrivons-le brièvement, à l'aide d'un exemple.

8.1.1 Un exemple

Soit un système formé par trois composants A, B et C et le fichier Data. Le composant A lit le fichier, fait des calculs en utilisant les procédures B1 et B2 du composant B et stocke les résultats des calculs en utilisant le service de stockage S1 du composant C. La Figure 8-1 montre graphiquement le système.

Nous supposons ici que A, B, et C sont des composants primitifs, c'est-à-dire qu'ils correspondent chacun à un fichier de code. Ils sont reliés par les connecteurs d'invocation de procédures (▷) et le connecteur de lecture de fichier (▷).

Les acteurs du composant A sont :

- RFile : acteur de type *ReadFile (lecture de fichier)*.
- B1Call : acteur de type *RoutineCall* (invocation de procédure), pour l'invocation du service B1 du composant B.

- B2Call : acteur de type *RoutineCall* (invocation de procédure), pour l'invocation du service B2 du composant B.
- S1Call : acteur du type *RoutineCall* (invocation de procédure), pour l'invocation du service S1 du composant C.

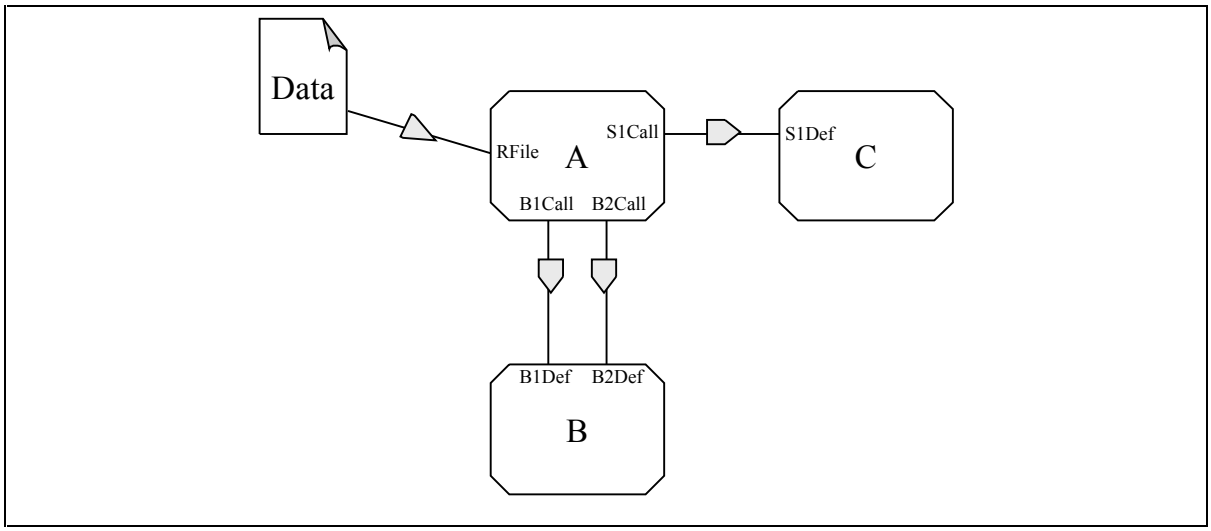


Figure 8-1: Description graphique de l'exemple

UNICON propose un ensemble de types de composants, et des acteurs pour chaque type. Les types proposés sont *module, computation, ShareData, SeqFile, Filter, Process* et *General.* Parmi les acteurs, on trouve: *RoutineDef, RoutineCaller, ReadFile, WriteFile, StreamIn, StreamOut, RPCDef, RPCCall, GlobalDataDef* et *GlobalDataUser*. Un type spécial d'acteur est le *PLBundle* (représenté graphiquement comme ⊂⊂⊂⊂⊃) qui sert à définir des collections cohérentes d'acteurs pour les procédures et les données.

Le connecteur d'invocation de procédures est fourni par UNICON. Il a un rôle *definer* pour la connexion au composant qui définit la procédure, et un rôle *caller* pour la connexion au composant qui invoque la procédure.

Les types de connecteurs proposés par UNICON sont *Pipe, FileIO, ProcedureCall, DataAccess, RemoteProcCall* et *PLBundler*. Pour chaque type de connecteur, UNICON définit ses rôles, ainsi que les acteurs que chaque rôle accepte.

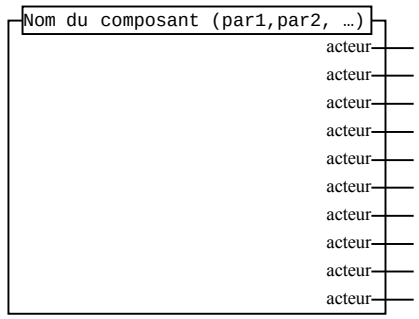
8.1.2 Les extensions proposées

Pour simplifier l'expression de notre architecture ARGOS, nous proposons trois extensions au langage graphique d'UNICON.

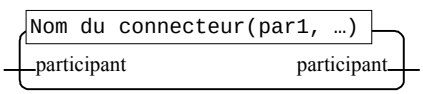
Une première extension concerne les types d'acteurs. UNICON fournit, pour l'appel à distance de procédures, des acteurs basés uniquement sur la technique RPC de UNIX. Pour généraliser le concept à d'autres techniques, par exemple CORBA ou JavaRMI, nous introduisons les types d'acteurs *RPCGDef* et *RPCGCall.*

La deuxième extension porte sur le groupement des acteurs. UNICON ne fournit des *PLbundles* que pour les acteurs de types définition et invocation de procédures, et de types définition et utilisation de données globales. Nous étendons les acteurs du type *PLbundle* pour couvrir aussi des acteurs de types définition et invocation de procédures distantes génériques (*RPCGDef, RPCGCall*). Pour cela, nous définissons un nouveau rôle *RPCGBundle* et un nouveau type de connecteur *RPCGBundler.* Cette extension nous permettra de regrouper par types les services d'une simulation.

La troisième extension donne la possibilité de paramétrer les composants de type processus et les connecteurs de type *RPCGBundler*. Cette extension est nécessaire car les composants que nous voulons décrire sont dépendants des protocoles de communication des composants. La notation graphique de cette extension est présentée dans la Figure 8-2.



Composants de type processus paramétrables



Connecteur de type collection d'invocation de procédures à distance (*RPCGBundler*)

Figure 8-2: Notations graphiques pour les extensions de paramétrage

8.2 Description d'ARGOS

ARGOS a pour objectif principal de faciliter la construction d'une simulation pédagogique par couplage de deux composants existants : une application de contrôle pédagogique et une simulation. Nous proposons d'ajouter quelques composants afin de faciliter la réutilisation.

Les caractéristiques essentielles que nous souhaitions offrir à travers cette architecture sont :

1. Une souplesse d'intégration d'applications nouvelles ou déjà existantes.
2. Une souplesse d'adaptation aux différents protocoles de communication entre applications.
3. Un haut degré de réutilisation des implémentations de l'architecture.

La présentation d'ARGOS suit une approche descendante: nous montrons d'abord un diagramme général avant de détailler chaque partie. L'annexe 2 contient la description d'ARGOS dans le langage textuel proposé par UNICON.

8.2.1 Diagramme général

ARGOS est basé sur une architecture Client-Serveur (cf. Figure 8-3). Le serveur englobe la simulation ; les services fournis sont ceux spécifiés dans le chapitre 5 (cf. §5.4.3). Le client est le contrôle pédagogique (*CPClient*).

La relation client-serveur a deux paramètres : le premier est le protocole que le contrôle pédagogique utilise pour accéder aux services de la simulation (*CPProtocol*) ; le deuxième paramètre est le protocole qu'utilise la simulation pour offrir ses services (*SimProtocol*). Rappelons que les deux applications (simulation et contrôle pédagogique) étant indépendantes, ces deux protocoles sont a priori différents.

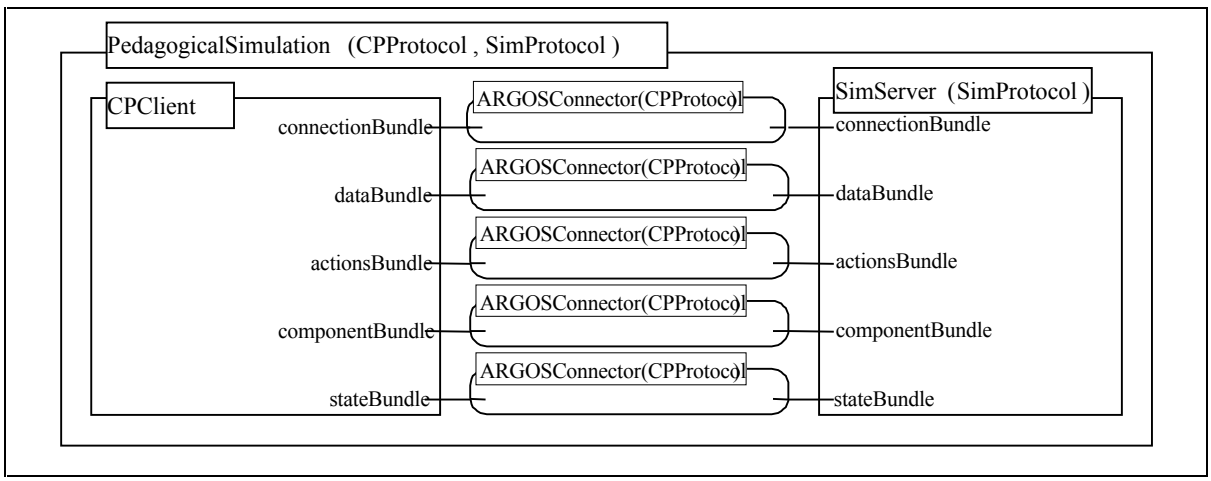


Figure 8-3: Diagramme général d'ARGOS

Nous considérons le contrôle pédagogique, *CPClient*, comme un composant primitif, et le serveur de simulation, *SimServer*, comme une composition (cf. annexe 2).

Nous présentons maintenant la description de chaque composant et du connecteur.

8.2.2 CPClient: le contrôle pédagogique

Le contrôle pédagogique est un processus qui doit surveiller et contrôler une simulation. Notre architecture n'est pas concernée par la structure interne de ce contrôle. Il y a toutefois deux contraintes pour le contrôle pédagogique :

- Il ne doit utiliser que les services prévus par l'architecture.
- Il doit être capable de traiter les notifications de changements en provenance de la simulation.

D'autre part, l'architecture a besoin de connaître le protocole de communication qu'utilise le contrôle pédagogique pour ses demandes de services à la simulation. Le nom de ce protocole constitue le paramètre *CPProtocol*. Ce nom doit faire partie de ceux reconnus par l'architecture. Pour un nouveau protocole, il faudra alors compléter les éléments de l'architecture qui implémentent les fonctionnalités requises pour ce protocole.

8.2.3 SimServer: le serveur de simulation

Le composant *SimServer* est chargé de fournir les services demandés à la simulation. Le contrôle pédagogique n'est pas concerné par la façon dont la simulation fournit ces services. Il ne doit pas dépendre des noms et des paramètres des services, ni du protocole utilisé par la simulation.

Nous identifions pour ce composant deux responsabilités :

1. L'adaptation des services offerts par la simulation.
2. La communication avec la simulation.

Pour accomplir ces tâches, le composant a trois sous-composants (cf. Figure 8-4) :

1. *Simulation* : La simulation elle-même.
2. *CommAdapter* : Composant en charge de la communication avec la simulation.

3. *ServiceAdapter* : Composant responsable d'adapter les services fournis par la simulation à ceux requis par le contrôle pédagogique.

Le mécanisme global est le suivant : le *ServiceAdapter* reçoit les demandes du contrôle pédagogique. Il consulte le fichier *MappingData* et construit une demande de service pour la simulation. Le composant *CommAdapter* envoie ce message à la simulation selon le protocole qu'elle utilise.

Détaillons maintenant le travail de chaque composant. Nous commençons par la simulation, qui, rappelons-le, est un composant préexistant.

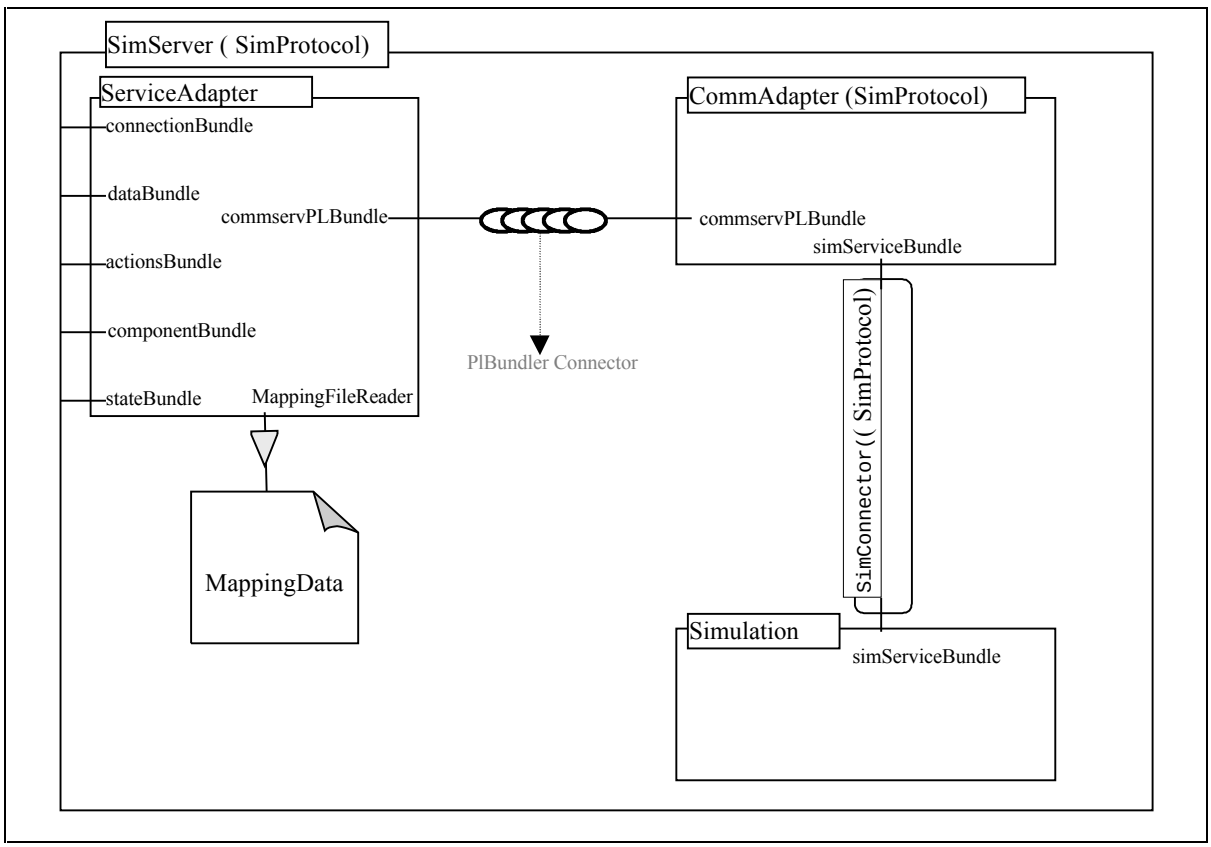


Figure 8-4: Description graphique du composant SimServer

La simulation

La simulation est l'application qui doit être surveillée et contrôlée. Pour être utilisable dans ARGOS, elle doit fournir un ensemble minimum de services :

Pour être inspectable, il faut qu'elle fournisse un service qui retourne la valeur d'une variable de nom donné, ou un service qui retourne une liste de paires (nom_variable, valeur).

Pour être scriptable (ou contrôlée), il faut qu'elle fournisse un service qui permet de changer la valeur d'une variable, ou un service qui, étant donnée une liste de paires (nom_variable, nouvelle_valeur), permet de changer les valeurs de toutes les variables de la liste ; ainsi que des services pour activer et désactiver la simulation.

Ces services doivent être accessibles par un protocole de communication : DDE, OLE, RPC, RMI, etc.

ServiceAdapter : l'adaptateur des services de la simulation

Ce composant fournit au contrôle pédagogique un ensemble bien défini de services, quelle que soit la simulation. Certains services peuvent ne pas être proposés par une simulation donnée et doivent alors être construits à partir de services plus simples. Ainsi, lorsqu'une simulation ne fournit pas le service *init_monitoring_variable*, mais seulement le service *get_variable_value*, le *ServiceAdapter* va se charger de scruter à intervalles réguliers la valeur de la variable et d'informer le contrôle pédagogique en cas de changement.

Si une simulation offre l'ensemble minimum de services évoqués plus haut, le *ServiceAdapter* peut alors fournir au contrôle pédagogique les services ARGOS listés dans le Tableau 8-1.

get_variables_list
get_variables_values
get_list_variables_values
set_variables_values
set_list_variables_values
notify_variable
init_monitoring_variable
stop_monitoring_variable
get_actions_list
execute_action (activer /désactiver seulement)

Tableau 8-1 : Services minimums fournis par le *ServiceAdapter*

Si d'autres services sont souhaités, mais ne sont pas recensés dans le tableau 6.1, c'est à la simulation de fournir des services adaptés.

Le fonctionnement de ce composant nécessite un fichier, *MappingData*, qui contient la liste des services effectivement fournis par la simulation et une information sur la façon de faire correspondre les noms et les paramètres avec les services de la simulation.

Ce composant est indépendant des protocoles utilisés et de la manière dont la simulation offre ses services. C'est un composant réutilisable.

CommAdapter : l'adaptateur de protocole de communication entre applications

Une première barrière pour le couplage d'une simulation et d'un contrôle pédagogique est la variété des mécanismes utilisés pour communiquer entre les applications. Des simulations construites avec Toolbook, Rapid ou LabView, travaillent avec d'anciens protocoles de bas niveau comme DDE (Dynamic Data Exchange). D'autres simulations, développées avec des langages de programmation, utilisent d'autres mécanismes comme CORBA et JavaRMI.

Le composant *CommAdapter* encapsule les fonctions de communication vers la simulation, ce qui permet au contrôle pédagogique de ne pas se préoccuper du protocole utilisé. Il se charge d'envoyer à la simulation les messages selon le protocole qu'elle impose.

Pour être réutilisable, ce composant doit permettre d'utiliser différents protocoles de communication. Nous avons, pour cela, appliqué le patron de conception *Adaptateur* [GAM 95]. La Figure 8-5 montre le schéma de classes du composant.

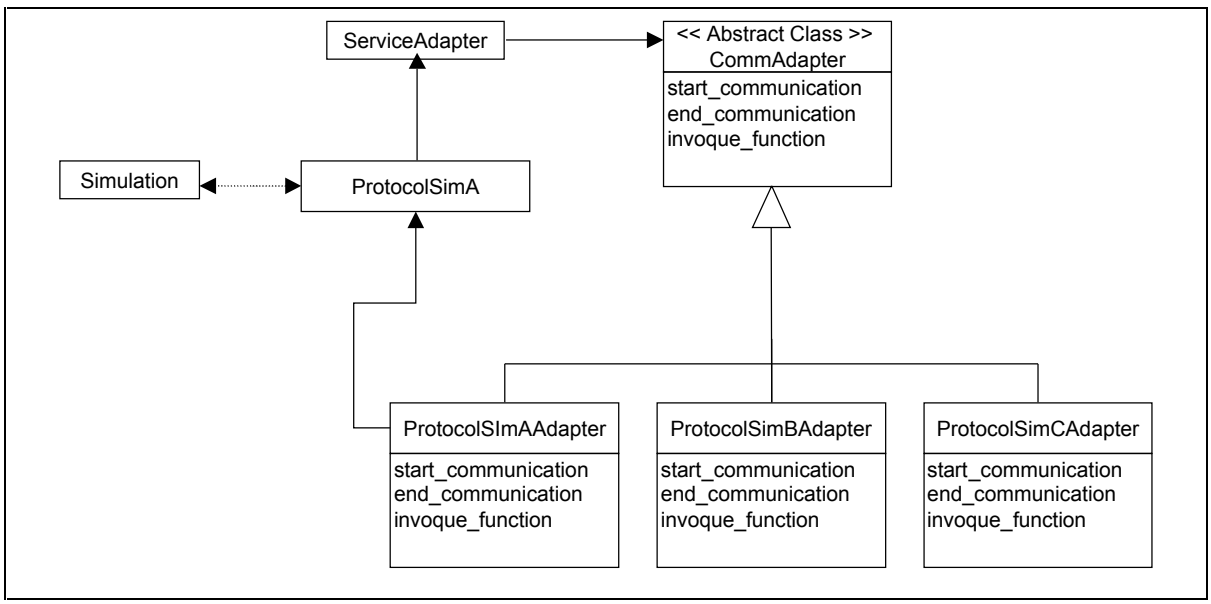


Figure 8-5: Schéma de classes du composant CommAdapter d'ARGOS

Le *ServiceAdapter* demande un service à la simulation en passant par la méthode `invoque_function` du *CommAdapter*. Cette méthode reçoit en paramètre le nom du service demandé ainsi que les paramètres de cette demande.

CommAdapter est une classe abstraite qui définit une interface commune pour tous les protocoles. Pour chaque protocole (par exemple *A*), il faut faire une classe concrète (*ProtocolSimAAdapter)* qui implémente les méthodes de la classe abstraite *CommAdapter*, en utilisant les services d'une classe (*ProtocolSimA*) responsable de la communication selon ce protocole. Quand la simulation envoie une notification de changement au contrôle pédagogique, la classe *ProtocolSimA* délègue la responsabilité au composant *ServiceAdapter*.

Le fait de traiter le *CommAdapter* comme un adaptateur permet au *ServiceAdapter* de communiquer avec la simulation indépendamment du protocole.

8.2.4 Le connecteur ARGOSConnector

Ce connecteur reçoit les demandes du contrôle pédagogique, exprimées dans le protocole utilisé par le contrôle pédagogique, et les transmet au serveur de simulation. Le connecteur s'exécute en fonction du protocole passé en paramètre.

La Figure 8-6 montre le schéma de classes du composant. Pour rendre ce composant réutilisable, nous utilisons de nouveau le patron de conception *Adaptateur* [GAM 95] afin de faciliter l'ajout de nouveaux protocoles.

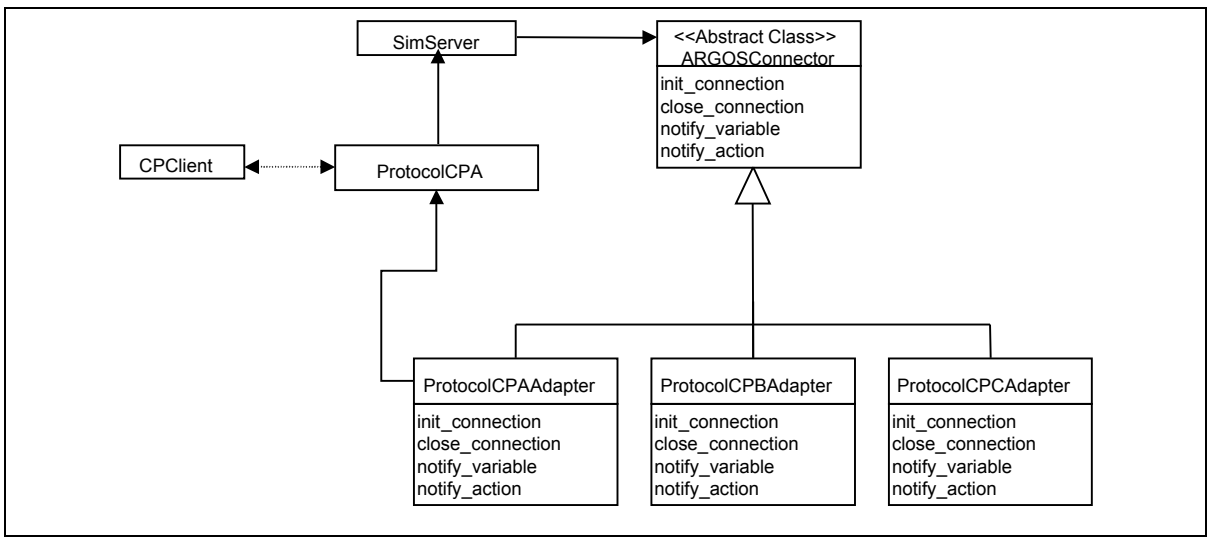


Figure 8-6: Schéma de classes du connecteur ARGOSConnector

Le connecteur est une classe abstraite qui permet d'accéder aux différents protocoles. Chaque protocole doit implémenter le traitement des notifications que le composant *SimServer* peut envoyer au contrôle pédagogique. Pour cela, il utilise des méthodes propres

au protocole, implémentées dans une classe correspondant au protocole (*ProtocolCPA* dans la figure).

Cette classe a la responsabilité de recevoir les demandes envoyées par le contrôle pédagogique à la simulation et de faire les adaptations nécessaires pour déléguer la demande au *SimServer*. Elle se charge également d'utiliser le protocole adéquat pour établir la communication avec le contrôle pédagogique. La mise en œuvre du patron Adaptateur permet au contrôle pédagogique d'accéder aux services du composant *SimServer* indépendamment du protocole.

8.3 SWIPS: un exemple pour l'application d'ARGOS

Nous allons maintenant montrer un exemple où l'application d'ARGOS peut rendre réutilisable un contrôle pédagogique et une simulation. Nous présentons tout d'abord le contexte, puis le prototype que nous avons développé et les besoins de généralisation et réutilisation rendus possibles par notre architecture. Finalement, nous montrons comment l'architecture actuelle peut être modifiée pour la rendre plus réutilisable et générale.

8.3.1 Contexte

De nos jours l'éducation à distance utilisant les nouvelles technologies et les réseaux prend de plus en plus d'importance. Ces technologies ont permis de reproduire à distance la plupart des situations que l'on trouve dans l'enseignement traditionnel:

- Les video-conférences et/ou les documents hypermédias dans le WEB permettent de reproduire des séances dans lesquelles l'enseignant transmet des informations aux élèves.
- Une séance de discussion peut se faire en utilisant des forums ou tout simplement le courrier électronique.
- L'évaluation peut être réalisée avec des outils spécialisés (par exemple pour faire des examens de style QCM).

Les outils pour gérer les cours et le curriculum à distance sont également assez répandus. Néanmoins, il existe encore des situations qu'on ne peut pas reproduire dans l'enseignement à distance: celles où l'interaction enseignant-élève est essentielle, comme les séances de travaux dirigés et de travaux pratiques. Nous nous intéressons

particulièrement à la réalisation des outils qui permettent de reproduire des situations de TD/TP qui s'appuient sur des simulations pédagogiques.

Dans une telle séance, un enseignant et un groupe d'étudiants se rencontrent à travers le réseau, à une date et à une heure fixées à l'avance, pour réaliser un TP/TD avec une simulation pédagogique. Les étudiants font des exercices proposés par la simulation pédagogique. L'enseignant est présent pour suivre le travail des élèves. Il peut donner des indications au groupe ou à un étudiant particulier, guider un étudiant qui est sur une voie sans issue, observer le travail fait par un étudiant, etc.

8.3.2 Le prototype

Dans ce cadre, nous avons développé SWIPS, un prototype en Java pour la réalisation d'une séance virtuelle de TD pour un enseignement sur les réseaux [ARE 98]. L'objectif du prototype a été de fournir à l'enseignant un outil pour surveiller et guider, à distance, le travail des élèves sur une simulation pédagogique.

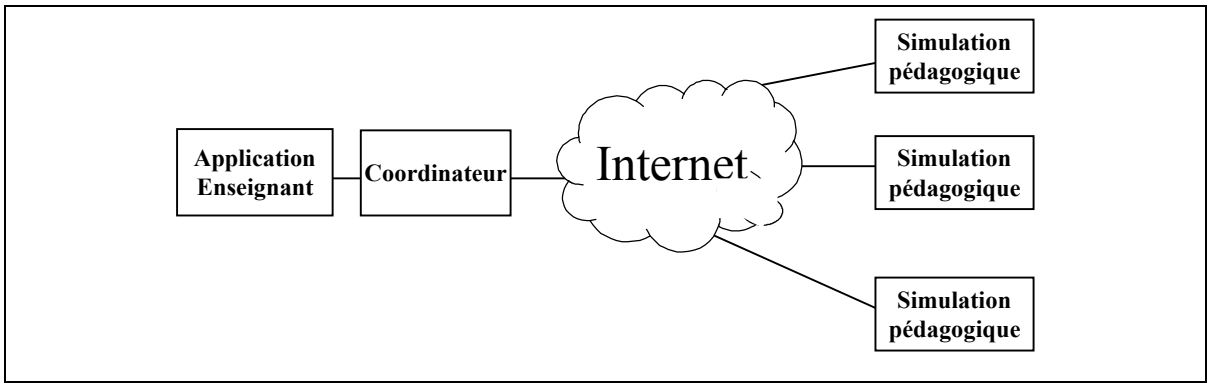


Figure 8-7: La séance de TD virtuel

La Figure 8-7 montre le schéma proposé pour le TD virtuel. Il y a un composant coordinateur qui gère l'information commune du système, et des utilisateurs qui utilisent l'information: l'enseignant et les élèves. Parmi les fonctions du coordinateur, nous pouvons mentionner la gestion des utilisateurs du système, le contrôle d'accès au système, la coordination d'envoi et de réception d'informations entre l'application de l'enseignant et les applications des élèves, et la gestion du stockage des informations partagées par les utilisateurs.

Le déroulement typique d'une séance est le suivant :

- Le coordinateur est actif à tout moment.

- L'enseignant et les étudiants se connectent au système, puis à un cours, et commencent une séance. Le coordinateur donne accès aux utilisateurs inscrits pour le cours. Les étudiants ne peuvent pas démarrer l'exercice sans l'autorisation de l'enseignant.
- Quand l'enseignant voit que tous ses étudiants sont arrivés, il démarre la séance.
- Les étudiants font les exercices; l'enseignant suit le travail des étudiants "en regardant" leurs écrans ou en demandant au coordinateur l'envoi des informations sur l'activité d'un étudiant.

La Figure 8-8 montre l'interface de l'enseignant pour la surveillance à distance de la séance. L'annexe 5 montre en détail l'utilisation du prototype.

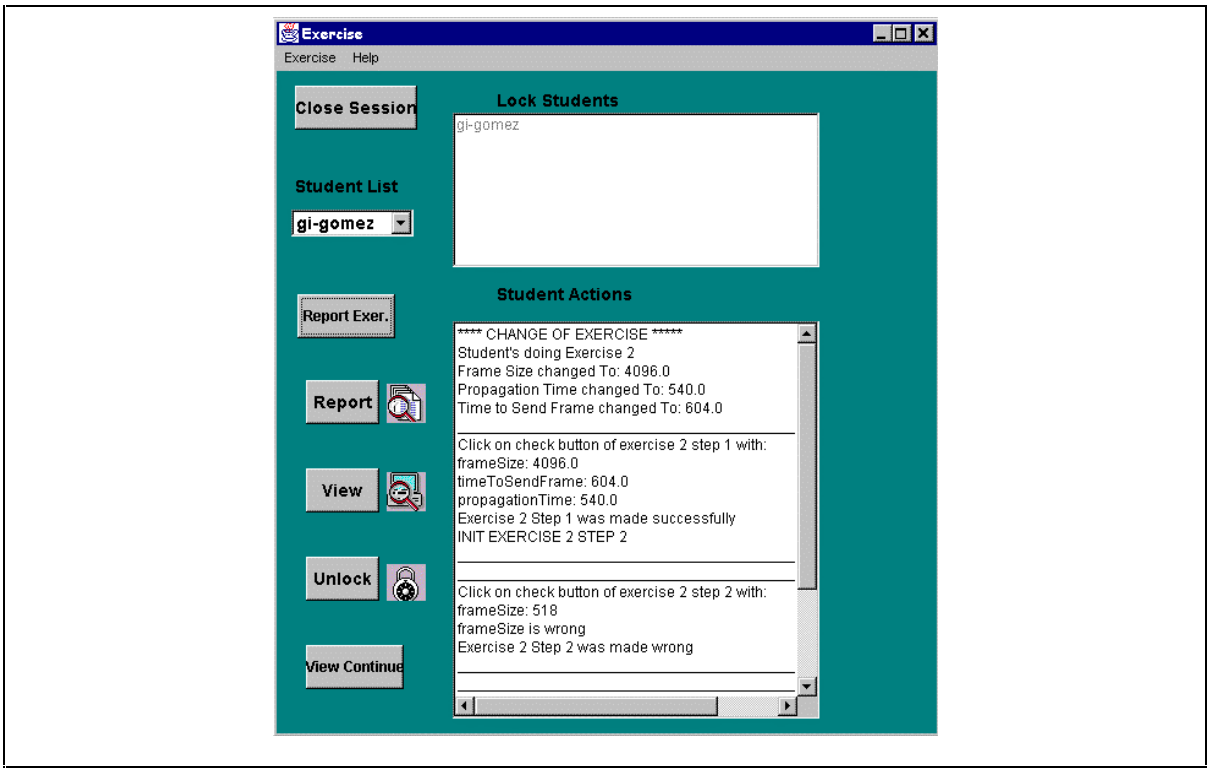


Figure 8-8: Interface de l'enseignant dans le TD à distance

8.3.2.1 La simulation pédagogique

Détaillons les deux composants de la simulation pédagogique : la simulation proprement dite et le contrôle pédagogique.

La simulation

L'application *SimSlide* simule le protocole de fenêtre glissante (Slide Window Protocol) utilisé dans la couche de liaison (Data Link Layer) du modèle OSI de communication [TAN 94]. La simulation a été réalisée avec MARS comme méthodologie de développement. Elle peut être exécutée indépendamment en l'absence de tout contrôle pédagogique.

La Figure 8-9 montre l'interface de la simulation. Quand la simulation est en marche, la machine A envoie des paquets à la machine B. L'élève peut suivre le fonctionnement du protocole de fenêtre glissante et étudier son comportement en changeant les variables. Les variables que l'élève peut changer sont :

- *Noise*: le bruit du canal de communication (0-1.0)
- *Delay*: le délai du canal
- *Capacity*: La capacité du canal à transmettre des données (en Kbytes/seconds)
- *Frame Size*: La taille des paquets de données.
- *Bits enum Window*: La valeur binaire qui est utilisée pour numéroter les paquets à envoyer.
- *dt*: le delta de temps utilisé par la simulation

Le contrôle pédagogique

Nous avons développé un contrôle pédagogique consistant à exécuter des exercices similaires à ceux proposés par OASIS (cf. chapitre 4) avec quelques extensions pour envoyer des informations au coordinateur.

Le contrôle pédagogique permet à l'élève d'utiliser la simulation en mode simulation libre ou de travailler sur une liste d'exercices.

Pour les exercices, nous avons utilisé la notion de scénario proposé par OASIS, en y ajoutant deux caractéristiques:

- La possibilité de masquer des variables au début de l'exercice et au début des étapes.
- La possibilité que le contrôle pédagogique bloque la simulation si la vérification d'une étape (demandée par l'élève) détecte un échec ou si un

contrôle est déclenché. L'élève reste alors bloqué jusqu'à l'arrivée d'un message débloquant émis par le coordinateur.

Pour utiliser la simulation pédagogique dans les TD virtuels, nous avons ajouté aussi à l'application de contrôle pédagogique, la fonctionnalité d'envoyer au coordinateur des informations sur le déroulement des exercices (début de l'exercice, début d'une étape, déclenchement d'un contrôle, etc.).

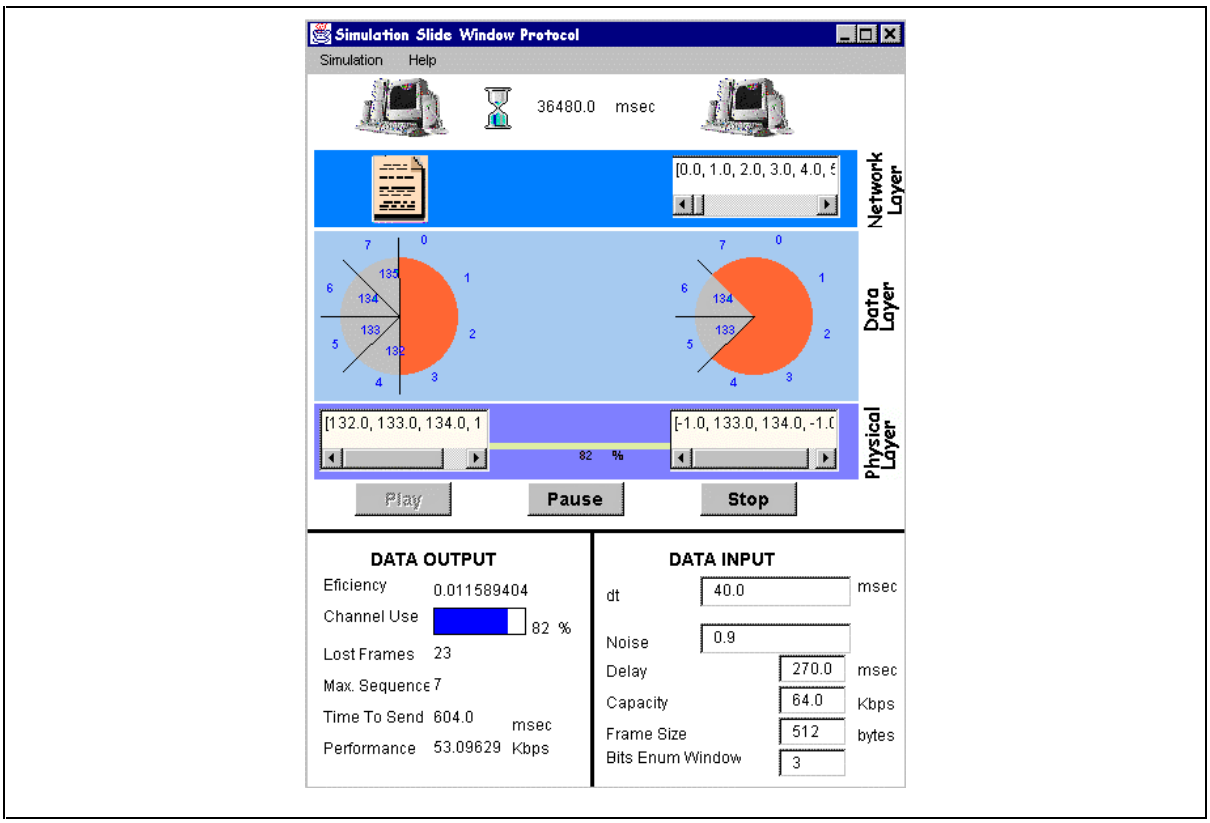


Figure 8-9: *SimSlide* : Simulation d'un protocole de fenêtre glissante

8.3.3 Besoins de réutilisation et de généralisation

Nous voulons rendre réutilisables, indépendamment, la simulation et le contrôle pédagogique que nous avons développés dans le prototype. La simulation doit être réutilisable avec divers contrôles pédagogiques et le contrôle pédagogique doit être réutilisable avec d'autres simulations.

8.3.4 Démarche d'application d'ARGOS

L'application de l'architecture ARGOS nécessite de faire certaines modifications, selon les étapes suivantes que nous détaillerons plus loin.

La première étape vise à satisfaire les contraintes suivantes :

- La simulation doit fournir un ensemble de services minimums et utiliser un protocole de communication.
- Le contrôle pédagogique doit requérir de la simulation uniquement les services prévus par l'architecture et doit utiliser un protocole de communication.

La deuxième étape consiste à définir le fichier *MappingData* pour établir les correspondances entre les services de la simulation et les services d'ARGOS.

La dernière étape consiste à compléter les composants d'ARGOS responsables des communications afin qu'ils prennent en charge les protocoles utilisés : les composants *ARGOSConnector* et *CommAdapter*. Ces extensions ne sont à faire qu'une fois par nouveau protocole.

Détaillons maintenant chacune de ces étapes.

8.3.4.1 Adaptation de la simulation

Commençons par les contraintes qui portent sur la simulation. Dans notre cas, la simulation est implémentée au départ par une classe qui fournit les méthodes suivantes :

- Activer et désactiver la simulation pour l'élève (méthodes *lock*/*unlock*).
- Retourner les valeurs des variables de la simulation (*getproperties*).
- Changer les valeurs des variables de la simulation (*setproperties*).
- Cacher/montrer une variable (*addhideproperty* / *delhideproperty*).
- Changer la modifiabilité d'une variable (méthodes *addlockproperty*/ *dellockproperty*).

Nous devons ajouter une couche à cette simulation pour permettre un accès par un protocole de communication. Étant donné que notre simulation est développée en Java, le protocole le mieux adapté est RMI Java (Remote Method Invocation). Ce protocole fournit des mécanismes pour obtenir des références à des objets distants, échanger des informations et invoquer des méthodes entre deux applications Java (distantes ou locales). Pour cela, chaque application spécifie, dans un fichier d'interface, les méthodes qu'elle veut rendre accessibles et elle implémente les méthodes et les classes de l'interface.

Précisons ce qu'implique la construction de cette couche RMI :

1. Définition de l'interface de la simulation, en étendant la classe *java.rmi.Remote*. Dans notre exemple, nous profitons de la construction de cette couche pour utiliser directement comme noms des services les noms standards imposés par l'architecture. (Il faut remarquer que si la simulation fournit déjà les services à travers un protocole, il n'est alors pas nécessaire d'adapter à ce moment les noms et paramètres, et il faut utiliser le fichier *MappingData* pour cela.) La Figure 8-10 montre la déclaration des méthodes correspondantes.

```
import java.util.*;
import ARGOSPackage.*; // package des classes d'ARGOS
import SimPackage.*; // package des classes de la simulation

public interface SimulationInt extends java.rmi.Remote {
    Vector get_variables_values()
                throws java.rmi.RemoteException;
    void set_variables_values(Vector newValues)
                throws java.rmi.RemoteException;
    void set_element_visibility(String name, int visibility)
                throws java.rmi.RemoteException;
    …
}
```

Figure 8-10: Interface RMI de la simulation

2. Implémentation de l'interface en utilisant les méthodes de la classe Simulation. Pour cela, il est commode de créer une classe *CommServer* qui utilise la simulation pour satisfaire les demandes des clients et pour gérer la communication RMI. (cf. Figure 8-11)

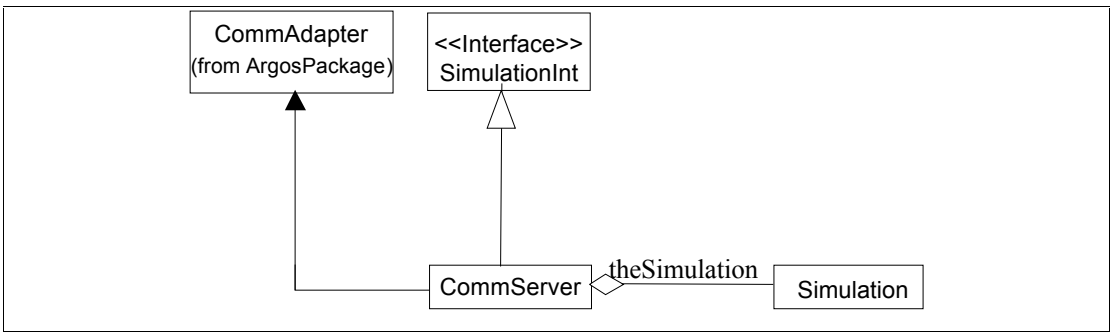


Figure 8-11: Schéma de classes du serveur de simulation

La classe *CommServer* doit adapter les services demandés aux méthodes proposées par la simulation. La Figure 8-12 montre, par exemple, l'implémentation de la méthode *set_element_visibility*.

```
      void set_element_visibility(String name, int visibility)
      {
            if (visibility==1) {
                  theSimulation.delhideproperty(name);
            }
            else {
                  theSimulation.addhideproperty(name);
            }
      }
```

Figure 8-12: Exemple d'implémentation de l'interface de la simulation

8.3.4.2 Adaptation du contrôle pédagogique

Nous continuons notre première étape d'adaptation en passant au contrôle pédagogique.

Le contrôle pédagogique, sous sa forme actuelle, invoque directement les méthodes de la classe qui implémente la simulation. Comme avec la simulation, il faut entourer l'application de contrôle pédagogique d'une couche communication. Le protocole le mieux adapté est ici encore RMI Java.

Il faut d'abord construire l'interface du contrôle pédagogique (cf. Figure 8-13). Comme nous voulons utiliser le contrôle pédagogique avec des simulations conformes à ARGOS, il est nécessaire d'implémenter dans le contrôle pédagogique le traitement des notifications de changements en provenance de la simulation.

```
import java.util.*;
import ARGOSPackage.*; // package des classes d'ARGOS

public interface ControlePedagogiqueInt extends java.rmi.Remote {
    void notify_variable(String nom, String type,String newvalue)
                throws java.rmi.RemoteException;
    void notify_action(String nom)
                throws java.rmi.RemoteException;
    …
}
```

Figure 8-13: Interface RMI de l'application de contrôle pédagogique

Nous devons ensuite manuellement adapter dans le code les invocations de méthodes de la simulation pour les rendre conformes à ARGOS. Il faut, par exemple, transformer les appels des méthodes `addhideproperty/delhideproperty` de la simulation en appels de la méthode `set_element_visibility d'ARGOS`.

Dans notre prototype, l'application de contrôle pédagogique est responsable de la surveillance des changements des variables. Avec ARGOS, cette responsabilité est du ressort du composant *ServiceAdapter*. Il faut donc modifier manuellement le code du contrôle pédagogique pour utiliser les services *init_monitoring_variable* et *notify_variable* d'ARGOS.

8.3.4.3 Contenu du fichier MappingData

Nous abordons maintenant les détails de la deuxième étape d'application d'ARGOS. Le fichier *MappingData* doit contenir la liste de services ARGOS fournis par la simulation, et, si nécessaire, les correspondances de paramètres.

Le fichier utilisé pour la simulation de notre prototype est montré dans la Figure 8-14. Nous donnons d'abord, sous l'intitulé [Services], la liste de ceux des services standards ARGOS qu'il est possible de satisfaire directement avec la simulation dont nous disposons. Nous précisons ensuite si besoin les correspondances de noms pour chaque service. Par exemple, pour le service *execute_action*, si l'action ARGOS demandée est *activate_simulation*, il faut exécuter la méthode *unlock()* de la simulation.

```
[Services]
get_variables_values
set_variables_values
set_variable_visibility
set_variable_modifiability
execute_action

[execute_action]
actionARGOS=activate_simulation
actionSim=unlock()
actionARGOS=deactivate_simulation
actionSim=lock()
actionARGOS=lockproperty(String name)
actionSim=addlockproperty(String name)
actionARGOS=unlockproperty(String name)
actionSim=hidelockproperty(String name)
…
```

Figure 8-14: Fichier *MappingData* pour SWIPS

8.3.4.4 Extension des composants d'ARGOS

Cette étape n'est nécessaire que pour un nouveau protocole.

L'extension du connecteur *ARGOSConnector* pour le protocole RMI consiste à implémenter une classe concrète qui hérite de la classe abstraite *ARGOSConnector* et la

classe *ProtocolCPRMI* correspondante pour le traitement de la communication avec le contrôle pédagogique.

L'extension du composant *CommAdapter* pour le protocole RMI est similaire. Il faut implémenter une classe concrète qui hérite de la classe abstraite *CommAdapter* et la classe *ProtocolSimRMI* correspondante pour le traitement de la communication avec la simulation.

8.4 Une démarche générale pour l'application d'ARGOS

Une simulation et un contrôle pédagogique peuvent interagir en utilisant ARGOS si :

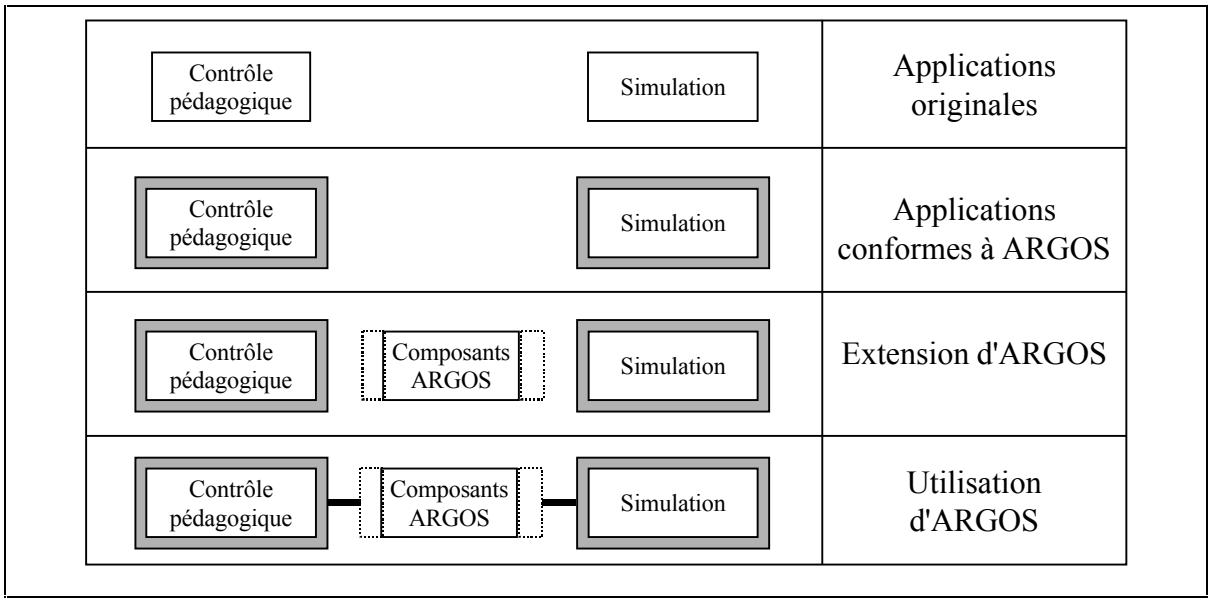
1. Les deux applications sont conformes à l'architecture.
2. Les protocoles utilisés par les applications sont implémentés dans les composants ARGOS.

Une simulation est conforme à l'architecture si elle fournit, à travers un protocole de communication, au moins les services minimums établis dans le paragraphe §6.2.3. Si la simulation ne fournit pas ces services, il est nécessaire de la modifier ou de l'entourer d'une couche responsable de fournir ces services et un protocole. Dans ce dernier cas, le choix du protocole dépend de l'outil ou du langage de programmation avec lequel la simulation est construite ; si c'est possible, il est préférable d'utiliser un protocole déjà supporté par ARGOS.

Un contrôle pédagogique est conforme à l'architecture s'il requiert de la simulation uniquement les services prévus par l'architecture et s'il utilise un protocole de communication. Si le contrôle pédagogique n'est pas conforme à l'architecture, il doit être modifié ou complété par une couche utilisant les services d'ARGOS et un protocole de communication. Le protocole choisi va dépendre du langage de programmation utilisé pour réaliser le contrôle pédagogique ; comme pour la simulation, il est préférable d'utiliser un protocole déjà supporté par ARGOS.

Enfin, si la simulation et/ou le contrôle pédagogique utilisent un protocole de communication qu'ARGOS ne supporte pas encore, il est nécessaire de compléter en conséquence les composants d'ARGOS.

La Figure 8-15 résume ce processus d'application d'ARGOS.



CONCLUSION

Dans cette thèse, j'ai, à la fois, présenté des résultats propres à l'équipe ARCADE, contexte de mon séjour, et mes travaux personnels qui se sont appuyés sur ceux de l'équipe et qui en constituent une prolongation. Je vais essayer ici de faire le point sur mes contributions personnelles, sur leurs apports et sur leurs perspectives.

Le travail fait

Dans la première partie, j'ai fait une étude des différents systèmes de production de simulations pédagogiques en prenant, comme grille de lecture, le modèle MARS introduit par l'équipe. Le chapitre 2 porte sur les aspects M (Modèle), A (Associations), et R (Représentation), et le chapitre 3 sur l'aspect S (Scénario).

L'outil OASIS de l'équipe ARCADE était en cours de réalisation à mon arrivée et j'ai pu ainsi contribuer directement à la spécification de son espace de travail Scénario.

J'ai ensuite pris en charge la totalité de l'expérimentation d'OASIS avec les enseignants du CAFIM, ce qui est décrit dans le chapitre 4. Le nombre réduit de participants a permis un suivi individuel en profondeur, ce qui s'est avéré très profitable. L'espace Scénario a été jugé, par les auteurs, comme très satisfaisant, tant en fonctionnalités offertes qu'en facilité d'utilisation. Quelques modifications d'interfaces ont été suggérées et ont été prises en compte dans l'outil.

Le bilan combiné de mon expérimentation au CAFIM et de celle de l'équipe ARCADE avec le TPEC m'a permis de dégager deux idées fortes qui ont conduit aux travaux exposés dans les parties 2 et 3. Ces deux idées sont, en grande partie, indépendantes et se sont traduites par des réalisations bien distinctes.

La **première idée** est issue d'une convergence de diverses remarques.

- Des outils généralistes, tels qu'OASIS, fournissent des formalismes puissants, mais trop distants des formalismes spécifiques à différentes spécialités, ce qui augmente pour les auteurs la difficulté de la modélisation.

- Ces outils, encore une fois du fait de leur généralité, ne peuvent prendre en compte des aspects spécifiques ou répétitifs de certaines applications, sauf de manière limitée par le biais de bibliothèques d'objets. L'expérimentation au TPEC a montré, avec GeneSimu, les avantages d'un outil plus spécialisé et de ce fait capable d'automatiser plus largement certaines tâches de l'auteur.

Ceci fait apparaître l'intérêt de disposer de systèmes auteurs spécialisés. On peut alors se demander comment faciliter le développement de ces systèmes et on est ainsi amené à s'intéresser non plus seulement aux auteurs de simulation pédagogique, mais aussi, en amont, aux développeurs de systèmes auteurs. J'ai alors souhaité proposer à cet effet un environnement adapté.

Mon idée est de proposer un framework qui intègre tout le savoir-faire acquis par l'équipe (modèle MARS, réalisation de MELISA, d'OASIS, de GeneSimu). La deuxième partie décrit cet aspect de mon travail. Le chapitre 5 fait un inventaire des techniques de Génie Logiciel qui facilitent la réutilisation. Dans le chapitre 6, je détaille ma proposition de framework. Le degré d'avancement de sa réalisation a permis de le valider sur deux exemples.

Avec le premier exemple, SIMARS, je montre que le framework proposé permet, grâce à ses mécanismes par défaut, d'obtenir, en un mois, un équivalent fonctionnel d'OASIS (à l'exception de l'espace Scénario). L'investissement fait dans le framework se traduit bien par une réduction substantielle de l'effort et du temps de développement. Avec le déuxième exemple, FENIX, je présente un usage plus typique de l'usage envisagé du framework.

La **deuxième idée**, permettre une certaine indépendance entre une simulation et le contrôle pédagogique, est venue des remarques de certains enseignants qui appréciaient beaucoup l'espace Scénario et auraient souhaité disposer aussi des mêmes fonctionnalités avec des simulations produites par d'autres outils qu'OASIS. Dans la troisième partie, j'étudie le problème de la communication entre deux applications : la simulation (sans exercices) et le contrôle pédagogique (qui définit et gère les exercices). Je fais dans le chapitre 7 une étude des mécanismes mis en œuvre dans les modes exercices de divers systèmes. Ceci me permet d'identifier avec une certaine précision les services que doivent fournir chacune des deux applications (simulation et contrôle pédagogique) pour pouvoir collaborer. Je propose ensuite dans le chapitre 8 l'architecture ARGOS qui permet cette collaboration en limitant les contraintes imposées aux deux applications. Je décris SWIPS, un exemple concret appliquant cette architecture, et j'en dégage une démarche de mise en œuvre d'ARGOS.

Les perspectives

Bien que mon expérimentation avec les enseignants ait permis une première validation de l'espace Scénario, il faudrait maintenant passer à une plus grande échelle pour aboutir à une validation statistiquement plus significative. Ceci semble possible à l'université J. Fourier, qui envisage de mettre en œuvre les résultats des projets européens ARIADNE et ARIADNE II dont notre équipe est partenaire avec le produit OASIS.

Le framework proposé doit encore être amélioré. Si la partie FREAK semble assez stable, l'outil d'édition doit, conformément à l'approche évolutive choisie, être confronté à plusieurs réalisations pour mieux identifier ses aspects stables. La simulation SWIPS est utilisée en Colombie dans un cours sur les réseaux. D'autres besoins de simulations sont en cours d'identification pour cet enseignement. Je dispose donc ici de bonnes opportunités pour tester le framework.

D'autre part, ce framework devra intégrer les résultats de la partie 3 afin que les simulations produites avec les outils spécialisés puissent collaborer avec des applications de contrôle pédagogique. Ce travail est en cours.

Il serait intéressant de prolonger l'expérience acquise en développant ce framework en abordant la construction d'un autre framework dédié, lui, à la production d'applications de contrôle pédagogique.

Il faut également poursuivre la validation de l'architecture ARGOS en l'utilisant pour faire communiquer des applications (simulations et contrôles pédagogiques) produites avec la plus grande variété possible de systèmes existants.

La plus intéressante perspective est l'intégration de cette architecture pour faire communiquer des simulations et des contrôles pédagogiques dans un contexte de formation à distance. Ceci est prévu dans le cadre du projet FORMID (FORMation Interactive à Distance) du laboratoire CLIPS [CLI 98]. L'équipe ARCADE pourra ainsi expérimenter non seulement des contrôles pédagogiques variés sur de nombreuses simulations, mais aussi l'exploitation des contrôles soit par un formateur (situations de tutorat) soit par des co-apprenants (travail collaboratif).

Et nous nous prenons à rêver d'un monde où la technologie se sera suffisamment rapprochée des besoins des formateurs pour qu'ils l'exploitent de façon naturelle, dans tout son potentiel pédagogique, et pour le plus grand bénéfice des apprenants d'un monde éducatif où les technologies ne seront plus "nouvelles" parce que formateurs et apprenants se les seront complètement appropriées.

BIBLIOGRAPHIE

[AIC 94] HYDE, J., MEDLEY M. **Simulation Interoperability.** Aviation Industry CBT Committee (AICC), Document No MPD010. May 1995.

[APR 92] **Dictionary of science and technology**. Academic Press, 1992.

[ARE 98] AREVALO, M., DIAZ, M., GOMEZ, G. **PDCERL: Plataforma para el desarrollo de ejercicios en laboratorios a distancia**. Rapport Ingénieur en Informatique, Université des Andes. Bogotá Colombie, Juin1998.

[ARI 98] ARISTIZABAL, G. **FÉNIX : Herramienta para la creación de simulaciones en matemáticas construida sobre el framework FREAK.** Rapport Master en Informatique, Université des Andes. Bogotá Colombie, Decembre, 1998.

[BAS 98] BASS, L., CLEMENTS, P., KAZMAN, R. **Software Architecture in Practice.** Addison Wesley, 1998.

[BAU 97] BÄUMER, D., GRYCZAN, G., KNOLL, R., LILIENTHAL, C., RIEHLE, D., ZÜLLIGHOVEN, H. **Framework Development for Large Systems**. Communications of the ACM, Vol 40, No 10, Octobre,1997.

[BEC 96] BECK, K., CROCKER, R., MESZAROS, G., VLISSIDES, J., COPLIEN, J. O., DOMINICK, L., PAULISCH, F. **Industrial experience with design patterns**. Proceedings of the 18th international conference on Software engineering, ICSE '96, Berlin, Allemagne, mars 1996.

[BIS 92] BISCHOFBERGER, W. POMBERGER G. **Prototyping-Oriented Software Development.** Springer-Verlag, 1992.

[BUS 96] BUSCHMANN, F., MEUNIER, R., ROHNERT H., SOMMERLAND P., STAL M. A **system of Patterns**. John Wiley and Sons. 1996.

[BOE 88] BOEHM, B.W. **A spiral model for software development and enhancement**. IEEE Computer, pages 61–72, May 1988.

[CAG 97] CAGNAT, J-M. **"OASIS: Manuel Utilisateur"**. Rapport interne, Equipe ARCADE, Laboratoire CLIPS-IMAG, Grenoble, France. 1997.

[CEC 88] CECCHINI, A. **Simulation is education**. IUAV-DAEST, Venice, Italie, p. 213-228,1988.

[CHA 98] CHAPPELL. D. **ActiveX Controls vs JavaBeans**. Component strategies magazine, janvier 1998.

[CJJ 98] CORTES, J. **SIMARS: Herramienta para la creación de simulaciones bajo el modelo MARS creada sobre el framework FREAK.** Rapport Master en Informatique, Université des Andes. Bogotá Colombie, Decembre, 1998.

[CLI 98] CLIPS-IMAG, **Rapport d'activité 1995-1998**, UMR-CNRS 5524, Grenoble, France, 1998.

[COA 92] COAD. P. **Object-Oriented Patterns**. Communications of the ACM, Vol 35 No 9, Septembre 1992.

[COA 95] COAD, P., NORTH, D., MAYFIELD, M. **Object Models: Strategies, Patterns, & Applications.** Prentice Hall. 1995.

[COP 92] COPLIEN, J.O. **Advanced C++ - Programming Styles and Idioms**. Addison Wesley, Reading, MA, 1992.

[COU 96] COULANGE, B. **Réutilisation du logiciel.** Masson, Paris,1996.

[COR 88] CORTES, G. **SimLab: simulador de tres experimentos de física**. Projet pour obtenir le diplôme d'ingénieur. Université des Andes, Bogotá-Colombia, 1988.

[COR 97] CORTES, G., COUDRET, F. **Validating the teacher's needs for simulation based exercises.** World Conference on Educational Multimedia/Hypermedia (EDMEDIA'97), Calgary, Canada. June 1997.

[COR 97a] CORTES, G., GUERAUD, V. **Helping the teacher to create simulation based exercises.** Computer Assisted Education in Engineering, CAEE'97, Cracovia, Polonia. Septembre 1997.

[COR 97b] CORTES, G. **OASIS - Report of experimentation with science teachers.** Rapport interne, Equipe ARCADE, Laboratoire CLIPS-IMAG, Grenoble, France. Octobre, 1997.

[COR 98] CORTES, G., GUERAUD, V. **Experimentation of an authoring tool for pedagogical simulations.** International Conference on Computer Aided Learning and Instruction in Science and Engineering, CALISCE'98, Göteborg, Sweden, 1998.

[COR 98a] CORTES, G. **OASIS: Un sistema autor para el desarrollo de simulaciones con propósito educativo.** "4to Congreso Iberoamericano de Informática Educativa", Brasil, Octobre, 1998.

[DEJ 91] De JONG, T. **Learning and Instruction with computer simulation**. Education and Computing**.** vol 6. Elsevier, 1991.

[DEJ 94] De JONG, T. & all. **SMISLE: System for Multimedia Integrated Simulation Learning Environments**. In Design and Production of Multimedia and Simulation-Based Learning Material. Kluwer Academic Publishers, 1994.

[DEJ 96a] De JONG, T. Van Jooligen, W. **Discovery learning with computer simulations of conceptual domains.** IST-Memo-96-O2, University of Twente, 1996.

[DEJ 96b] De JONG, T. Editeur. **Designing integrated computer simulation environments for discovery learning.** Servive Project (ET 1020), First project progress report. Decembre 1996.

[DIV 93] DIVER-STAMNES, A. **Simulating Society: An Experimental Approach to Teaching Race/Class Relations**. Annual Meeting of the American Phycological Association, Toronto, Canada, 1993.

[DOS 97] DOSCHER, D. HODGES, R. **SEMATECH's Experiences with the CIM Framework**. Communications of the ACM, Vol 40, No 10, Octobre,1997.

[DOU 95] DOULAI, P. **Computer assisted teaching/learning method for power systems education**. Dans Proceedings of the International Power Engineering Conference (IPEC'95), pages 746-750, Singapore, mars 1995.

[ESC 90] ESCOBEDO, H. **El computador en el laboratorio de física: Una alternativa para la enseñanza de la física en el bachillerato**. Revista de Informática Educativa, Vol. 3 - No 2, pp. 129-139, 1990.

[ELS 91] **Computer Simulations in an Instructional Context**. Education and Computing**.** vol 6. Elsevier, 1991.

[FAR 98] FARANCE, F., TONKEL, J. **LTSA specification, Learning Technology Systems Architecture.** Document de base du groupe Architecture and Reference Model du IEEE-LTSC. Mai 1998. http://www.manta.ieee.org/p1484/.

[FAY 97] FAYAD, M.E., SCHMIDT, D.C. **Object Oriented Application Frameworks**. Communications of the ACM, Vol 40, No 10, Octobre,1997.

[FLE 96] FLEMING, J., HORWITZ, C. **Applications of the Rapid Intelligent System Development Shell (RIDES).** Intelligent Tutoring Systems'96, Workshop on Architectures and Methods for Designing cost-effective and Reusable ITS. Montreal, Canada. June 1996.

[FOR 97a] FORTE E., WENTLAND FORTE M., DUVAL E. **The ARIADNE Project (Part I) : Knowledge Pools for Computer-based and Telematics-supported Classical, Open and Distance Education**. European Journal of Engineering Education, vol 22, n°1, p.61-74, 1997.

[FOR 97b] FORTE E., WENTLAND FORTE M., DUVAL E. **The ARIADNE Project (Part II) : Knowledge Pools for Computer-based and Telematics-supported Classical, Open and Distance Education**. European Journal of Engineering Education, vol 22, n°2, p.153-166, 1997.

[FRO 97] FRONT, A. **Développement de systèmes d'information à l'aide de patrons : Application aux bases de données actives**. Thèse de l'Université Joseph Fourier, 1997.

[GAL 92] GALVIS, A. **Teorías de Aprendizaje como sustento al diseño y evaluación de ambientes de enseñanza-aprendizaje.** Dans: Ingeniería de Software Educativo. Chapitre 4. Ediciones Uniandes. Bogotá - Colombia, 1992.

[GAM 93] GAMMA, E., HELM, R., VLISSIDES, J., JOHNSON, R. E. **Design Patterns: Abstraction and Reuse of Object-Oriented Design**. Proceedings ECOOP'93, LNCS 707, Springer-Verlag, Kaiserslautern, Allemagne, Juillet 1993.

[GAM 94] GAMMA E., HELM, R., VLISSIDES, J., JOHNSON, R. E.. **Design Patterns : Elements of reusable Object Oriented Software.** Addison-Wesley, 1994.

[GAR 95] GARLAN, D., DEWAYNE, E. P. **Introduction to the Special Issue on Software Architecture**. IEEE Transactions on Software Engineering, vol. 21, No. 4, avril 1995

[GAR 95a] GARLAN, D. **Research directions in software architecture**. ACM Computing Surveys, vol. 27, No. 2, juin 1995.

[GAR 95b] GARLAN, D. ,MONROE, R. , WILE, D. **ACME: An architectural Interconnection Language.** Rapport technique CMU-CS-95-219. Carnegie Mellon University, November 1995.

[GAT 93] GATTO, D. **The use of interactive computer simulations in training**. Australian Journal of Educational Technology, vol 9. No 2, 1993.

[GOL 84] GOLDBERG, A. Smaltalk 80, **The interactive Programming Language**. Addison Wesley, 1988.

[GUE 91] GUERAUD, V., CAGNAT J.M., PEYRIN, J.P. **Teaching and Learning made easier by the ARCADE Laboratory**. International Conference on Computer Aided Learning and Instruction in Science and Engineering, CALISCE'91, Lausanne, Switzerland, 1991.

[GUE 93] GUERAUD, V. PEYRIN, J.P., DAVID, J.P., PERNIN, J.P. **Environnements logiciels pour une intégration quotidienne de l'E.A.O. dans l'enseignement.** Conférence Hypérmedias et Apprentissage, Lille, 1993.

[GUE 99] GUERAUD, V., PERNIN, J-P. **Developing pedagogical simulations: Generic and specific authoring approaches.** Conference on Artificial Intelligence in Education. Le Mans, France, Juillet 1999.

[HER 94] HERZOG, J.M., FORTE, E.N. **A Goal Oriented Simulation in Chemical Thermodynamics**. International Conference on Computer Aided Learning and Instruction in Science and Engineering, CALISCE'94, Paris, 1994.

[HIX 90] Hix, D. **Generations of User-Interface Management Systems.** IEEE Software, September 1990.

[HOR 98] HORWITZ, C., FLEMING, J. **Effecting Principled Instruction in Virtual Environment Simulations: Lessons Learned**. Position paper. Workshop on Architectures for ISLE, Conference on Artificial Intelligence in Education. Japon, Août 1998.

[ID2 95] ID2 Research Group. **Instructional Simulator.** ID2 Research Group, Utah State University. 1995.

[JAC 97] JACOBSON, I., GRISS, M., JONSSON, P. **Making the reuse bussiness work.** IEEE Computer, Vol. 30, No. 10. Octobre 1997.

[JOH 91] JOHNSON, R. E., RUSSO, V. F. **Reusing Object-Oriented Desings**. Technical report UIUCDCS 91-1696, University of Illinois, 1991.

[JOH 97] JOHNSON, R. E. **Framework (Components + Patterns)**. Communications of the ACM, Vol 40, No 10, Octobre,1997.

[JOO 96] Van JOOLIGEN, W. R., De JONG, T. **Supporting the authoring process for simulation based discovery learning**. European Conference on Artificial Intelligence in Education. Lisbonne, Portugal, 1996.

[KOZ 98] KOZACZYNSKI, W., BOOCH, G. **Component Based Software Engineering.** IEEE Software, Vol. 15, No. 5, septembre / octobre 1998.

[LEA 97] LEA, D. **Concurrent Programming in Java™: Design Principles and Patterns**. Addisson-Wesley, 1997.

[LUC 95] LUCKHAM, D. C., VERA, J. **An Event-Based Architecture Definition Language.** IEEE Transactions on Software Engineering, Vol. 21, No. 9. September 1995.

[MAR 95] MARCELINO, M. J., MENDES, T. **SimBest A Set of integrative Tools to Support the Development of Computer-Based Educational Programs**. Proceedings of the 6th IFIP World Conference on Computers in Education, pp 945-951. Chapman and Hall, 1997.

[MAR 96] MARCELINO, M. J., MENDES, T. **SIM-BEST - Um sistema para apoiar actividades de modelação e de simulação en ensino**. Simpósio Investigação e Desenvolvimento de Software Educativo, Costa da Caparica, Portugal. Octobre,1996.

[MAR 97] MARCELINO, M. J., MENDES, T. **Diversifying the scenario in computer-based educational simulation activities with SimBest**. Proceedings. of CAL'97. Exeter, United Kingdom, March 1997.

[MIL 68] M.D. McILLROY. **Mass-produced software components**. Dans J.M Buxton, P. Naur, and B. Randell, editeurs, Software Engineering

Concepts and Techniques. NATO Conference on Software Engineering, pages 88–98, 1968.

[MON 97] MONROE, R., KOMPANEK, A., MELTON, R., GARLAN, D. **Architectural Styles, Design Patterns, and Objects.** IEEE Software. Janvier 1997.

[MUN 93] MUNRO, A., et all. **Attributed-Centered Simulation Authoring for Instruction.** World Conference on Artificial Intelligence in Education. Edinburgh, Scotland. August 1993.

[MUN 95] MUNRO, A. **RIDES QuickStart**. Behavioral Technology Laboratories, University of Southern California. 1995.

[MUN 95a] MUNRO, A. **RIDES Reference Manual**. Behavioral Technology Laboratories, University of Southern California. 1995.

[MYC 96] MARIÑO, O., CORTÉS, G.C.. **Cosmología: Un sistema heurístico para la integración de conocimientos en el escenario de la evolución de la humanidad**. Dans Memorias del III Congreso Iberoamericano de Informática Educativa, RIBIE / RIBIE-COL, Barranquilla, Colombia, 1996.

[NEM 97] NEMIROVSKY, A. M. **Building Object-Oriented Frameworks**. TALIGENT White Papers, Taligent INC. http://www.software.ibm.com/developer/library/oobuilding/.

[NIE 92] NIERSTRASZ, O, GIBBS, S., TSICHRITZIS, D. **Component Oriented Software Development.** Communications of the ACM,Vol. 32, No. 9. Septembre 1992.

[PER 96] PERNIN, J-P. **M.A.R.S. Un modèle opérationnel de conception de simulations pédagogiques**. Thèse de l'Université Joseph Fourier, 1996.

[PER 98] PERNIN, J-P. **Comparing two Authoring Approaches of Instructional Simulations: An Industrial Experimentation**. International Conference on Computer Aided Learning and Instruction in Science and Engineering, CALISCE'98. Göteborg, Sweden, 1998.

[PER 98a] PERNIN, J-P. **L'intégration effective de la simulation dans la formation technique : un enjeu à ne pas manquer**. Conférence Nouvelles Technologies de l'information et de la communication dans les formations d'ingénieurs et dans l'industrie. Rouen, France, November, 1998.

[PIL 96] PILKINGTON, R., ALEC, G. **Generating explanations in a simulation-based learning environment**. International Journal Human-Computer Studies, vol 45, p; 527-551, 1996.

[POS 97] POSNAK, E. J., LAVENDER, R G., VIN, H. M. **An Adaptive Framework for Developing Multimedia Software Components**. Communications of the ACM, Vol 40, No 10, Octobre, 1997.

[RAD 97] RADA, R., SCHOENING, J. R. **Educational Technology Standards**. Communications of the ACM Vol 40, No 9, 1997.

[REI 89] REIGELUTH, C.M., SCHWARTZ, E. **An Instructional Theory for the design of computer-based simulation**. Journal of Computer-Based Instruction. Vol 16, No 1, 1989.

[RIT 96] RITTER,S., KOEDINGER,K. **An architrecture for Plug-in Tutor Agents.** Journal of Artificial Intelligence in Education. Vol 7 No (3/4), 1996.

[RIT 97] RITTER, S. et all**. Tool: Agent Communication: Base Document.** IEEE Computer Society P1484. http://domino.psy.cmu.educ/ieee/, Septembre 1997.

[RBB 98] ROBBINS, J. **Overview of the Graph Editing Framework (GEF)**. Arcadia project at the University of California, Irvine, 1998.

[ROB 97] ROBERTS, D., JOHNSON, R. **Envolving Frameworks: A Pattern Lenguage for Developming Object-Oriented Frameworks**. PLoP '96 -- Allerton Park, Illinois, Sept. 4-6, 1996.

[ROS 92] ROSMALEN, P. Van (Ed). **SAM Definition Study.** Report to the European Comission-DELTA Project. May 1992.

[ROS 93] ROSMALEN, P. Van (Ed). **SAM Version 1 Report.** Report to the European Comission-DELTA Project. Septembre 1993.

[ROS 94] ROSMALEN, P. Van. **SAM Simulation and Multimedia.** Dans De Jong et all, Design and Production of Multimedia and Simulation Based Training. Kluwer Academic Publishers, June 1994.

[ROY 70] ROYCE,W.W.. **Managing the development of large software systems**. In Proc. IEEE Western Computer Conf., Los Alamitos, CA, August 1970. IEEE Computer Society Press.

[SAM 97] SAMACA, L.F. **Análisis y diseño de software educativo para pequeños y medianos empresarios en el área de marketing "Mercaplus"**. Projet pour obtenir le diplôme d'ingénieur. Université des Andes, Bogotá-Colombia, 1997.

[SHA 95] SHAW, M., DE LINE, R., KLEIN,D. V. , ROSS, T. L,. YOUNG, D. M., and ZELESNIK, G. **Abstractions for Software Architecture and Tools to Support Them.** IEEE Transactions on Software Engineering, vol. 21, No. 4, avril 1995.

[SHA 96] SHAW, M., GARLAN, D**. Software Architecture, perspectives on an emerging discipline**. Prentice Hall, 1996.

[SCH 99] SCHMIDT, D. **An Architectural Overview of the ACE Framework: A Case-study of Successful Cross-platform Systems Software**. Tools Issue, Login Magazin, USENIX, 1999.

[SKI 53] SKINNER, B.F. **Science and Human Behavior**. Macmillan, New York, 1953.

[SKI 68] SKINNER, B.F. **The Technology of Teaching**. Appleton-Century-Crofts, New York. 1968.

[SWA 96] SWAAK, J. **Support for Simulation-Based Learning; The Effects of Model Progression and Assignments on Learning about Oscillatory Motion**. Report IR018081, Twente University. Hollande, 1996.

[TAL 93] TALIGENT. **Leveraging Object-Oriented Frameworks**. TALIGENT White Papers, Taligent INC. http://www.software.ibm.com/developer/library/ooleveraging/.

[TAN 94] TANNENBAUM, A. **Computer Networks.** Third Edition. Ed. PRENTICE HALL. 1994.

[THO 90] THOMAS, R.A., LEWIS, D.O. **What can simulation add to conventional instruction?** Conference CATS'90, Barceelona, Spain, 1990.

[TOW 95] TOWNE, D. **Learning and Instruction in Simulation Environments**. Educational Technology Publications, 1995.

[VIV 91] VIVET, M. **Expertise pédagogique et Usage de tuteurs intelligents**. 13èmes Journées francophones sur informatique. Formation Intelligemment assistée par ordinateur, 1991.

[WAR 99] WARREN, K. **Beefeater Restaurants**. London Business School. Dans http://www.smsim.co.uk/beef/.

[ZEL 96] ZELESNIK, G. **The UniCon Language User Manual.** School of Computer Science, Carnegie Mellon University. Juin 1996.

SITES WWW PAR THEMES

Les références aux pages WEB indiquent la date de la dernière consultation faite par moi-même.

Outils pour la réalisation de simulations pédagogiques

[INT 99] Page du projet INTERACT : http://interact-www.eng.cam.ac.uk/
Juin 1999

[IS 99] Page du système Instructional Simulation :
http://www.coe.usu.edu/it/id2/instsim.html
Juin 1999

[RID 99] Page du Behavioral Technology Laboratories de l'université de la Californie du sud.
Projets RIDES et VIVIDIS : http://btl.usc.edu/
Juin 1999

[SER 99] Page du Projet SERVIVE:
http://www.simquest.to.utwente.nl/simquest/servive/
Juin 1999

Outils de simulation

[RAP 99] Page du système RAPID: http://www.emultek.com/
Juin 1999

[LOG 99] Page de Logal Express: http://www.logal.net/
Juin 1999

[INP 99] Page de Interactive Physic: http://www.krev.com/products/ip.html/
Juin 1999

[BER 99] Page de Interactive Physics Problem Set (Introductory Physics) de l'université de Berkeley : http://socrates.berkeley.edu:7521/projects/IPPS/
Juin 1999

[MOD 99] Page du programme Modellus de l'Université de Lisbonne : http://phoenix.sce.fct.unl.pt/modellus/
Juin 1999

Standards

[AIC 99] Page de Aviation Industrie CBT Committee
http://www.aicc.org/
Juin 1999

[LTS 99] Page de Learning Technology Standards Committee
http://ltsc.ieee.org/ltsc/
Juin 1999

Frameworks, Patrons et Architecture de Logiciel

[TAL 99] Site de Taligent Inc. Maintenant à IBM.
http://www.software.ibm.com/ad/taligent/
Juin 99

[GEF 99] Site du Framework GEF, de l'Université de Californie à Irvine
http://www.ics.uci.edu/pub/arch/gef/
Juin 1999

[SCH 99] Page de Douglas Schmidt, de l'Université de Washington
Framework ACE et patrons pour logiciels de communication
http://www.cs.wustl.edu/~schmidt/
Juin 1999

[SCG 99] Bibliographie en Systèmes Orientés Objet (frameworks, patrons et architectures). Base de données maintenue par le Software Composition Group, de l'Université de Berne.
http://www.iam.unibe.ch/cgi-bin/oobib
Juin 1999

Annexes

ANNEXE 1

SPECIFICATION DE SIMULATIONS DEVELOPPEES AVEC OASIS

Cette annexe donne les spécifications détaillées de quelques applications développées pendant l'expérimentation au CAFIM. On y a aussi placé un exemplaire « vide » de chacun des formulaires utilisés avec les enseignants.

1. **Spécification de la Pile de Daniell**

2. **Spécification du Théorème de Shannon**

3. **Spécification de la Machine de dosage automatique**

4. **Spécification de l'Asservissement d'une bille sur un rail**

5. **Formulaires pour la spécification des simulations**

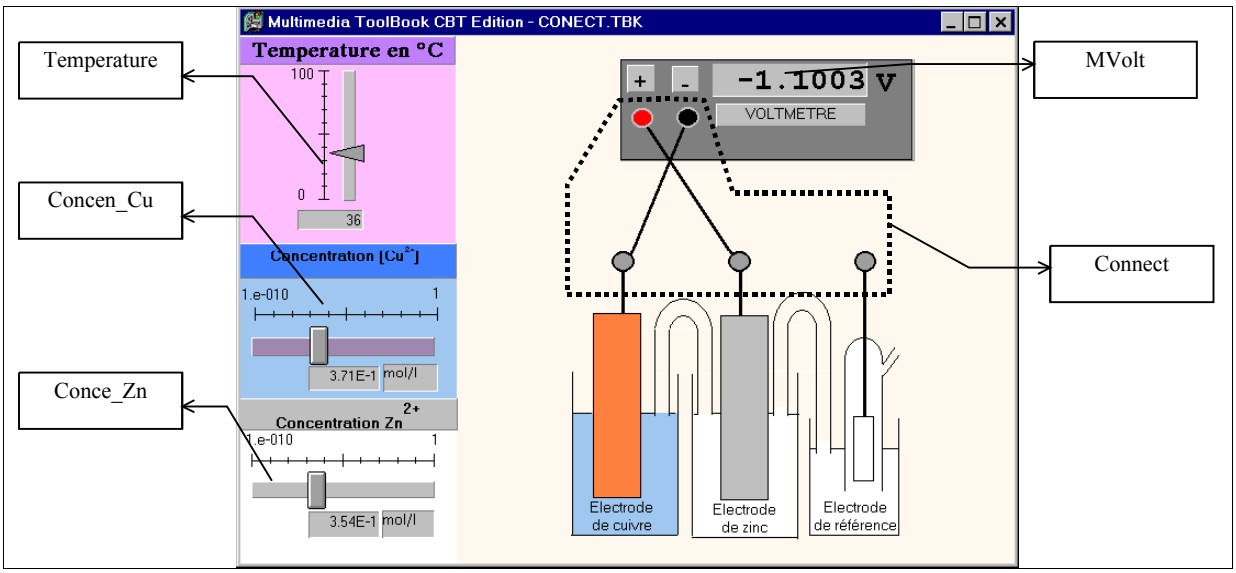
1. Spécification de la Pile de Daniell

Description générale

La pile est composé de trois solutions : cuivre, zinc et référence, chacune avec une électrode. L'élève peut changer la température ambiante et la concentration des solutions de cuivre et de zinc. Il dispose également d'un voltmètre pour mesurer le potentiel entre deux électrodes de son choix.

Représentation

Ecran



Interactions

Action	Effet
Drag-and-drop d'une électrode à un pôle (ou vice versa)	Connexion de l'électrode au pôle.
Déplacement du curseur de la température	Changement de la valeur de la température. La valeur mesurée par le voltmètre change si c'est nécessaire.
Déplacement du curseur de la concentration de zinc	Changement de la valeur de la concentration de zinc. La valeur mesurée par le voltmètre change si c'est nécessaire.
Déplacement du curseur de la concentration de cuivre	Changement de la valeur de la concentration de cuivre. La valeur mesurée par le voltmètre change si c'est nécessaire.

Modèle

Propriétés

Nom	Type	Valeur initiale	Publique	Description
P_E1	REAL	0	oui	Potentiel de la solution de cuivre
P_E2	REAL	0	oui	Potentiel de la solution de zinc
P_T	REAL	298		Température ambiante

Nom	Type	Valeur initiale	Publique	Description
P_Cnegatif	REAL	0	oui	Etat de connexion du pôle négatif du voltmètre. 1: A l'électrode de cuivre 2: A l'électrode de zinc 3: A l'électrode référence
P_Cpositif	REAL	0	oui	Etat de connexion du pôle positif du voltmètre. 1: A l'électrode de cuivre 2: A l'électrode de zinc 3: A l'électrode référence
P_mvolt	REAL	0	oui	Potentiel mesuré par le voltmètre.
P_CE1	REAL	0		Concentration de la solution de cuivre.
P_CE2	REAL	0		Concentration de la solution de zinc.

Méthodes

Nom	Publique	Paramètres	Description
M_mesure			Calcule le potentiel mesuré par le voltmètre en fonction des connections.
M_Cpositif	oui	pvalue	Met la valeur de P_Cpositif à pvalue.
M_Cnegatif	oui	pvalue	Met la valeur de P_Cnegatif à pvalue.
M_T	oui	pvalue	Met la valeur de P_T à pvalue.
M_E1	oui	pvalue	Met la valeur de P_CE1 à pvalue et calcule le potentiel P_E1 selon la formule: P_E1 = 0.34 + ((8.324*my P_T)/192970)* log(pValue,10)
M_E2	oui	pvalue	Met la valeur de P_CE2 à pvalue et calcule le potentiel P_E2 selon la formule: P_E2 = 0.34 + ((8.324*my P_T)/192970)* log(pValue,10)
F_potential		x	Retourne le potentiel de l'électrode désignée par x (x=1 => cuivre, x=2 => zinc, sinon 0)

Graphe d'états

Aucun

Associations

Modèle - Objet

Condition sur une propriété ou sur l'état du modèle	⇒	Objet	Action sur l'objet
P_mvolt = <AnyValue>	⇒	MVout	send M_SetValue (P_mvolt)

Objet - Modèle

Objet	Condition sur une propriété ou sur l'état de l'objet	⇒	Action sur le modèle
Concen_Cu	P_CurrentVal = <AnyValue>	⇒	send M_E1 (P_CurrentVal)
Conce_Zn	P_CurrentVal = <AnyValue>	⇒	send M_ E2 (P_CurrentVal)
Temperature	P_CurrentVal = <AnyValue>	⇒	send M_T (P_CurrentVal)
Connection	P_connected1 = <AnyValue>	⇒	send M_Cpositif (P_connected1)
Connection	P_connected2 = <AnyValue>	⇒	send M_Cnegatif (P_connected2)

Scénarios

Description des scénarios

Scénario 1 - Mesure du potentiel de cuivre

No	Nom	Situation à enregistrer	Situation attendue (ou à surveiller)	Réactivité
0	Situation initiale	Aucune électrode connectée au voltmètre.		
1	Mesure potentiel de cuivre	Electrode de cuivre connectée au pôle positif du voltmètre. Electrode référence connectée au pôle négatif du voltmètre.	Ignorer les valeurs du voltmètre et des potentiels. P_Cpositif = 1 P_Cnegatif=3 P_mvolt=anyvalue P_E1=anyvalue P_E2=anyvalue	Instructions: Vous devez mesurer le potentiel de l'électrode de cuivre par rapport à celui de l'électrode de référence. Feedback success: Vous avez réussi. On doit effectivement connecter l'électrode de référence au pôle - du millivoltmètre pour obtenir le potentiel de l'électrode de cuivre avec le signe convenable.
	Contrôle 1	Electrode de zinc connectée au pôle positif du voltmètre. Electrode référence connectée au pôle négatif du voltmètre.	Ignorer les valeurs du voltmètre et des potentiels. P_Cpositif = 2 P_Cnegatif=3 P_mvolt=anyvalue P_E1=anyvalue P_E2=anyvalue	Feedback : Vous mesurez bien le potentiel d'une électrode par rapport à celui de l'électrode de référence, mais c'est le potentiel de l'électrode de zinc et non celui de l'électrode de cuivre. Remedial: Vous devez modifier l'une des connections.
	Contrôle 2	Electrode de cuivre connectée au pôle positif du voltmètre. Electrode de zinc connectée au pôle négatif du voltmètre.	Ignorer les valeurs du voltmètre et des potentiels. P_Cpositif = 1 P_Cnegatif=2 P_mvolt=anyvalue P_E1=anyvalue P_E2=anyvalue	Feedback : Vous êtes en train de mesurer le potentiel de l'électrode de cuivre par rapport à celui de l'électrode de zinc. Ca n'est pas du tout ce qui est demandé. Remedial: Vous devez modifier les 2 connections.
	Contrôle 3	Electrode de zinc connectée au pôle positif du voltmètre. Electrode de cuivre connectée au pôle négatif du voltmètre.	Ignorer les valeurs du voltmètre et des potentiels. P_Cpositif = 2 P_Cnegatif=1 P_mvolt=anyvalue P_E1=anyvalue P_E2=anyvalue	Feedback : Vous êtes en train de mesurer le potentiel de l'électrode de zinc par rapport à celui de l'électrode de cuivre. Ce n'est pas cela qui est demandé. Remedial: Vous devez modifier les 2 connections.
	Contrôle 4	Electrode référence connectée au pôle positif du voltmètre. Electrode de cuivre connectée au pôle négatif du voltmètre.	Ignorer les valeurs du voltmètre et des potentiels. P_Cpositif = 3 P_Cnegatif=1 P_mvolt=anyvalue P_E1=anyvalue	Feedback : Vous pouvez lire le potentiel de l'électrode de cuivre, mais le signe n'est pas convenable. Remedial: Il faut intervertir les connexions. L'électrode référence doit toujours

			P_E2=anyvalue	être raccordé à la borne - du voltmètre si l'on veut lire le potentiel mesuré avec le bon signe.
	Contrôle 5	Electrode référence connectée au pôle positif du voltmètre. Electrode de zinc connectée au pôle négatif du voltmètre.	Ignorer les valeurs du voltmètre et des potentiels. P_Cpositif = 2 P_Cnegatif=1 P_mvolt=anyvalue P_E1=anyvalue P_E2=anyvalue	Feedback : Vous mesurez le potentiel de l'électrode de référence par rapport à celui de l'électrode de zinc. Remedial: Changez donc de lunettes. Il faut bien établir 2 connections, mais différemment.

Scénario 2 - Mesure de la f.e.m. de la pile

No	Nom	Situation à enregistrer	Situation attendue (ou à surveiller)	Réactivité
0	Situation initiale	Aucune électrode connectée au voltmètre.		
1	Mesure potentiel de zinc	Electrode de zinc connectée au pôle positif du voltmètre. Electrode référence connectée au pôle négatif du voltmètre.	Ignorer les valeurs du voltmètre et des potentiels. P_Cpositif = 2 P_Cnegatif=3 P_mvolt=anyvalue P_E1=anyvalue P_E2=anyvalue	Instructions: Vous pouvez mesurer le potentiel de chaque électrode de la pile. Prière de commencer par l'électrode zinc. Merci. Feedback success: Un papier et un crayon vous permettrons de noter la valeur de ce potentiel. Son unité ? Les V/Référence. Feedback failure: Il faut revoir l'exercice "Mesure_Cu".
	Contrôle 1 : Mesure du potentiel de cuivre	Electrode de cuivre connectée au pôle positif du voltmètre. Electrode référence connectée au pôle négatif du voltmètre.	Ignorer les valeurs du voltmètre et des potentiels. P_Cpositif = 1 P_Cnegatif=3 P_mvolt=anyvalue P_E1=anyvalue P_E2=anyvalue	Feedback : Il semblait que l'on vous avez demandé de mesurer d'abord le potentiel de l'électrode de zinc !
	Contrôle 2 : Mesure de le fem	Electrode de cuivre connectée au pôle positif du voltmètre. Electrode de zinc connectée au pôle négatif du voltmètre.	Ignorer les valeurs du voltmètre et des potentiels. P_Cpositif = 1 P_Cnegatif=2 P_mvolt=anyvalue P_E1=anyvalue P_E2=anyvalue	Feedback : Vous avez décidé de brûler les étapes...mais peut-être est-ce un hasard ? Mais savez-vous quelle est l'électrode positive de la pile ?
2	Mesure du potentiel de cuivre	Electrode de cuivre connectée au pôle positif du voltmètre. Electrode référence connectée au pôle négatif du voltmètre.	Ignorer les valeurs du voltmètre et des potentiels. P_Cpositif = 1 P_Cnegatif=3 P_mvolt=anyvalue P_E1=anyvalue P_E2=anyvalue	Instructions: Vous avez mesuré le potentiel de l'électrode de zinc. Vous devez maintenant mesurer celui de l'électrode de cuivre. Feedback success: Puisque vous avez enfin trouvé un papier et un crayon, notez donc le potentiel de l'électrode de cuivre. Pouvez-vous calculer la fem de la

				pile Daniell ? Feedback failure: Il faut revoir l'exercice "Mesure_Cu".
3	Mesure de le fem	Electrode de cuivre connectée au pôle positif du voltmètre. Electrode de zinc connectée au pôle négatif du voltmètre.	Ignorer les valeurs du voltmètre et des potentiels. P_Cpositif = 1 P_Cnegatif=2 P_mvolt=anyvalue P_E1=anyvalue P_E2=anyvalue	Instructions: Vous pouvez maintenant connecter les électrodes de cuivre et de zinc au millivoltmètre de façon à lire la fem. Rappel : la fem est une grandeur positive.
	Contrôle 1 : Inverse	Electrode de zinc connectée au pôle positif du voltmètre. Electrode de cuivre connectée au pôle négatif du voltmètre.	Ignorer les valeurs du voltmètre et des potentiels. P_Cpositif = 2 P_Cnegatif=1 P_mvolt=anyvalue P_E1=anyvalue P_E2=anyvalue	Feedback : Vous avez mesuré la fem changée de signe parce que la polarité de la pile est opposée à celle qui est indiquée sur les bornes du millivoltmètre. Remedial: Il faut intervertir les connections pour avoir la fem.

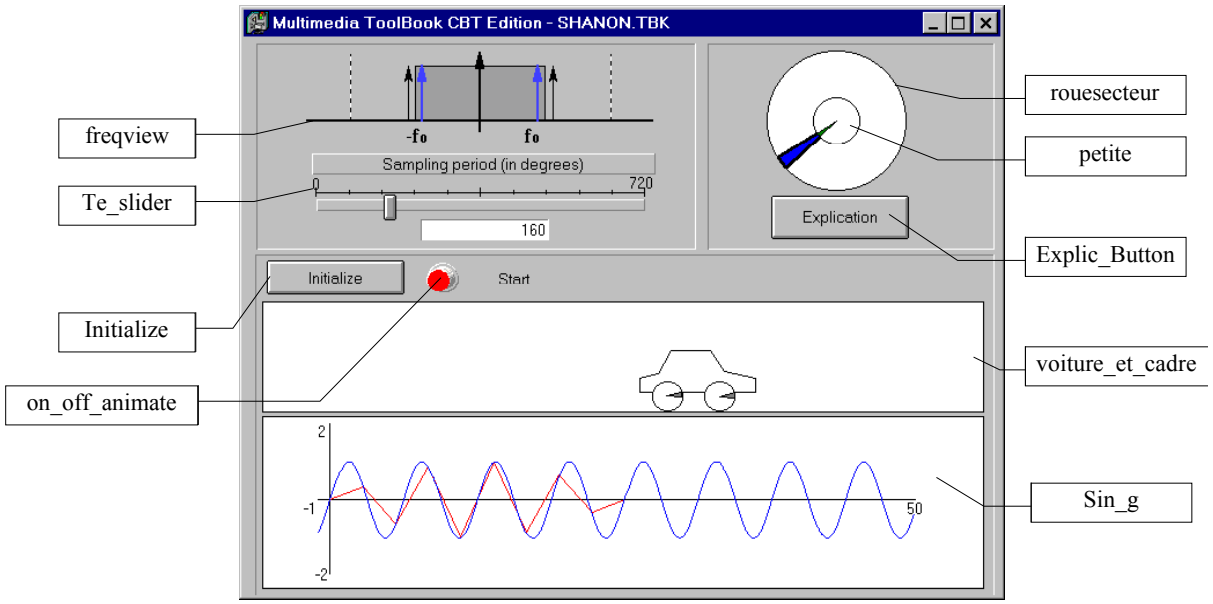
2. Spécification de la simulation du Théorème de Shannon

Description générale

Cette simulation a pour but d'illustrer le théorème de Shannon. La simulation considère une voiture qui roule à une vitesse constante, sur laquelle se font des échantillonnages de l'angle que fait le secteur de cercle avec l'horizontale (voir représentation). L'élève peut choisir une période d'échantillonnage et voir dans un graphique le signal original et le signal échantillonné.

Représentation

Ecran



Interactions

Action	Effet
Clic sur le bouton *Explication*	La roue *petite* avance son secteur selon les degrés fixés dans la période d'échantillonnage par petits pas de 1/10 de la période. Après, la roue *rouesector* avance son secteur selon les degrés fixés dans la période d'échantillonnage.
Clic sur le bouton *Initialize*	Initialise la voiture à la position initiale, efface la courbe rouge du graphique, et initialise les deux roues à l'angle 0.
Déplacement du curseur période d'échantillonnage	Change la période d'échantillonnage. L'objet freqview visualise la nouvelle valeur.
Clic sur le bouton *Start/Stop*	Met en marche/Arrête l'animation de la voiture. Quand l'animation est en marche, la voiture se déplace selon la période d'échantillonnage et la courbe rouge du graphique montre le sinus de l'angle échantillonné.

Modèle

Propriétés

Nom	Type	Valeur initiale	Publique	Description
P_angle_cr	REAL	0	oui	Angle du secteur de la roue extérieure.
P_periode	REAL	30	oui	Période d'échantillonnage choisie.
P_angle_pt	REAL	0	oui	Angle du secteur de la roue petite.
P_t	REAL	0	oui	Angle total de la voiture en radians.
P_y	REAL	0	oui	Sinus de P_t.
P_angle_voiture	REAL	0	oui	Angle du secteur des roues de la voiture.
P_initialiser	LOGICAL	False	oui	Change sa valeur quand l'utilisateur clique sur le bouton *Initialize*.

Méthodes

Nom	Publique	Paramètres	Description
M_initialiser	oui	pValue	Change la valeur de P_initialiser en pValue
M_calc_new_angle_voiture			Incrément P_angle_voiture en P_periode et actualise P_t et P_y
M_calc_new_angle	oui		Change la valeur de P_angle_pt par pas de 1/10 de P_periode dix fois. Après, incrémente P_angle_cr de P_periode.
M_periode	oui	pValue	Change la valeur de P_periode en pValue
M_Init	oui		Restitue en plus, à l'initialisation d'OASIS, la valeur de P_periode.

Graphe d'états

Quand la simulation est dans l'état *anime*, la voiture est en mouvement. Nous utilisons le script d'activité de l'état *anime* pour calculer un nouvel angle pour les secteurs des roues. Chaque fois que le script s'exécute les roues de la voiture tournent d'un angle de P_periode.

Associations

Modèle - Objet

Condition sur une propriété ou sur l'état du modèle	⇒	Objet	Action sur l'objet
P_angle_cr = <AnyValue>	⇒	rouesecteur	send M_angle (P_angle_cr)
P_angle_pt = <AnyValue>	⇒	petite	send M_angle (P_angle_pt)
P_y = <AnyValue>	⇒	sin_g	send M_appendvert (2,P_t,P_y)
P_periode = <AnyValue>	⇒	Te_slider	send M_SetValue (P_periode)
P_periode = <AnyValue>	⇒	freqview	send M_Te (P_periode)
P_angle_voiture = <AnyValue>	⇒	voiture_et_cadre	send M_angle (P_angle_voiture,P_periode)
P_initialiser = <AnyValue>	⇒	sin_g	send M_init_curve (2)

P_initialiser = <AnyValue>	⇒	freqview	send M_Init ()
P_initialiser = <AnyValue>	⇒	voiture_et_cadre	send M_Init ()

Objet - Modèle

Objet	Condition sur une propriété ou sur l'état de l'objet	⇒	Action sur le modèle
explic_Button	P_Trigger = TRUE	⇒	send M_calc_new_angle ()
Te_slider	P_CurrentVal = <AnyValue>	⇒	send M_periode (P_CurrentVal)
on_off_animate	P_ActualState = TRUE	⇒	Send Event arrete_vers_anime
on_off_animate	P_ActualState = FALSE	⇒	Send Event anime_vers_arrete
Initialize	P_Trigger = TRUE	⇒	send M_initialiser (TRUE)
Initialize	P_Trigger = TRUE	⇒	send M_Init ()

Scénarios

Description des scénarios

Scénario 1 - Reconstruction approprié du signal originel

Dans ce scénario, approprié incorrect

No	**Nom**	**Situation à enregistrer**	**Situation attendue** (ou à surveiller)	**Réactivité**
0	Situation initiale	Enregistrer la situation après appui sur le bouton *initialiser*. Le bouton *start/stop* doit avoir avec le label *start*.		Instructions (de la réactivité globale du scénario: ENONCE: On considère une voiture qui roule à une vitesse constante. En bas, l'axe horizontal représente l'angle que fait le secteur de cercle avec l'horizontale et l'axe vertical représente la projection de l'extrémité droite du secteur sur la verticale. Une prise de vues est effectuée à chaque fois que le secteur a réalisé une rotation de Te degrés. Changer la période d'échantillonnage (Te) de manière à récupérer, après reconstruction idéale, la fréquence du signal originel. Feedback if success : Effectivement, le mouvement de la voiture est conforme à la réalité
1	Période appropriée	P_periode]0-180[P_t >1	P_periode]0-180[P_t >1	Instructions: Choisissez une période d'échantillonnage, initialisez, ensuite démarrez la simulation. Feedback if success : Vous avez fait un bon choix. En fait toute valeur dans l'intervalle (0,180°) est correcte. Feedback if failure : ERREUR ! Essayez une autre valeur de Te. Remedial : Regardez en haut le spectre du signal. La fréquence du signal

				récupéré se situe à l'intérieur du filtre passe-bas idéal entre [-fe/2, fe/2]. Vous pouvez également vous servir du bouton EXPLICATION pour regarder le mouvement des roues pas par pas.
	Contrôle 1	P_periode]180-360[P_t >20	P_periode]180-360[P_t >20	Feedback : Attention ! Le sens de rotation de la roue est contraire au sens normal. Remedial : Essayez une autre valeur pour la période d'échantillonnage.

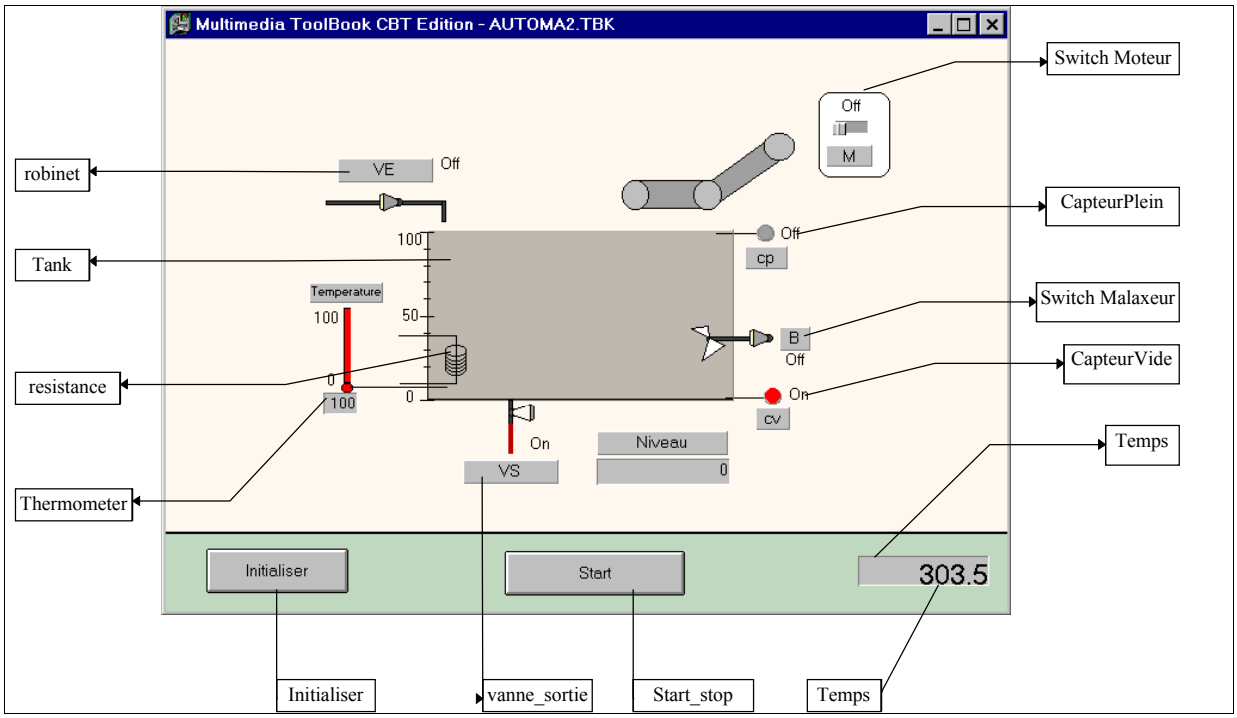
3. Spécification de la machine de dosage automatique

Description générale

Cette simulation a pour but de simuler une machine doseur/malaxeur. La machine a un réservoir de 100 litres, une vanne pour ajouter des produites liquides, un tapis pour ajouter des briquettes solubles, une vanne de sortie pour vider le réservoir, un malaxeur, un système de chauffage, un thermomètre pour lire la température du mélange et deux capteurs. L'élève peut manipuler la machine grâce aux commandes du clavier ou avec la souris.

Représentation

Ecran



Interactions

Action	Effet
Appui sur la touche E ou clic sur l'interrupteur de la vanne VE	Ouverture/Fermeture de la vanne VE.
Appui sur la touche T ou clic sur la résistance	Activation/inactivation du chauffage. La résistance est jaune si le chauffage est actif.
Appui sur la touche M ou clic sur l'interrupteur du moteur M	Activation/inactivation du moteur du tapis, M. Si le moteur est actif, des briquettes tombent dans le réservoir.
Appui sur la touche S ou clic sur l'interrupteur de la vanne VS	Ouverture/Fermeture de la vanne VS.
Appui sur la touche B ou clic sur l'interrupteur du malaxeur	Activation/inactivation du malaxeur, B. Si le malaxeur est actif, il tourne.
Clic sur le bouton *initialiser*	Initialise la simulation : tous les interrupteurs éteints, les vannes fermées, le réservoir vide et la température à la température ambiante.
Clic sur le bouton *start*	Si le nom du bouton est *start*, met la simulation en marche et change le nom du bouton pour *stop*. Si le nom du bouton est *stop*, arrête la simulation et change le nom du bouton pour *start*.

Modèle

Propriétés

Nom	Type	Valeur initiale	Publique	Description
P_temperature	REAL	0	oui	Température du mélange.
P_NIVEAU	REAL	0	oui	Niveau du réservoir [0-100]
P_VE	LOGICAL	False	oui	True si la vanne d'entrée du produit est ouverte
P_VS	LOGICAL	False	oui	True si la vanne de sortie du réservoir est ouverte
P_Chauffage	LOGICAL	False	oui	True si le chauffage est actif
P_Moteur	LOGICAL	False	oui	True si le moteur du tapis est allumé.
P_Brassage	LOGICAL	False	oui	True si le malaxeur est allumé.
P_ve_securite	LOGICAL	False	oui	True si la sécurité de la vanne d'entrée est active. Si active, la vanne d'entrée se ferme automatiquement quand le réservoir est plein.
P_VS_si_cuve_vide	LOGICAL	False	oui	True si la sécurité de la vanne de sortie est active. Si active, la vanne de sortie se ferme automatiquement quand le réservoir est vide.
P_Cuve_pleine	LOGICAL	False	oui	True si le réservoir est plein.
P_Cuve_vide	LOGICAL	True	oui	True si le réservoir est vide.
P_t	REAL	0	oui	Temps total de la simulation. Ce temps est initialisé à 0 chaque fois que la simulation est initialisée.
P_dt	REAL	0.5		Delta du temps à chaque pas de la simulation.
P_etat_melange	REAL	0	oui	Représente le grade de mélange du produit [0,9] : 0 totalement mélangé, 9 non mélangé.
P_Nb_briques_en_dt	REAL	1		Quantité de briquettes qui tombent du tapis en dt.
P_Nb_briques_additionnes	REAL	0	oui	Quantité de briquettes additionnées au mélange.
P_temperature_ambiante	REAL	20		Valeur de la température ambiante.

Méthodes

Nom	Publique	Paramètres	Description
`M_Moteur`	`oui`	pValue	Changent la valeur de la propriété du même nom (M_Moteur _> P_Moteur, etc) par pValue.
`M_Brassage`	`oui`	pValue	
`M_VS_si_cuve_vide`	`oui`	pValue	
`M_VS`	`oui`	pValue	
`M_VE_securite`	`oui`	pValue	
`M_chauffage`	`oui`	pValue	
`M_VE`	`oui`	pValue	
`M_Init`	`oui`		

Graphe d'états

Quand la simulation est en marche, le temps passe, c'est-à-dire que la machine fonctionne et actualise ses propriétés. Pour modéliser ce passage du temps, nous utilisons le script d'activité de l'état *Marche*. Chaque fois que le script s'exécute, nous supposons que P_dt secondes ont passé et nous actualisons toutes les propriétés en fonction de l'état de la machine: si les vannes sont ouvertes, il faut changer le niveau ; si le chauffage est allumé, il faut changer la température, etc.
C'est dans ce script (voir code) que l'on peut changer le comportement de la machine.

Associations

Modèle - Objet

Condition sur une propriété ou sur l'état du modèle	⇒	**Objet**	**Action sur l'objet**
P_NIVEAU = <AnyValue>	⇒	Tank	send M_SetValue (P_NIVEAU)
P_VE = <AnyValue>	⇒	robinet	M_SetState (P_VE)
P_Cuve_vide = <AnyValue>	⇒	CapteurVide	send M_SetState (P_Cuve_vide)
P_Cuve_pleine = <AnyValue>	⇒	CapteurPlein	send M_SetState (P_Cuve_pleine)
P_temperature = <AnyValue>	⇒	Thermometer	send M_SetValue (P_temperature)
P_Moteur = <AnyValue>	⇒	Switch Moteur	send M_SetState (P_Moteur)
P_Brassage = <AnyValue>	⇒	Switch Malaxeur	send M_SetState (P_Brassage)
P_VS = <AnyValue>	⇒	vanne__sortie	send M_SetState (P_VS)
P_Chauffage = <AnyValue>	⇒	resistance	send M_SetState (P_Chauffage)
P_etat_melange = <AnyValue>	⇒	Tank	send M_SetPattern (P_etat_melange)
P_t = <AnyValue>	⇒	Temps	send M_SetValue (P_t)
State = Marche	⇒	start_stop	send M_SetState (TRUE)
State = Arret	⇒	start_stop	send M_SetState (FALSE)

Objet - Modèle

Objet	**Condition sur une propriété**	⇒	**Action sur le modèle**
robinet	P_ActualState = <AnyValue>	⇒	send M_VE (P_ActualState)
resistance	P_ActualState = <AnyValue>	⇒	send M_chauffage (P_ActualState)
Initialiser	P_Trigger = TRUE	⇒	send M_Init ()
Switch Moteur	P_ActualState = <AnyValue>	⇒	send M_Moteur (P_ActualState)
Switch Malaxeur	P_ActualState = <AnyValue>	⇒	send M_Brassage (P_ActualState)
vanne__sortie	P_ActualState = <AnyValue>	⇒	send M_VS (P_ActualState)
start_stop	P_state = TRUE	⇒	Send Event Arrete_Enmarche
start_stop	P_state = FALSE	⇒	Send Event Enmarche_arrete

Scénarios

Description des scénarios

Scénario 1 - Elaboration gomme X

Enoncé:
Le processus pour produire la gomme X est décrit ci-dessous:

Ajouter 50 litres d'eau.
Chauffer l'eau jusqu'à une température de 50°C.
Ajouter 25 briques de base de gomme X
Mélanger jusqu'à la dissolution complète des briques.
Chauffer le mélange jusqu'à une température de 80°C.

Vous devez écrire un grafcet pour produire la gomme X et tester votre grafcet avec la simulation de la machine doseur/malaxeur.

No	**Nom**	**Situation à enregistrer**	**Situation attendue** (ou à surveiller)	**Réactivité**
0	Situation initiale	Tous les interrupteurs éteints. Simulation arrêtée et temps = 0.		
1	Ajouter l'eau	Tous les interrupteurs éteints. Simulation arrêtée. P_Niveau = 50	Ignorer la valeur du temps et l'état de la simulation.	Instructions: Ajouter l'eau.
	Contrôle 1	Tous les interrupteurs éteints sauf le chauffage. Simulation en marche. P_Niveau = 3	P_Niveau: [0,10]	Feedback : Le liquide n'est pas suffisant pour chauffer.
2	Chauffer	Tous les interrupteurs éteints. Simulation arrêtée. P_Niveau = 50 P_temperature = 50	Ignorer la valeur du temps et l'état de la simulation. P_temperature: [48,52]	Instructions: Chauffer.
3	Ajouter briquettes	Tous les interrupteurs éteints. Simulation arrêtée. P_Niveau = 50 `P_Nb_briques_additionnes = 25`	Ignorer la valeur du temps et l'état de la simulation.	Instructions: Ajouter briquettes..
4	Mélanger	Tous les interrupteurs éteints. Simulation arrêtée. P_Niveau = 50 `P_etat_melange = 0`	Ignorer la valeur du temps et l'état de la simulation.	Instructions: Ajouter briquettes.
5	Chauffer	Tous les interrupteurs éteints. Simulation arrêtée. P_Niveau = 50 P_temperature = 80	Ignorer la valeur du temps et l'état de la simulation. P_temperature: [78,82]	Instructions: Chauffer.
6	Vider	Tous les interrupteurs éteints. Simulation arrêtée. P_Niveau = 0	Ignorer la valeur du temps et l'état de la simulation.	Instructions: Vider le réservoir

Scénario 2 - Séquences simultanées

No	**Nom**	**Situation à enregistrer**	**Situation attendue** (ou à surveiller)	**Réactivité**
0	Situation initiale	Tous les interrupteurs éteints. Simulation arrêtée est temps = 0.		
1	Ajouter l'eau et chauffer	Tous les interrupteurs éteints. Simulation arrêtée. P_Niveau = 50 P_temperature >=20	Ignorer la valeur du temps et l'état de la simulation.	Instructions: Ajouter 50 l. d'eau en chauffant en même temps.
	Contrôle 1 Que remplir	Tous les interrupteurs éteints sauf la vanne d'entrée. P_Niveau = 3	Ignorer la valeur du temps et l'état de la simulation. P_Niveau: [0,10]	Feedback : Vous devez remplir et chauffer en même temps.
	Contrôle 2 Que chauffer	Tous les interrupteurs éteints sauf le chauffage. Simulation en marche.	Ignorer la valeur du temps et l'état de la simulation.	Feedback : Vous devez remplir et chauffer en même temps.

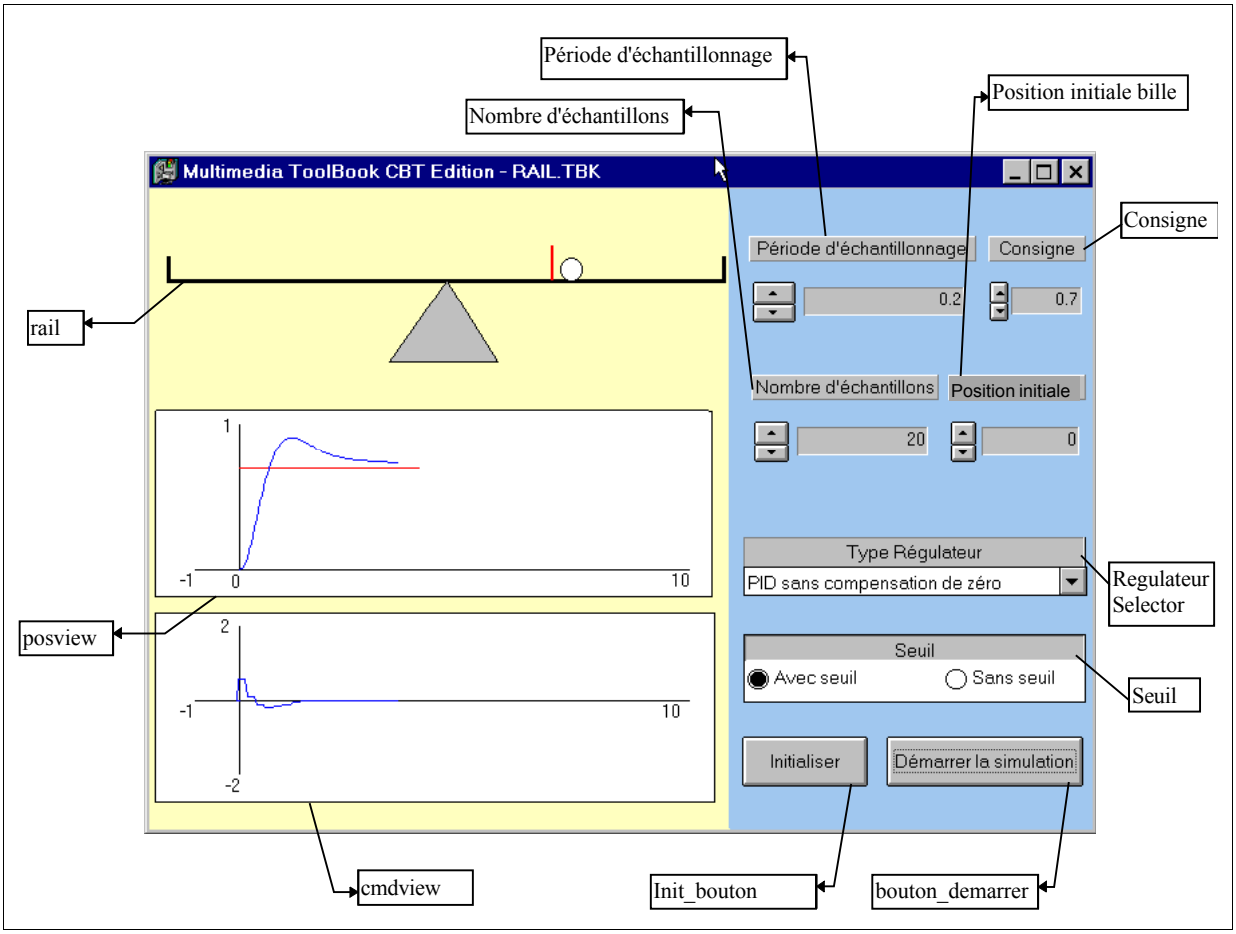
4. Spécification de l'Asservissement d'une bille sur un rail

Description générale

Cette simulation permet d'asservir la position d'une bille sur un rail avec un correcteur numérique (régulateur). L'utilisateur peut choisir le type de régulateur à utiliser pour l'asservissement. Il peut également choisir une période d'échantillonnage, le nombre d'échantillons, la position initiale de la bille, la consigne (position finale de la bille) et le type de rail (avec ou sans seuil). La simulation montre un graphique de la position de la bille et un graphique de l'angle du rail, par rapport au temps.

Représentation

Ecran



Interactions

Action	Effet
Clic sur les boutons **t** ou **u**	Augmente/Diminue la valeur de la variable associée.
Clic sur les boutons Avec seuil/Sans Seuil	Change le paramètre de Seuil selon le bouton choisi.
Clic sur la liste Type Regulateur	Permet de choisir un régulateur dans la liste proposée.
Clic sur le bouton initialiser	Initialise la simulation : initialise les paramètres aux valeurs par défaut et efface les graphiques.
Clic sur le bouton Démarrer la simulation	Efface les graphiques et lance la simulation pour le nombre d'échantillons établis par l'utilisateur. Le graphique posview montre la position de la bille par rapport au temps et le graphique cmdview l'angle du rail par rapport au temps.

Modèle

Propriétés

Nom	Type	Valeur initiale	Publique	Description
P_NbEch	REAL	20	oui	Nombre d'échantillons
P_INTRA	REAL	5		Nombre de calculs entre échantillons
P_theta	REAL	0	oui	Angle du rail par rapport à l'horizontale
P_y	REAL	0	oui	Position de la bille sur le rail.
P_CX	REAL	0.7	oui	Consigne.
P_Te	REAL	0.2	oui	Période d'échantillonnage
P_nb_Te1	REAL	0	oui	Variable de calcul.
P_long_rail	REAL	1		Longueur du rail
P_pos_init_bille	REAL	0	oui	Position initiale de la bille
P_initialiser	LOGICAL	True	oui	Change de valeur quand l'utilisateur clique sur le bouton Initialiser
P_Type_Regulateur	REAL	1	oui	Numéro du régulateur sélectionné
P_tableau_consigne	STRING		oui	Liste avec deux points qui définissent la courbe de la consigne.
P_Seuil	REAL	1	oui	Egal à 1 si "avec seuil" est choisi
P_seuil_bool	LOGICAL	True	oui	Choix de seuil
P_theta_deg	REAL	0	oui	Angle du rail en degrés.
P_temps	REAL	0	oui	Temps de la simulation.

Méthodes

Nom	Publique	Paramètres	Description
M_Seuil	oui	pValue	Change la valeur de la propriété P_seuil en pValue et actualise P_seuil_bool
M_pos_init_bille	oui	pValue	Change la valeur de la propriété P_pos_init_bille en pValue
M_Simuler	oui		Méthode qui fait l'asservissement de la position de la bille avec les paramètres fixés par l'utilisateur
M_Type_Regulateur	oui	pValue	Change la valeur de la propriété P_Type_Regulateur en pValue
M_Te	oui	pValue	Change la valeur de la propriété P_Te en pValue
M_CX	oui	pValue	Change la valeur de la propriété P_Te en pValue Actualise P_Tableau_consigne
M_NbEch	oui	pValue	Change la valeur de la propriété P_NbEch en pValue Actualise P_Tableau_consigne
M_Init	oui		Initialise les variables avec les valeurs par defaut.

Associations

Modèle - Objet

Condition sur une propriété ou sur l'état du modèle	⇒	Objet	Action sur l'objet
P_NbEch = <AnyValue>	⇒	Nombre d'échantillons	send M_SetValue (P_NbEch)
P_Te = <AnyValue>	⇒	Période d'échantillonnage	send M_SetValue (P_Te)
P_CX = <AnyValue>	⇒	Consigne	send M_SetValue (P_CX)
P_pos_init_bille = <AnyValue>	⇒	Position initiale bille	send M_SetValue (P_pos_init_bille)

Condition sur une propriété ou sur l'état du modèle	⇒	Objet	Action sur l'objet
P_nb_Te1 = <AnyValue>	⇒	posview	send M_appendvert (1,P_temps,P_y)
P_nb_Te1 = <AnyValue>		cmdview	send M_appendvert (1,P_temps,P_theta)
P_initialiser = <AnyValue>	⇒	posview	send M_init_curve (1)
P_initialiser = <AnyValue>	⇒	cmdview	send M_init_curve (1)
P_tableau_consigne = <AnyValue>	⇒	posview	send M_plottable (2,P_tableau_consigne)
P_CX = <AnyValue>	⇒	rail	send M_consigne (P_CX)
P_nb_Te1 = <AnyValue>	⇒	rail	send M_set_pos_et_angle (P_theta,P_pos_init_bille)
P_Type_Regulateur = <AnyValue>	⇒	Regulateur Selector	send M_selection (P_Type_Regulateur)
P_Seuil = <AnyValue>	⇒	Seuil	send M_selection (P_Seuil)
P_seuil_bool = <AnyValue>	⇒	rail	send M_seuil (P_seuil_bool)

Objet - Modèle

Objet	Condition sur une propriété ou sur l'état de l'objet	⇒	Action sur le modèle
Période d'échantillonnage	P_CurrentVal = <AnyValue>	⇒	send M_Te (P_CurrentVal)
Consigne	P_CurrentVal = <AnyValue>	⇒	send M_CX (P_CurrentVal)
Nombre d'échantillons	P_CurrentVal = <AnyValue>	⇒	send M_NbEch (P_CurrentVal)
bouton_demarrer	P_Trigger = <AnyValue>	⇒	send M_simuler ()
initButton	P_Trigger = <AnyValue>	⇒	send M_Init ()
Position initiale bille	P_CurrentVal = <AnyValue>	⇒	send M_pos_init_bille (P_CurrentVal)
Seuil	P_selection = <AnyValue>	⇒	send M_Seuil (P_selection)
Regulateur Selector	P_selection = <AnyValue>	⇒	send M_Type_Regulateur (P_selection)

5. Formulaires pour la spécification des simulations

Description générale

Représentation

Ecran

Interactions

Action	Effet

Modèle

Propriétés

Nom	Type	Publique	Description

Méthodes

Nom	Publique	Paramètres	Description

Graphe d'états

Actions des états

État	Action en entrée	Action en sortie	Action pendant	Événement Automatique

Associations

Modèle - Objet

Condition sur une propriété ou sur l'état du modèle	⇒	**Action sur un objet**
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	

Objet - Modèle

Condition sur une propriété ou sur l'état d'un objet	⇒	**Action sur le modèle**
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	
	⇒	

Scénarios

Description des scénarios

Pour décrire les scénarios créés, nous utiliserons un tableau qui donne la description des étapes et des contrôles. Le tableau contient les informations suivantes :

1. Situation à enregistrer : décrit l'état dans lequel doit se trouver la simulation pour faire la photo de l'étape ou du contrôle.
2. Situation attendue: décrit la situation dans laquelle l'élève doit mettre la simulation.
3. Situation à surveiller : décrit une situation particulière qui doit être repérée lorsque l'élève l'atteint au cours d'une étape.
4. Réactivité : décrit la réactivité de l'étape ou du contrôle. Montre uniquement les changements par rapport aux valeurs par défaut de OASIS.

Les informations 2 et 3 peuvent correspondre exactement à la situation enregistrée ou bien elles peuvent correspondre à une situation plus large, englobant la situation enregistrée.

No	Nom	Situation à enregistrer	Situation attendue (ou à surveiller)	Réactivité

No	Nom	Situation à enregistrer	Situation attendue (ou à surveiller)	Réactivité

Formulaire pour la description d'objets

Objet : _____

Les vues de l'objet :

Les vues :
Les composants graphiques :

Description :

Propriétés déclenchantes :

Nom	Type	Description

Méthodes publiques :

Nom	Paramètres	Description

Paramètrage par menu local :

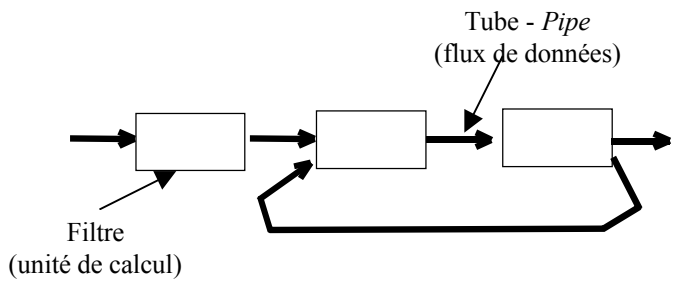
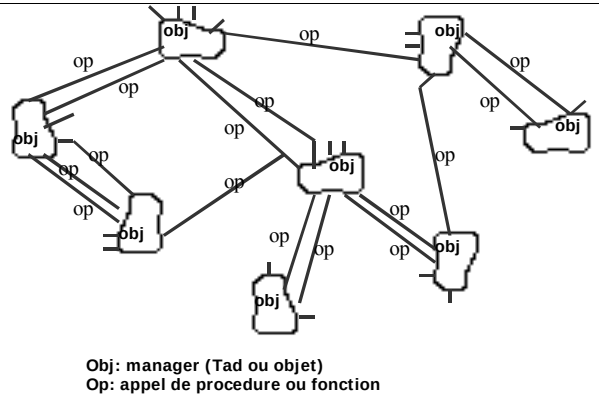
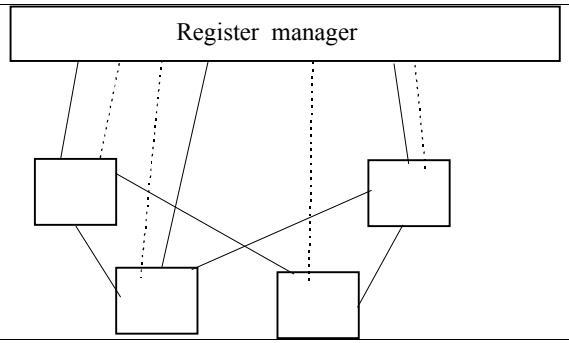
Item	Description

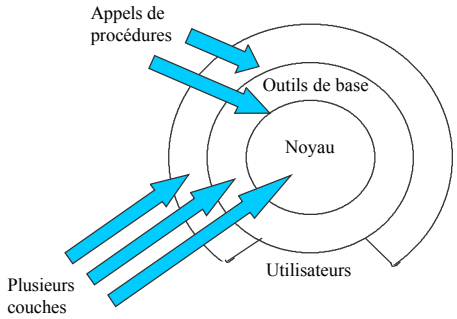
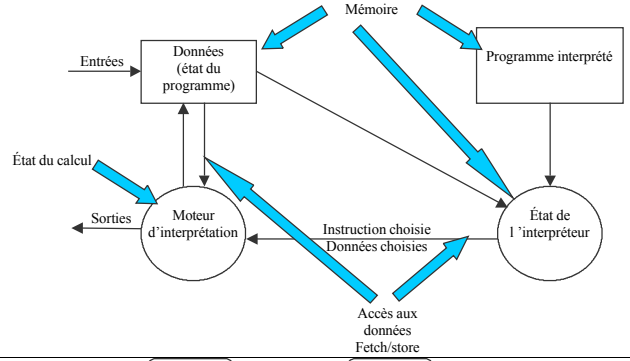
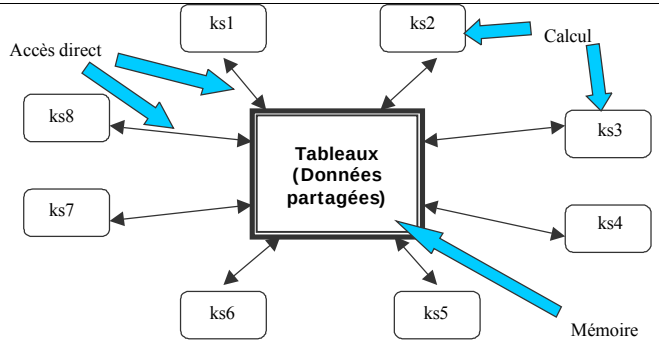
Annexe 2

STYLES D'ARCHITECTURE

Cette annexe fournit sous forme de tableaux récapitulatifs les styles d'architecture identifiés par Shaw et Garlan [SHA 96]. Nous faisons référence en particulier au style Interpréteurs dans le chapitre 6.

Styles d'architecture identifiés par Shaw et Garlan [SHA 96]

Style	Représentation graphique	Composants / Connecteurs	Caractéristiques
Pipes et Filtres Ex. Le shell d'UNIX.		**Composants** (Filtres) : un filtre est un composant de calcul ou de transformation qui a un ensemble d'entrées et un ensemble de sorties. Les entrées et les sorties sont fournies dans un format standard. **Connecteurs** (tubes, *pipes*) : Conduit pour passer les données entre filtres.	• Le système est conçu comme des transformations successives des entrées. • Les filtres peuvent être facilement changés et échangés. => réutilisation • Favorise les traitements séquentiels des données. • Non adaptés pour des applications interactives
Types Abstraits et Organisation Orientée par Objets		**Composants** (Objets ou types abstraits de données) : un composant est une entité que encapsule la représentation des données et les opérations sur ces données. **Connecteurs** (appels à fonctions ou procédures) **:** la connexion entre composants se fait par l'invocation des opérations des composants.	• Le système est conçu comme une collection d'objets qui interagissent. • L'encapsulation de la représentation favorise la réutilisation et maintenance des composants. • Pour interagir, les composants ont besoin de connaître l'identité des autres.
Architectures basées sur des événements et des appels implicites Ex : intégration d'outils (vérificateurs d'orthographe et grammaire, ou debuggers)		**Composants :** modules qui fournissent un ensemble d'opérations et un ensemble d'évènements. Chaque composant publie dans le manager de registres ses événements et souscrit aux événements qui l'intéressent. **Connecteurs :** appels implicites de procédures.	• Favorisent la réutilisation : pour ajouter un composant, il suffit d'enregistrer les événements du composant; pour utiliser un composant, on ne doit pas connaître son identité; les changements des interfaces des composants n'affectent pas les autres composants,… • Les composants ne peuvent pas prévoir le déroulement du contrôle.

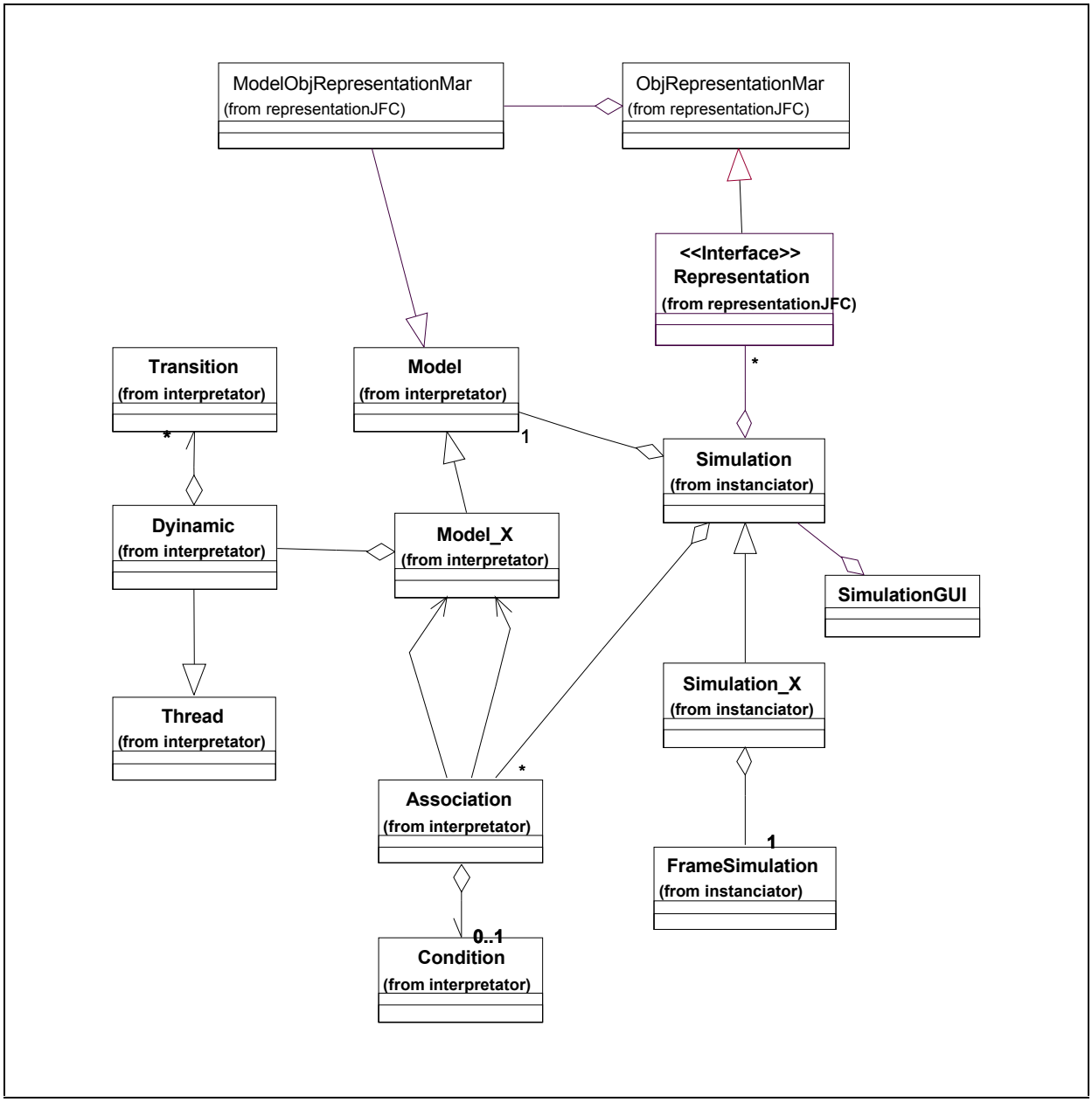
Architectures par couches Ex: Architecture réseaux de l'OSI		**Composants** : Les couches. Chaque couche est cliente de la couche au-dessous et est serveur de la couche au-dessus. Les couches interagissent seulement avec les couches voisines. **Connecteurs** : Les protocoles de communication entre couches.	• Le concepteur peut concevoir les systèmes par niveaux d'abstraction. • Une couche peut être remplacée par une autre qui offre les mêmes services sans affecter les autres couches. • La conception des systèmes par niveaux n'est pas toujours évidente.
Interpréteurs Ex : Machine Virtuelles		**Composants :** le moteur d'interprétation, le programme interprété, l'état du programme, l'état du contrôle de l'interpréteur. **Connecteurs :** flux de données.	• L'architecture d'interpréteur réalise en logiciel une machine virtuelle pour un langage.
Dépôts (*Repositories)* Ex: Bases de données, tableaux noirs		**Composants :** un dépôt des données qui stocke l'état du système et une collection de composants indépendants qui utilisent les données. **Connecteurs** : l'accès aux données.	• Le contrôle du système définit le type de l'architecture : si le déclenchement des composants indépendants est déterminé par l'état du dépôt, alors l'architecture est un tableau. Si le déclenchement des composants indépendants est déterminé par un flux de transactions, alors l'architecture est une base de données.

ANNEXE 3

EXEMPLES D'APPLICATIONS DES PATRONS DE CONCEPTION DANS FREAK

1. Diagramme de classes d'une simulation X

La figure suivante donne le schéma de classes des simulations MARS-S produites par le générateur de FREAK.



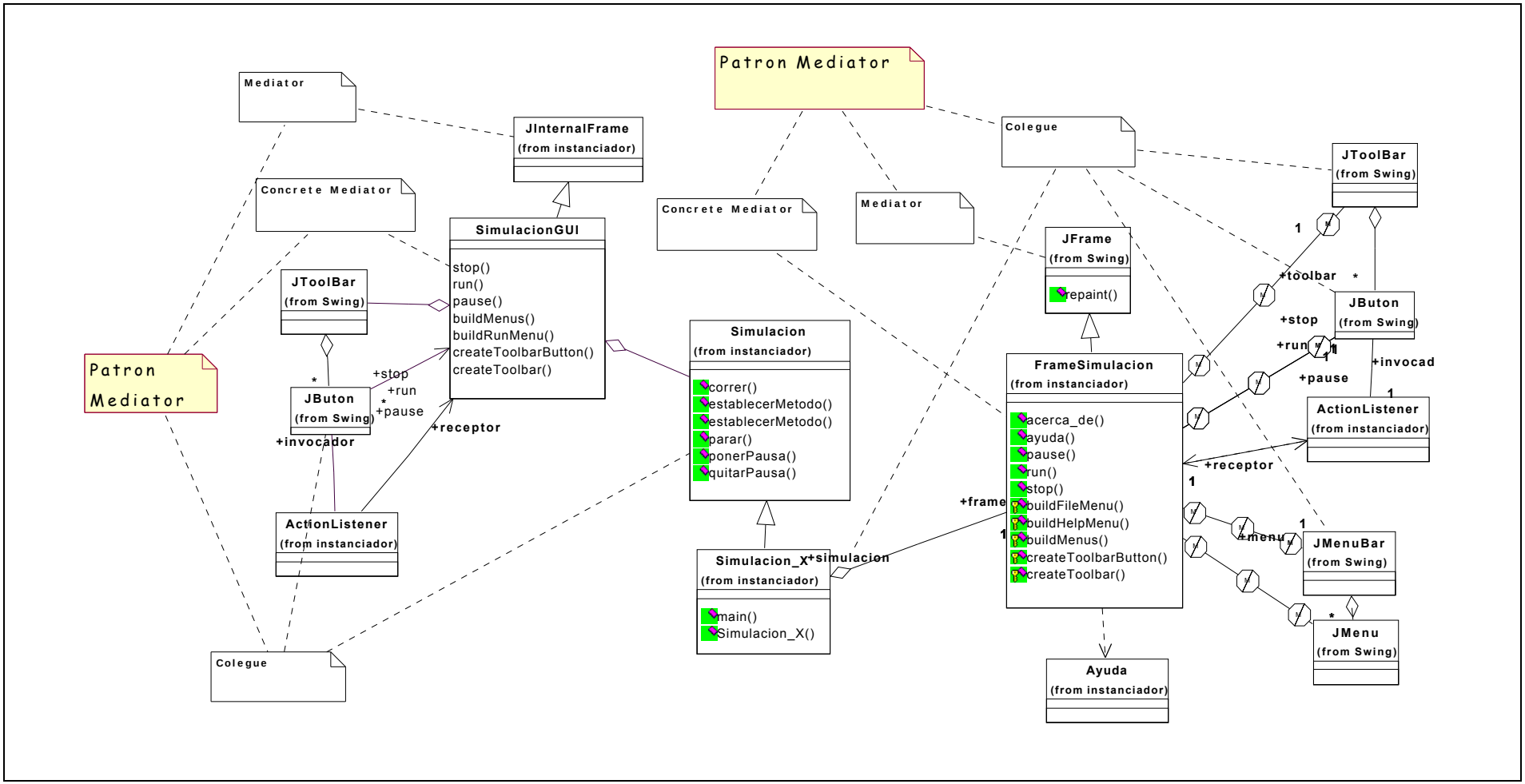
2. Application du patron Mediator

Problème:

Gestion d'interface. La simulation générée doit pouvoir s'exécuter comme une application indépendante ou doit pouvoir s'intégrer dans une fenêtre de test.

Solution: Application itérative du patron *Mediator* [GAM 94] plus délégation.

Le conteneur (*frame*) de l'interface d'une simulation (SimulacionGui) est un *Mediator* qui gère les boutons `stop`, `run` et `pause` et la simulation (Simulacion). Le conteneur qui permet l'exécution indépendante de la simulation est aussi un *Mediator* qui coordonne les éléments d'interface du menu et délègue au Mediator SimulacionGui les actions sur les boutons `stop`, `run` et `pause`. Le système de notification des changements des objets coordonnés est basé sur l'architecture des beans de Java.


Patron Mediator
Mediator
JInternalFrame
(from instanciador)
Concrete Mediator
SimulacionGUI
stop()
run()
pause()
buildMenus()
buildRunMenu()
createToolbarButton()
createToolbar()
JToolBar
(from Swing)
Patron
Mediator
JButon
(from Swing)
+stop
+run
+pause
+invocador
+receptor
ActionListener
(from instanciador)
Colegue
Patron Mediator
Concrete Mediator
Mediator
Colegue
Simulacion
(from instanciador)
correr()
establecerMetodo()
establecerMetodo()
parar()
ponerPausa()
quitarPausa()
Simulacion_X
(from instanciador)
main()
Simulacion_X()
+simulacion
+frame
JFrame
(from Swing)
repaint()
FrameSimulacion
(from instanciador)
acerca_de()
ayuda()
pause()
run()
stop()
buildFileMenu()
buildHelpMenu()
buildMenus()
createToolbarButton()
createToolbar()
Ayuda
(from instanciador)
JToolBar
(from Swing)
+toolbar
JButon
(from Swing)
+stop
+run
+pause
+invocad
ActionListener
(from instanciador)
+receptor
+menu
JMenuBar
(from Swing)
JMenu
(from Swing)

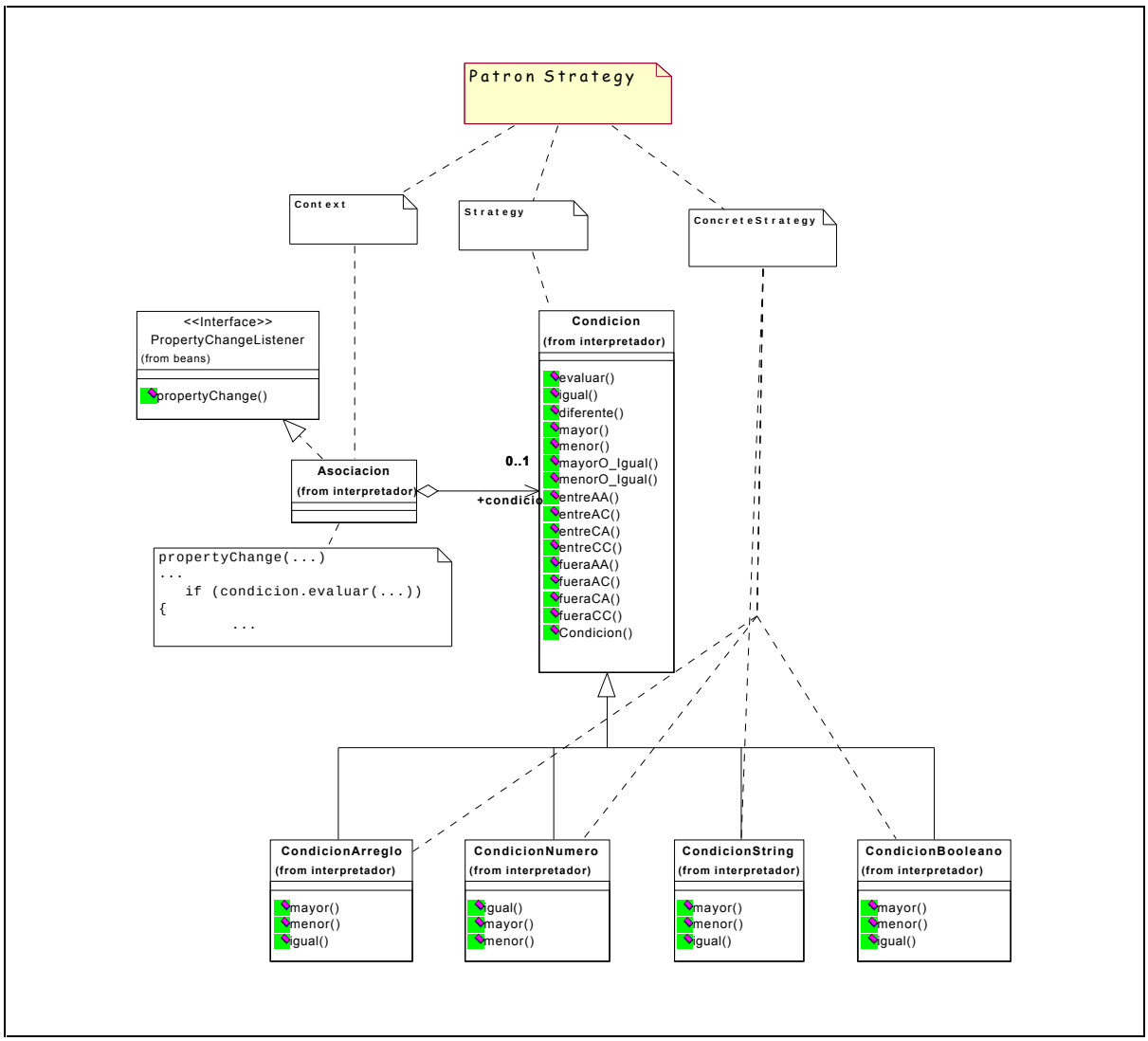
3. Application du patron Strategy

Problème :

Les conditions des associations dépendent du type de la propriété sur laquelle la condition est appliquée. Nous voulons avoir la possibilité d'ajouter d'autres types sans affecter les conditions existantes.

Solution : Application du patron *Strategy* [GAM 94]

L'association contient une référence à la condition et lui délègue l'évaluation. Une condition offre une interface commune pour les opérations d'évaluation. Des sous-classes de condition implémentent les opérations pour les types spécifiques. Pour ajouter un type de propriété XX, il suffit de construire une classe CondicionXX qui implémente les opérations et de faire une relation d'héritage avec la classe Condicion.



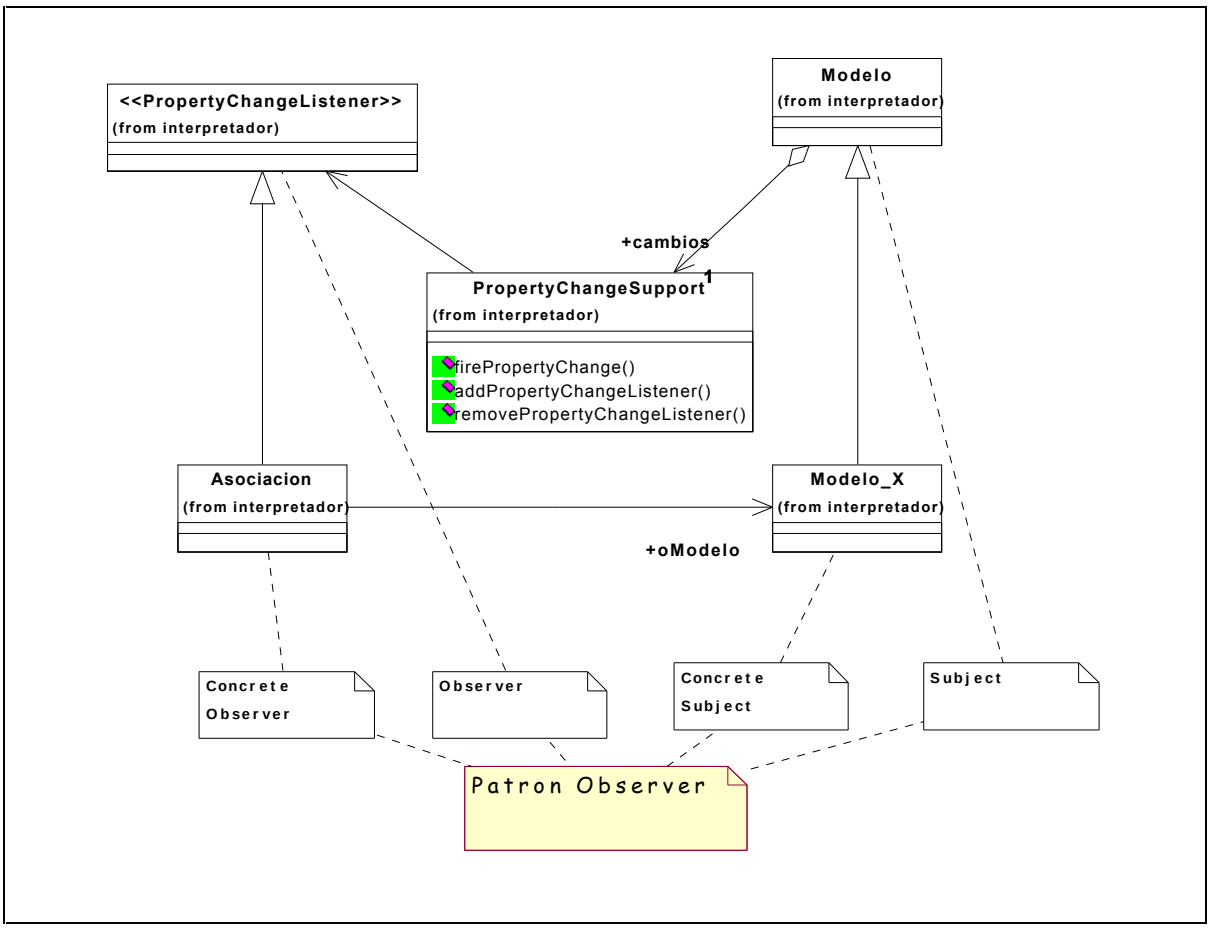
3. Application du patron Observer

Problème

Les associations jouent un rôle important dans le modèle MARS-S. Elles sont la base pour l'exécution de la simulation (cf. description de l'interpréteur). Deux questions de conception surgissent par rapport aux associations. La première est comment notifier les changements du modèle et des objets de présentation afin d'activer les associations. La deuxième est comment déclencher toutes les associations pertinentes lors de l'arrivée d'une notification. Les outils MELISA et OASIS appliquent le modèle MVC (Model-View-Controler) pour résoudre ces problèmes. Les changements de propriétés ou d'état du modèle (M) et des objets de présentation (V) sont signalés au gestionnaire des associations (C) qui maintient la cohérence entre le modèle et la visualisation. L'application du modèle MVC dans MELISA et OASIS est centralisé, c'est-à-dire qu'il n'existe qu'un contrôleur. La notification s'appuie sur le mécanisme de notification de ToolBook qui permet de détecter le changement de valeur d'une variable. Le déclenchement se fait à l'aide d'une liste d'associations qui est parcourue pour chercher où intervient la variable. Avec cette solution, l'augmentation du nombre d'associations dégrade la performance de la simulation.

Solution: Application du patron *Observer* [GAM 94] pour chaque association.

Le patron *Observer* sert justement à appliquer le modèle MVC dans les conceptions orientées objets. Les modèles (Modelo_X) sont des objets observables et les associations (Asociacion) sont des observateurs. L'observateur doit manifester son intérêt pour les objets observables et doit être capable de traiter les notifications. Chaque objet observable est responsable de maintenir une liste de ses observateurs et de les notifier quand il y a un changement. Dans cette solution, la responsabilité d'activer les associations est distribuée dans les objets de la classe Modelo, ce qui supprime toute recherche dans une liste d'associations. Pour les notifications, on s'appuie sur l'architecture des Java beans: la classe Modelo utilise la gestion d'observateurs et de notifications de la classe PropertyChangeSupport du package Beans de Java.


<<PropertyChangeListener>>
(from interpretador)
Modelo
(from interpretador)
+cambios
1
PropertyChangeSupport
(from interpretador)
firePropertyChange()
addPropertyChangeListener()
removePropertyChangeListener()
Asociacion
(from interpretador)
Modelo_X
(from interpretador)
+oModelo
Concrete
Observer
Observer
Concrete
Subject
Subject
Patron Observer

ANNEXE 4
SPECIFICATIONS D'ARGOS EN UNICON

Cette annexe introduit le langage textuel d'UNICON sur l'exemple simple donné seulement sous forme graphique au début du chapitre 8 et fournit ensuite sous cette forme textuelle les spécifications de l'architecture ARGOS.

1. Exemple de spécification dans le langage textuel d'UNICON

Afin d'introduire de manière simple, les notations du langage textuel d'UNICON, nous reprenons sous cette nouvelle forme l'exemple donné, sous forme graphique seulement, dans le paragraphe 8.1.1.

Description de composants.

Nous ne montrons ici que la description du composant A. Les composants B et C ont des descriptions similaires.

```
COMPONENT A
 INTERFACE IS
  TYPE      Module
  PLAYER Rfile IS ReadFile
    SIGNATURE ("struct datain *")
  END RFile
  PLAYER B1Call IS RoutineCall
    SIGNATURE ("struct datain *";"struct dataout *")
  END B1Call
  PLAYER B2Call IS RoutineCall
    SIGNATURE ("struct datain *";int)
  END B2Call
  PLAYER S1Call IS RoutineCall
    SIGNATURE ("struct dataout *")
  END S1Call
 END INTERFACE
IMPLEMENTATION IS
 VARIANT componentA IN A.c
  IMPLTYPE(source)
 END componentA
END A
```

Description du connecteur

```
CONNECTOR proc-call
 PROTOCOL IS
  TYPE      ProcedureCall
  ROLE def IS Definer
  ROLE caller IS Caller
 END PROTOCOL
IMPLEMENTATION IS
 BUILT IN // Fourni par UNICON
END proc-call
```

Description du système

```
COMPONENT allSystem
 INTERFACE IS
 //Ce composant n'exporte pas de services
 END INTERFACE
IMPLEMENTATION IS
 USES CompA INTERFACE A
 USES CompB INTERFACE B
 USES CompC INTERFACE C
 USES Data INTERFACE DataFile

 ESTABLISH proc-call WITH
  CompA.CallB1 AS caller
  CompB.DefB1 AS def
 END proc-call

 ESTABLISH proc-call WITH
  CompA.CallB2 AS caller
  CompB.DefB2 AS def
 END proc-call

 ESTABLISH proc-call WITH
  CompA.CallS1 AS caller
  CompC.DefS1 AS def
 END proc-call

 ESTABLISH fileIO WITH
  CompA.RFile AS reader
  Data.DefFile AS definer
 END fileIO
END allSystem
```

2. Spécification d'ARGOS

Description générale d'ARGOS

```
COMPONENT    Pedagogical_Simulation (CPProtocol,SimProtocol)
  INTERFACE IS
       TYPE General
  END INTERFACE
  IMPLEMENTATION IS

    USES client INTERFACE CPClient
    USES server INTERFACE SimServer(SimProtocol)

    ESTABLISH ARGOSConnector(CPProtocol) WITH
     server.connectionBundler AS participant
     client.connectionBundler AS participant

           MATCH (
      (client.connectionBundler.init_connection,
      server.connectionBundler.init_connection)

           (client.connectionBundler.close_connection,
           server.connectionBundler.close_connection)
    END ARGOSConnector

    ESTABLISH ARGOSConnector(CPProtocol) WITH
```

```
        server.dataBundler AS participant
        client.dataBundler AS participant

        MATCH (
        (client.dataBundler.get_variables_list,
        server.dataBundler.get_variables_list)

        (client.dataBundler.get_variables_values,
        server.dataBundler.get_variables_values)

        (client.dataBundler.get_list_variables_values,
        server.dataBundler.get_list_variables_values)

        (client.dataBundler.set_variables_values,
        server.dataBundler.set_variables_values)

        (client.dataBundler.set_list_variables_values,
        server.dataBundler.set_list_variables_values)

        (client.dataBundler.set_variable_visibility,
        server.dataBundler.set_variable_visibility)

        (client.dataBundler.set_variable_modifiability,
        server.dataBundler.set_variable_modifiability)

        (client.dataBundler.notify_variable,
        server.dataBundler.notify_variable)

        (client.dataBundler.init_monitoring_variable,
        server.dataBundler.init_monitoring_variable)

        (client.dataBundler.stop_monitoring_variable,
        server.dataBundler.stop_monitoring_variable)
END ARGOSConnector

ESTABLISH ARGOSConnector(CPProtocol) WITH
        server.actionsBundler AS participant
        client.actionsBundler AS participant

        (client.actionsBundler.get_actions_list,
        server.actionsBundler.get_actions_list)

        (client.actionsBundler.execute_action,
        server.actionsBundler.execute_action)

        (client.actionsBundler.notify_action,
        server.actionsBundler.notify_action)

END ARGOSConnector

ESTABLISH ARGOSConnector(CPProtocol) WITH
        server.componentBundler AS participant
        client.componentBundler AS participant

        (client.componentBundler.get_components_list
        server.componentBundler.get_components_list)

        (client.componentBundler.point_element,
        server.componentBundler.point_element)

        (client.componentBundler.set_element_highligthing,
        server.componentBundler.set_element_highligthing)
END ARGOSConnector

ESTABLISH ARGOSConnector(CPProtocol) WITH
        server.stateBundler AS participant
        client.stateBundler AS participant
```

```
            (client.stateBundler.get_simulation_state,
            server.stateBundler.get_simulation_state)

            (client.stateBundler.set_simulation_state
            server.stateBundler.set_simulation_state)
            END ARGOSConnector
```

Description du composant CPClient

```
COMPONENT   CPClient
      INTERFACE IS
            TYPE  Process
            PLAYER connectionBundler is PLBundler
                  MEMBER (init_connection; RPCCall;
                    SIGNATURE ("Simulation_name";list_of_strings))
                  MEMBER (close_connection; RPCCall
                    SIGNATURE ("Simulation_name";int))
            END connectionBundler

            PLAYER dataBundler is  PLBundler
                  MEMBER (get_variables_list; RPCCall;
                    SIGNATURE (; list_of_strings))
                  MEMBER (get_variables_values;RPCCall;
                    SIGNATURE (; list_of_variables))
                  MEMBER (get_list_variables_values;RPCCall;
                    SIGNATURE (list_of_strings; list_of_variables))
                  MEMBER (set_variables_values;RPCCall;
                    SIGNATURE (list_of_variables;))
                  MEMBER (set_list_variables_values;RPCCall;
                    SIGNATURE (list_of_variables;)
                  MEMBER (set_variable_visibility;RPCCall;
                    SIGNATURE (string;int))
                  MEMBER (set_variable_modifiability;RPCCall;
                    SIGNATURE (string;int))
                  MEMBER (notify_variable;RPCDef;
                    SIGNATURE (string;))
                  MEMBER (init_monitoring_variable;RPCCall;
                    SIGNATURE (string;int))
                  MEMBER (stop_monitoring_variable;RPCCall;
                    SIGNATURE (string;int))
            END dataBundler

            PLAYER actionsBundler is  PLBundler
                  MEMBER (get_actions_list;RPCCall;
                    SIGNATURE (;list_of_actions))
                  MEMBER (execute_action;RPCCall;
                    SIGNATURE (string;int))
                  MEMBER (notify_action;RPCDef;
                    SIGNATURE (string;))
            END actionsBundler

            PLAYER componentBundler is  PLBundler
                  MEMBER (get_component_list;RPCCall;
                    SIGNATURE (;component_tree))
                  MEMBER (point_to;RPCCall;
                    SIGNATURE (string;int))
                  MEMBER (set_element_highlighting;RPCCall;
                    SIGNATURE (string;int))
```

```
            END componentBundler

            PLAYER stateBundler is  PLBundler
                  MEMBER (get_simulation_state;RPCCall;
                    SIGNATURE (;state))
                  MEMBER (set_simulation_state;RPCCall;
                    SIGNATURE (;state))
            END stateBundler
      END INTERFACE
      IMPLEMENTATION IS
            VARIANT CPClient IN « CPClient.exe »
                  IMPLTYPE (Executable)
            END CPClient
      END IMPLEMENTATION
END CPClient
```

Description du composant SimServer

```
COMPONENT   SimServer (SimProtocol)
      INTERFACE IS
            TYPE  Process
            PLAYER connectionBundler is PLBundler
                  MEMBER (init_connection; RPCDef;
                    SIGNATURE (“Simulation_name”;list_of_strings))
                  MEMBER (close_connection; RPCDef;
                    SIGNATURE (“Simulation_name”;int))
            END connectionBundler

            PLAYER dataBundler is  PLBundler
                  MEMBER (get_variables_list; RPCDef;
                    SIGNATURE (; list_of_strings))
                  MEMBER (get_variables_values;RPCDef;
                    SIGNATURE (; list_of_variables))
                  MEMBER (get_list_variables_values;RPCDef;
                    SIGNATURE (list_of_strings; list_of_variables))
                  MEMBER (set_variables_values;RPCDef;
                    SIGNATURE (list_of_variables;))
                  MEMBER (set_list_variables_values;RPCDef;
                    SIGNATURE (list_of_variables;)
                  MEMBER (set_variable_visibility;RPCDef;
                    SIGNATURE (string;int))
                  MEMBER (set_variable_modifiability;RPCDef;
                    SIGNATURE (string;int))
                  MEMBER (notify_variable;RPCCall;
                    SIGNATURE (string;))
                  MEMBER (init_monitoring_variable;RPCDef;
                    SIGNATURE (string;int))
                  MEMBER (stop_monitoring_variable;RPCDef;
                    SIGNATURE (string;int))
            END dataBundler

            PLAYER actionsBundler is  PLBundler
                  MEMBER (get_actions_list;RPCDef;
                    SIGNATURE (;list_of_actions))
                  MEMBER (execute_action;RPCDef;
                    SIGNATURE (string;int))
                  MEMBER (notify_action;RPCCall;
```

```
                  SIGNATURE (string;))
            END actionsBundler

            PLAYER componentBundler is  PLBundler
                  MEMBER (get_component_list;RPCDef;
                    SIGNATURE (;component_tree))
                  MEMBER (point_to;RPCDef;
                    SIGNATURE (string;int))
                  MEMBER (set_element_highlighting;RPCDef;
                    SIGNATURE (string;int))
            END componentBundler

            PLAYER stateBundler is  PLBundler
                  MEMBER (get_simulation_state;RPCDef;
                    SIGNATURE (;state))
                  MEMBER (set_simulation_state;RPCDef;
                    SIGNATURE (;state))
            END stateBundler
      END INTERFACE
      IMPLEMENTATION IS
      //Ce composant est composé. Alors ici il faut décrire comment
      //on construit le composant.
            USES adapter(SimProtocol) INTERFACE  ARGOS-Adapter
            USES simulation INTERFACE  ARGOS-Simulation

            BIND connectionBundler TO adapter.connectionBundler
            BIND dataBundler TO adapter.dataBundler
            BIND actionsBundler TO adapter.actionsBundler
            BIND componentBundler TO adapter.componentBundler
            BIND stateBundler TO adapter.stateBundler



      END IMPLEMENTATION
END CPClient
```

Description du connecteur

```
CONNECTOR ARGOSConnector (Protocol)
      PROTOCOL IS
            Type PLBundler
            ROLE participant
      END PROTOCOL
      IMPLEMENTATION IS
            VARIANT Protocol == RMI IN ArgosRMIConnector.class
                  IMPLTYPE (Executable)
            END Protocol==RMI
            VARIANT Protocol == COM IN ArgosCOMConnector.class
                  IMPLTYPE (Executable)
            END Protocol==COM
            VARIANT Protocol == CORBA IN ArgosCORBAConnector.class
                  IMPLTYPE (Executable)
            END Protocol==CORBA
      END
END ARGOSConnector
```

ANNEXE 5
UTILISATION DU PROTOTYPE SWIPS

Cette annexe contient un petit texte distribué à l'équipe ARCADE pour indiquer comment mettre en œuvre le prototype SWIPS. On y voit plus en détail que dans le chapitre 8 ce que peuvent faire le coordinateur de session, l'enseignant et les étudiants pendant une séance avec SWIPS.

Description générale de l'architecture

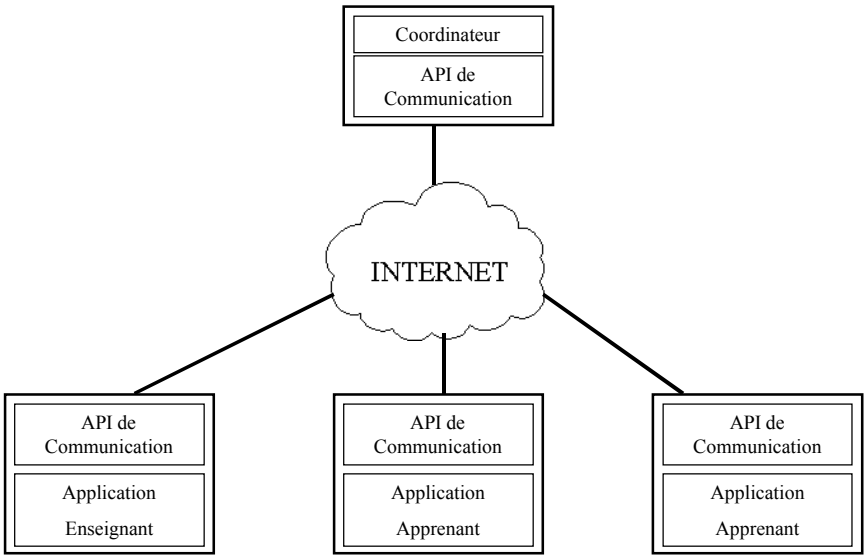
L'architecture de communication a été conçue pour permettre de faire «un exercice», à distance, de façon synchchrone, dans une séance de TD/TP. Les participants à la séance sont :

- L'enseignant qui est chargé de diriger la séance.
- Les étudiants qui font l'exercice proposé par l'enseignant.

Dans ce travail, nous réduisons les outils de communication entre l'enseignant et les étudiants à une communication textuelle, style chat.

Nous avons conçu une architecture de communication indépendant du type d'exercice à faire, car nous voulions séparer le problème de la communication du problème du contrôle pédagogique de l'exercice.

La figure montre la structure de l'architecture proposée :



Nous avons adopté le schéma, le plus simple, pour la communication dans les applications de collaboration et coopératives, le schéma centralisé. Dans ce schéma, il y a un coordinateur qui gère l'information commune, et des utilisateurs qui utilisent l'information. Dans ce cas, nous avons deux types d'utilisateurs.

Parmi les fonctions du coordinateur, nous pouvons mentionner : le contrôle de l'accès au système ; le stockage et la mise à jour de l'information partagée par les utilisateurs.

Le déroulement typique d'une séance est le suivant :

- Le coordinateur est actif en tout moment.
- L'enseignant et les étudiants se connectent au système, à un cours déterminé, et commencent une séance. Le coordinateur donne accès aux utilisateurs inscrits pour le cours. Les étudiants ne peuvent pas démarrer l'exercice sans l'autorisation de l'enseignant.
- Quand l'enseignant voit que tous ces étudiants sont arrivés à la séance, il démarre l'exercice.
- Les étudiants font l'exercice, l'enseignant suit le travail des étudiants, en regardant leurs exercices ou à travers les contrôles automatiques définis dans l'exercice.

Dans cette version du prototype, l'apprenant peut se connecter à une séance pour travailler avec la simulation libre ou faire deux exercices avec des contrôles pédagogiques synchrones et non asynchrones.

L'enseignant peut bloquer ou débloquer les participants, observer leur travail à un moment donné ou de façon continue, observer un rapport d'activités des participants ou observer un rapport sur l'exercice que les participants font.

Les applications

Le système a deux composants, l'application pour les utilisateurs - les apprenants et l'enseignant- et l'application du coordinateur. L'application du coordinateur doit être en exécution tout le temps pendant la séance, car c'est le serveur. L'adresse IP, ou le nom, de la machine qui exécute l'application du coordinateur, doit être connue.

Installation des applications (dans chaque machine)

1. L'ordinateur doit avoir Java 1.1.4 installé.
2. Créer un répertoire pour l'application.
3. Dans ce répertoire, décomprimer le fichier swips.zip.

4. Changer le fichier Host.txt avec l'adresse IP, ou le nom de la machine, où l'application du coordinateur va être exécutée.

   Par exemple : `pictionnary.imag.fr`

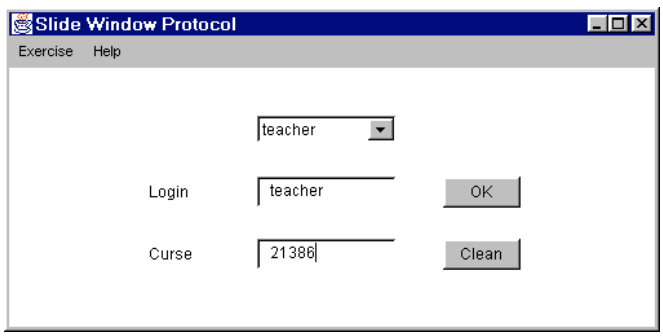
Exécution de l'application du coordinateur

1. Sur DOS, allez au répertoire de l'application.
2. Exécutez la commande : coord
3. Réduire la fenêtre de DOS

Connexion comme enseignant

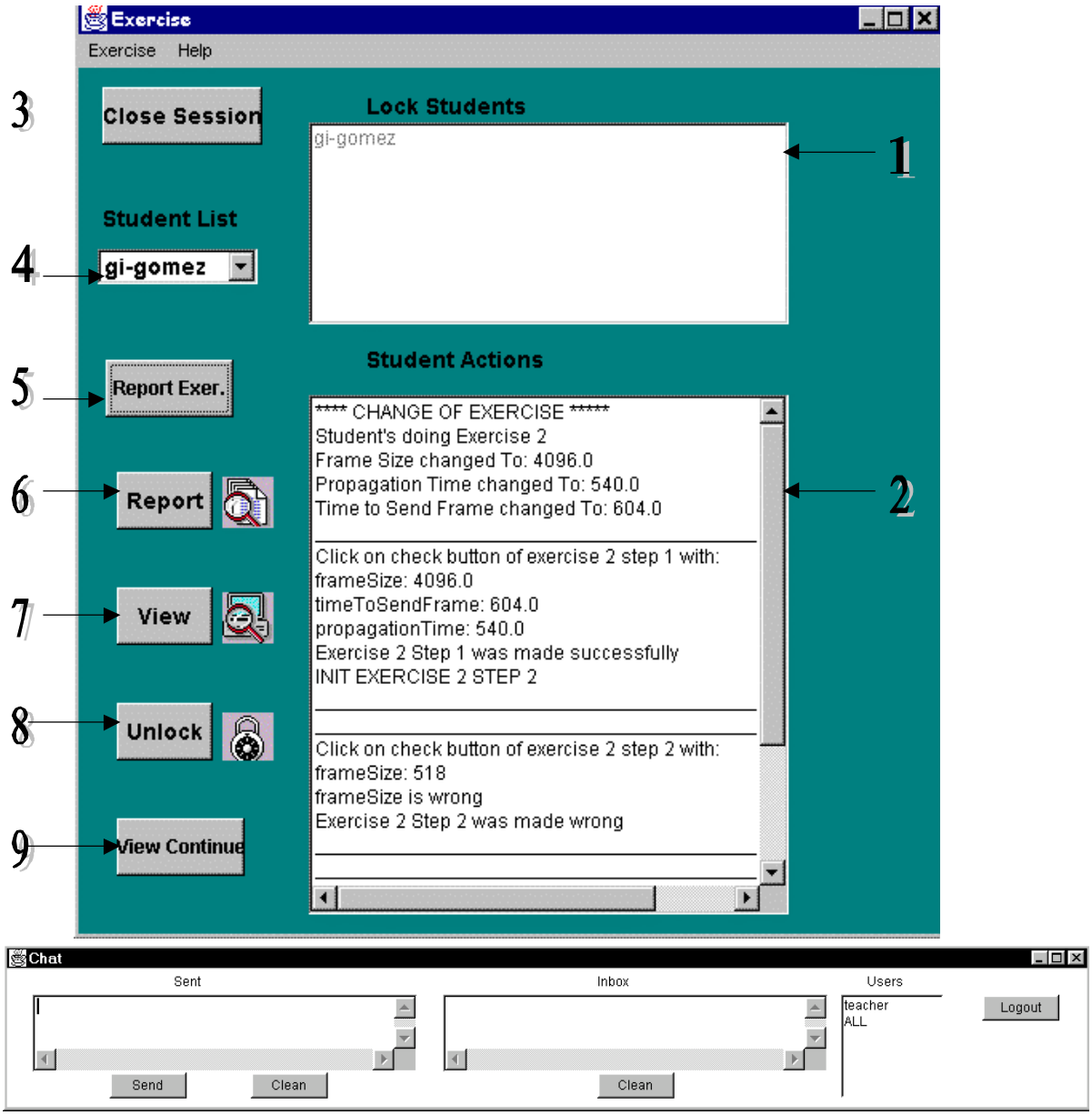
1. Sur DOS, allez au répertoire de l'application.
2. Exécutez la commande : sim

L'enseignant doit d'abord faire un login.



Et appuyer sur OK

L'image montre l'écran de l'enseignant. L'enseignant a une fenêtre pour contrôler les exercices et une fenêtre pour la communication.



- Après, il attend que tous ses étudiants arrivent.
- Quand ils sont arrivés, il démarre la séance.

  `Appuyer sur Open/Close Session (3)`

- S'il veut regarder le travail d'un étudiant à un instant donné.

  `il doit sélectionner l'étudiant dans la liste`

  `Appuyer sur View (7)`

Une fenêtre avec la simulation de l'apprenant s'ouvre.

Pour finir d'observer, l'enseignant doit appuyer sur End View (7).

- S'il veut regarder le travail d'un étudiant continûment.

  il doit sélectionner l'étudiant dans la liste.

  Appuyer sur View Continue (9).

  Une fenêtre avec la simulation de l'apprenant s'ouvre.

  Pour finir d'observer, l'enseignant doit appuyer sur Stop View (9).

- S'il veut bloquer un étudiant.

  il doit sélectionner l'étudiant dans la liste.

  Appuyer sur Lock (8).

- S'il veut débloquer un étudiant.

  il doit sélectionner l'étudiant dans la liste.

  Appuyer sur UnLock (8).

- S'il veut observer le rapport d'activité d'un étudiant.

  il doit sélectionner l'étudiant dans la liste.

  Appuyer sur Report (6).

- S'il veut observer le rapport d'exercice d'un étudiant.

  il doit sélectionner l'étudiant dans la liste.

  Appuyer sur Exercise Report (5).

- S'il veut envoyer un message à un étudiant (ou à tous).

  il doit sélectionner l'étudiant dans la liste d'utilisateurs de la fenêtre de communication (ou all).

  il doit écrire le message dans le champ Send.

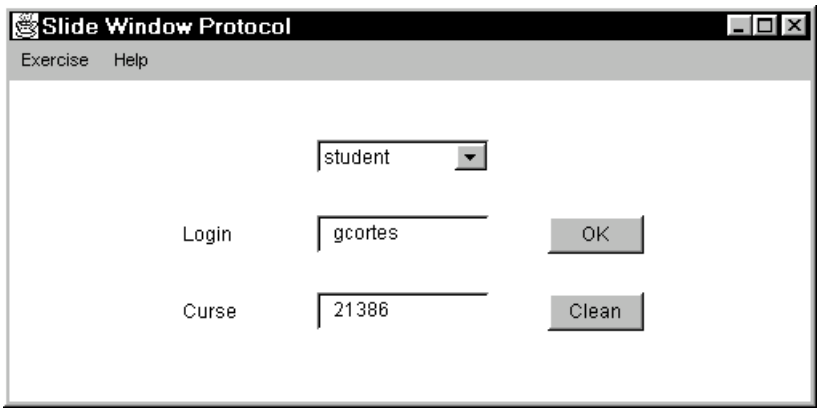
```
Appuyer sur Send.
```

Connexion comme étudiant

1. Sur DOS, allez au répertoire de l'application.

2. Exécutez la commande : sim

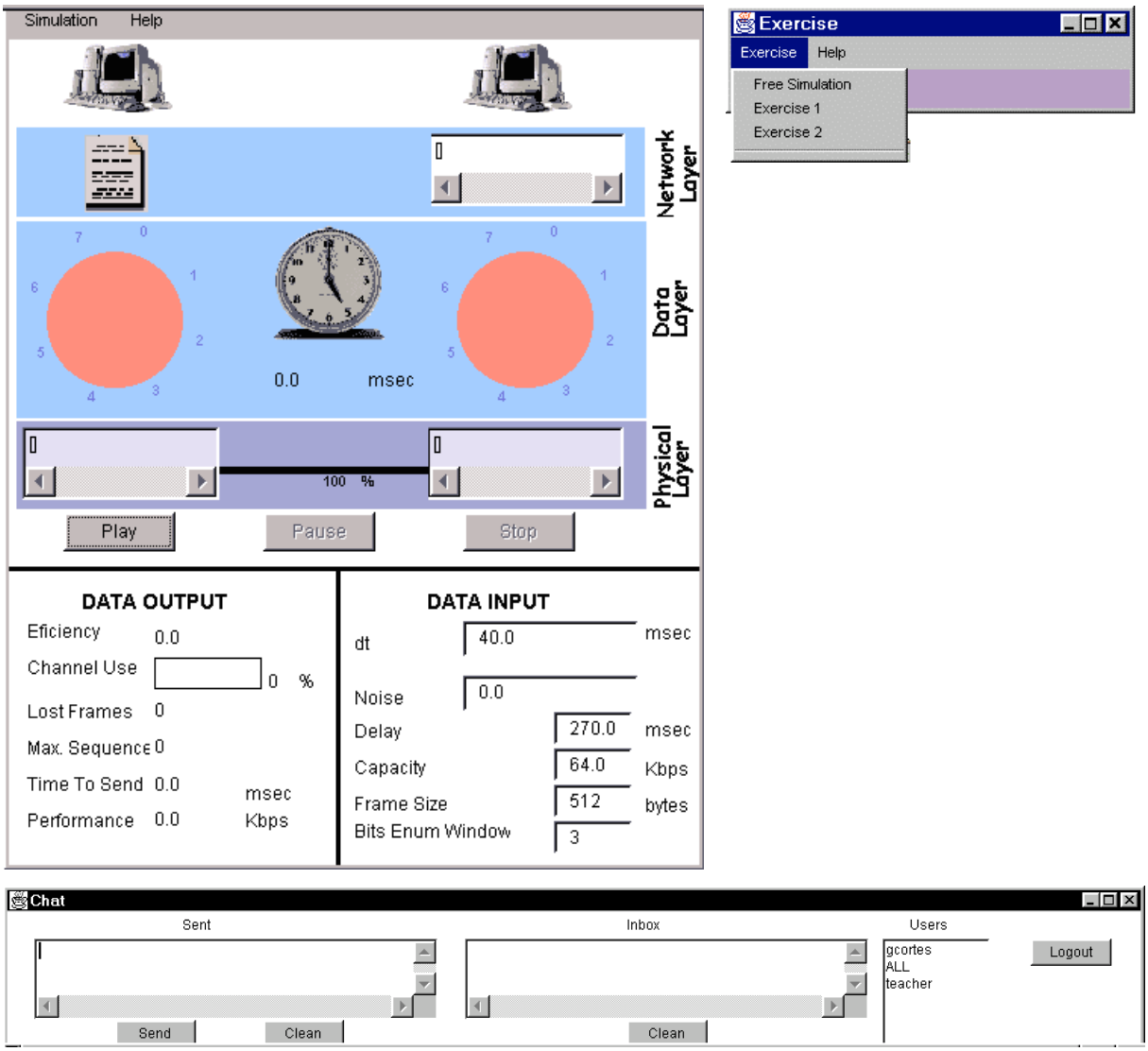
   - L'apprenant doit d'abord faire un login. Les apprenants existants sont : `gcortes, jph-pernin et jmadam`



3. L'interface de l'étudiant a trois fenêtres : une fenêtre avec la simulation, une fenêtre pour le contrôle pédagogique et une fenêtre de communication.

   Dans la fenêtre de la simulation, l'apprenant peut changer les variables du protocole et voir comment un ficher est transmis en utilisant le protocole *slide window* et comment la ligne de communication se comporte selon les paramètres fixés.

   La figure suivante montre l'interface de l'étudiant :



- Après il doit sélectionner un exercice à faire. Ici, il y a seulement deux exercices et la simulation libre. Le premier exercice a une étape et le deuxième a deux étapes.

- S'il veut envoyer un message à l'enseignant (ou à tous) :

  ```
  il doit sélectionner l'étudiant dans la liste d'utilisateurs de
  la fenêtre de communication (ou all).

  il doit écrire le message dans le champ Send.

  Appuyer sur Send.
  ```

Note : Quand l'apprenant est bloqué par l'enseignant, il ne peut pas interagir avec la simulation.

Pour changer une valeur de la simulation il faut appuyer sur Entrée.

Description des exercices :

- Exercice 1 : (1 étape)

  Objectif : Calculer les valeurs Frame size, Time to send et Propagation Time.

  Etat final :

  Frame Size = 4096 (Après deux échecs , synchrone)

  Time to send = 604 (Après quatre échecs, synchrone)

  Propagation time = 540 (Après trois échecs, synchrone)

  Contrôles locaux (non transmis de façon synchrone à l'enseignant) :

  Frame Size ≠ 4096 (Après deux échecs , synchrone)

  Time to send ≠ 604 (Après quatre échecs, synchrone)

  Propagation time ≠ 540 (Après trois échecs, synchrone)

  Contrôles synchrones :

  dt >0

- Exercise 2 (2 étapes)

  Etape 1

  Objectif : Calculer les valeurs Frame size, Time to send et Propagation Time.

  Etat final :

  Frame Size = 4096 (Après deux échecs , synchrone)

  Time to send = 604 (Après quatre échecs, synchrone)

  Propagation time = 540 (Après trois échecs, synchrone)

  Contrôles locaux (non transmis de façon synchrone à l'enseignant) :

Frame Size ≠ 4096 (Après deux échecs, synchrone)

Time to send ≠604 (Après quatre échecs, synchrone)

Propagation time ≠540 (Après trois échecs, synchrone)

Contrôles synchroniques :

dt >0

Etape 2

Objectif : Trouver une valeur pour la taille du paquet (Frame Size) pour obtenir une utilisation de la ligne entre 70% et 80%

Etat final :

202 < Frame Size < 233

Contrôles locaux (non transmis de façon synchrone à l'enseignant) :

dt >0

Frame Size <=0 (Après trois échecs, synchrone)